# GraphCast: Learning skillful medium-range global weather forecasting

**Remi Lam**[*,1], **Alvaro Sanchez-Gonzalez**[*,1], **Matthew Willson**[*,1], **Peter Wirnsberger**[*,1], **Meire Fortunato**[*,1], **Alexander Pritzel**[*,1], **Suman Ravuri**[1], **Timo Ewalds**[1], **Ferran Alet**[1], **Zach Eaton-Rosen**[1], **Weihua Hu**[1], **Alexander Merose**[2], **Stephan Hoyer**[2], **George Holland**[1], **Jacklynn Stott**[1], **Oriol Vinyals**[1], **Shakir Mohamed**[1] **and Peter Battaglia**[1]

[*]equal contribution, [1]DeepMind, [2]Google



**We introduce a machine-learning (ML)-based weather simulator—called "GraphCast"—which outperforms the most accurate deterministic operational medium-range weather forecasting system in the world, as well as all previous ML baselines. GraphCast is an autoregressive model, based on graph neural networks and a novel high-resolution multi-scale mesh representation, which we trained on historical weather data from the European Centre for Medium-Range Weather Forecasts (ECMWF)'s ERA5 reanalysis archive. It can make 10-day forecasts, at 6-hour time intervals, of five surface variables and six atmospheric variables, each at 37 vertical pressure levels, on a 0.25° latitude-longitude grid, which corresponds to roughly $25 \times 25$ kilometer resolution at the equator. Our results show GraphCast is more accurate than ECMWF's deterministic operational forecasting system, HRES, on 90.0% of the 2760 variable and lead time combinations we evaluated. GraphCast also outperforms the most accurate previous ML-based weather forecasting model on 99.2% of the 252 targets it reported. GraphCast can generate a 10-day forecast (35 gigabytes of data) in under 60 seconds on Cloud TPU v4 hardware. Unlike traditional forecasting methods, ML-based forecasting scales well with data: by training on bigger, higher quality, and more recent data, the skill of the forecasts can improve. Together these results represent a key step forward in complementing and improving weather modeling with ML, open new opportunities for fast, accurate forecasting, and help realize the promise of ML-based simulation in the physical sciences.**

*Keywords: Weather forecasting, ECMWF, ERA5, HRES, learning simulation, graph neural networks*

## 1. Introduction

Every day, people factor in the upcoming weather when they plan what they do, from deciding which jacket to wear, to deciding whether to flee a hurricane. When these decisions involve anticipating the weather over the next ten days, people rely on "medium-range" weather forecasts, which are provided up to four times a day by weather bureaus, such as the European Centre for Medium-Range Weather Forecasts (ECMWF), the US's National Oceanic and Atmospheric Administration, and the UK Met Office. Here we show that weather forecasting based on machine learning (ML) can rival the approaches these bureaus have traditionally used.

Medium-range weather forecasts are generated by simulations run on large high-performance computing (HPC) clusters, and involve two main components. The first component is "data assimilation", which is the process of inferring and tracking the weather, based on recent and past observations from satellites, weather stations, ships, etc. The resulting output of data assimilation is an estimate of the most recent sequence of weather states, termed "analysis". The second is a forecast model, traditionally based on "numerical weather prediction" (NWP), which predicts the future temporal evolution of variables that represent the state of the weather. These two components are closely

*Corresponding author(s): remilam@deepmind.com, alvarosg@deepmind.com, matthjw@deepmind.com, peterbattaglia@deepmind.com*

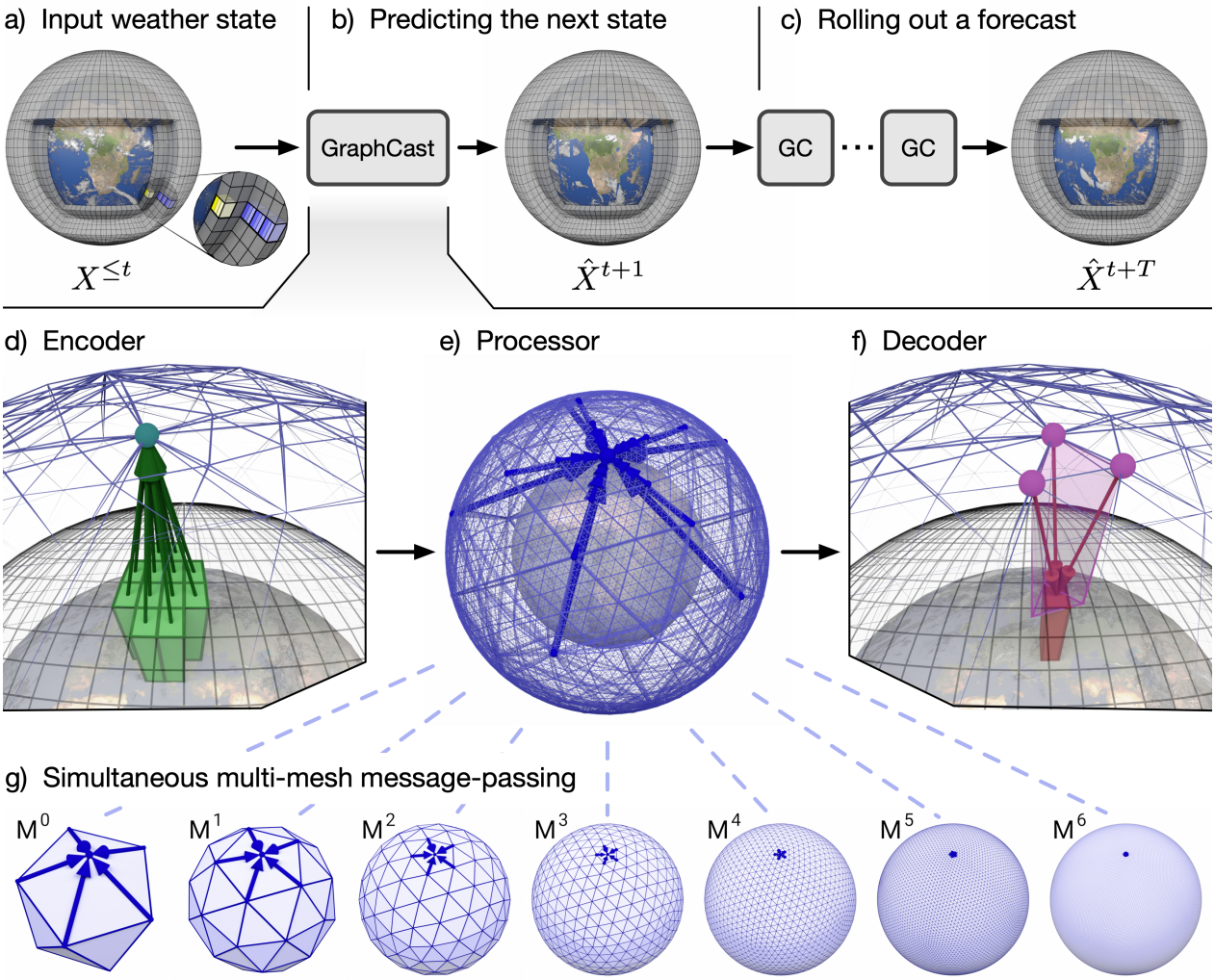


Figure 1 | **Model schematic.** (a) The input weather state(s) are defined on a high-resolution latitude-longitude-pressure-levels grid. In the closeup pop-out window, the yellow layers represent surface variables, and the blue layers represent atmospheric variables. Notice that the atmospheric variables' colors repeat several times, which signifies that they are represented at multiple pressure levels extending upwards. At each grid point, GraphCast (GC) models 5 surface variables, plus 6 atmospheric variables at each of 37 pressure levels, for 227 variables total. The 0.25° latitude-longitude grid has $1,038,240 = 721 \times 1440 = (180/0.25 + 1) \times (360/0.25)$ values, so in total the state representation contains $235,680,480$ values. (b) GraphCast predicts the next state of the weather as the latitude-longitude-pressure-levels grid. (c) A forecast is made by iteratively applying GraphCast to each previous predicted state, to produce a sequence of states which represent the weather as successive lead times. (d) The Encoder component of the GraphCast architecture maps local regions of the input (green boxes) into nodes of the multi-mesh graph representation (green, upward arrows which terminate in the green-blue node). (e) The Processor component updates each multi-mesh node using learned message-passing (heavy blue arrows that terminate at a node). (f) The Decoder component maps the processed multi-mesh features (purple nodes) back onto the grid representation (red, downward arrows which terminate at a red box). (g) The multi-mesh is a set of icosahedral meshes of increasing resolution, from the base mesh ($M^0$, 12 nodes) to the finest resolution ($M^6$, $40,962$ nodes), which has uniform resolution across the globe. Each node belongs to a particular mesh resolution, and is connected to all neighboring nodes at the same resolution, as well as higher resolutions. The learned message-passing over the different meshes' edges happens simultaneously, so that each node is updated by all of its incoming edges. (Earth texture credit [license: CC BY 4.0]: https://www.solarsystemscope.com/textures/.)

coupled: data assimilation uses an NWP model within its inference and tracking processes, previous forecasts are used to inform the assimilation estimates, and the most up-to-date analysis is used as input to the forecast model. This work focuses on improving the second component: forecast models.

The governing equations of Earth's weather have no known closed-form solution, thus NWP-based forecasting models approximate them numerically. NWP methods scale well with compute: accuracy generally increases with increased compute resources, because additional compute can be allocated to higher-resolution simulations, with more sophisticated parametrization schemes (Bauer et al., 2015; Benjamin et al., 2019). For decades, there has been tremendous investment in NWP systems, and some of the largest supercomputers in the world are used to make accurate forecasts of the weather (Met Office, 2021).

However, NWPs *do not* scale well with increasing amounts of available data. Today there are vast archives of weather and climatological observations, but traditionally there have been few direct means for these data to improve the quality of forecast models. The primary way that NWP methods are improved is through highly trained experts manually innovating better models, algorithms, and approximations, which is a time-consuming and costly process.

By contrast, ML methods generally *do* scale well with data. Across a wide range of scientific domains, ML-based methods can increase in accuracy as greater, higher-quality data becomes available, often at a much lower computational budget. In weather forecasting, ML systems are beginning to improve on NWP-based forecasting models, especially in regimes where traditional methods are relatively poor. Examples include sub-seasonal heat wave prediction (Lopez-Gomez et al., 2022) and precipitation nowcasting, both with and without NWP information (Espeholt et al., 2022; Ravuri et al., 2021; Shi et al., 2017; Sønderby et al., 2020).

In medium-range weather forecasting, however, ML-based methods have only recently begun to be competitive with traditional NWP. The most accurate medium-range operational forecasting system in the world, which is based on NWP, is ECMWF's Integrated Forecasting System (IFS), and is still regarded as superior to ML-based approaches. The IFS is comprised of two main components: HRES, a deterministic model that produces a single 10-day forecast, at 0.1° latitude-longitude resolution, several times per day; and ENS, an ensemble model that produces a set of 50 stochastically perturbed 15-day forecasts, at 0.2° resolution, several times per day. Among the most widely recognized and challenging goals in ML-based weather forecasting (Rasp and Thuerey, 2021; Weyn et al., 2019, 2020) is to outperform HRES and ENS, and in the past year there have been a number of exciting advances on this front. Keisler (2022) introduced a graph neural network (GNN)-based model, and reported results on 1° latitude-longitude resolution and 6-day forecasts, which approached the skill of HRES on several of variables and pressure levels, even improving over a coarsened resolution HRES on several variables. Kurth et al. (2022); Pathak et al. (2022) introduced FourCastNet(FCN), based on Fourier Neural Operators (Guibas et al., 2021; Li et al., 2020), and reported promising results on 0.25° latitude-longitude resolution and 7-day forecasts, the first to operate at that resolution globally. Bi et al. (2022) recently introduced Pangu-Weather, based on Vision Transformers (Dosovitskiy et al., 2020; Vaswani et al., 2017), and reported the most accurate ML-based weather forecasting on 0.25° latitude-longitude resolution and 7-day forecasts, outperforming HRES on many of the nine variables and pressure levels they presented. While these advances are encouraging, their results are generally reported on only a select handful of variables, without comprehensive comparison to operational forecasting systems, which begs the question: How skillful are ML-based weather models? The purpose of this work is to provide an answer to that question, and provide a powerful framework for competing with, and potentially rivaling operational systems.

We introduce GraphCast, which uses GNNs in an "encode-process-decode" (Battaglia et al., 2018) configuration, to autoregressively generate forecast trajectories. The encoder maps two consecutive

input frames of the latitude-longitude input grid, with hundreds of variables per grid point, into a multi-scale internal mesh representation. The processor performs many rounds of message-passing on this "multi-mesh", where the edges can span short or long ranges, and allow efficient communication without requiring an explicit hierarchy. The decoder maps the multi-mesh representation back to the latitude-longitude grid as a prediction for the next time step. GraphCast was trained on a corpus of 39 years of historical weather data, ECMWF's ERA5 reanalysis dataset (Hersbach et al., 2020), to make 10-day forecasts at 6-hour time steps, at 0.25° latitude-longitude resolution, for 5 surface variables and 6 atmospheric variables, each on 37 vertical pressure levels, that represent the state of the weather at a given location and time. The full state at a given time is represented as 235 million numbers, which is almost 900 megabytes at float32 precision, which posed unique challenges for implementing and training our model. Our GraphCast architecture is most related to Keisler (2022)'s approach, and follows a long tradition of GNN-based learned simulators (Battaglia et al., 2016; Cranmer et al., 2020; Fortunato et al., 2022; Kipf et al., 2018; Li et al., 2018, 2019; Mrowca et al., 2018; Pfaff et al., 2021; Rubanova et al., 2021; Sanchez-Gonzalez et al., 2018, 2019, 2020; Sun et al., 2019).

Comparing the forecast skill of different models is a challenging problem in its own right due to the many variables and time horizons these models predict. The way ECMWF improves its NWP-based HRES model is by assessing the skill of a new candidate version of HRES versus previous releases using the HRES Scorecard[1] (Haiden et al., 2018, 2021). The Scorecard compares two versions against one another on around 27 surface and atmospheric variables and levels, daily, for 10-day forecasts.

We adopt ECMWF's approach by introducing an analogous scorecard, for comparing HRES's skill with GraphCast's skill in a comprehensive way over many key variables. We evaluated our model's performance on all variables and levels in the Scorecard (except the ocean wave variables), as well as most of the WeatherBench (Rasp et al., 2020) variables and levels, giving 69 variables and levels in total. We used 10-day forecasts, at 6-hour time steps, on 0.25° latitude-longitude resolution.

Our primary findings are that across these 2760 combinations ($69 \times 40$) of evaluation variables and levels, and lead times, GraphCast has greater forecasting skill than HRES on 90.0% of them, for a held out test set from 2018. A key reason GraphCast can exceed HRES's performance is because it is trained directly from data, and thus, in principle, can capture weather phenomena, such as air masses, fronts, storms, etc., at scales which are not represented explicitly within NWP-based systems. We also find that GraphCast has greater forecasting skill than Pangu-Weather on 99.2% of the 252 targets that Bi et al. (2022) reported. Key advances of this work include:

- A novel multi-mesh GNN architecture for learned weather simulation.
- An autoregressive model that can be trained to generate forecasts on 0.25° latitude-longitude resolution and 37 levels of vertical resolution, for 40 or more steps.
- An evaluation protocol with comprehensive coverage of medium-range forecast variables.
- An ML-based forecasting model with greater skill than the best NWP-based deterministic model.
- The most accurate ML-based weather forecasting model.

[1] ECMWF Scorecards: https://sites.ecmwf.int/ifs/scorecards/. The current operational HRES (Cycle 47r3) is compared to the previous version, Cycle 47r2, here: https://sites.ecmwf.int/ifs/scorecards/scorecards-47r3HRES.html. The Scorecard uses both analysis and observation for verification, i.e., the "an" and "ob" sections, but here we focus on verifying via analysis, which is available from ECMWF.

## 2. ERA5 dataset

For training and evaluating GraphCast, we built our datasets from a subset of ECMWF's ERA5 (Hersbach et al., 2020)[2] reanalysis archive, for the years 1979-2018, at 6 hour time intervals (corresponding to 00z, 06z, 12z, 18z each day), at 0.25° horizontal latitude-longitude resolution, and 37 vertical atmospheric pressure levels[3]. "Reanalysis" means performing data assimilation on historical weather observations, to estimate the full state of the weather globally over time, and ERA5 is regarded as the most comprehensive and accurate reanalysis archive in the world.

Our model predicts a total of 227 target variables, which includes 5 surface variables, plus 6 atmospheric variables at each of 37 pressure levels (several other static and/or external variables were also provided as input context for our model—see Appendix Table A.1). These variables are uniquely identified by their short name (and the pressure level, for atmospheric variables). For example, the surface variable "2 metre temperature" is denoted `2T`; the atmospheric variable "Geopotential" at pressure level 500 hPa is denoted `Z500`. The modeled surface variables were: 2 metre temperature (`2T`), 10 metre u wind component (`10U`), 10 metre v wind component (`10V`), mean sea level pressure (`MSL`), and total precipitation (`TP`) (accumulated over the previous 6 hours). The atmospheric variables, which were represented at each of the 37 pressure levels, were: geopotential (`Z`), specific humidity (`Q`), temperature (`T`), u component of wind (`U`), v component of wind (`V`), and vertical velocity (`W`). The static/external variables include information such as the geometry of the grid/mesh, orography and radiation at the top of the atmosphere. See Appendix Table A.1 for more information about the variables, and Appendix A.1 and Appendix A.2.4 for full details.

As shown in Figure 1a, we represent the weather state at time index, $t$, as $X^t$. The grid encircling the Earth corresponds to the variables at every latitude, longitude, and pressure level. The surface and atmospheric variables are illustrated by the yellow and blue boxes in the magnified view, respectively. We refer to the subset of variables in $X^t$ that correspond to a particular grid point $i$ (1,038,240 in total) as $\mathbf{x}_i^t$, and to each variable $j$ of the 227 target variables as $x_{i,j}^t$.

## 3. GraphCast model

### 3.1. Generating a forecast

GraphCast takes as input two weather states, $(X^t, X^{t-1})$, which correspond to the current time, $t$, and the immediately preceding time, $t-1$, and predicts the weather state at the next time step (as depicted in Figure 1b),

$$\hat{X}^{t+1} = \text{GraphCast}(X^t, X^{t-1}) \quad . \tag{1}$$

To generate a $T$-step forecast, $\hat{X}^{t+1:t+T} = (\hat{X}^{t+1}, \ldots, \hat{X}^{t+T})$, GraphCast iteratively applies Equation (1) in an autoregressive fashion, feeding its own predictions back in as input to predict later steps (i.e., to predict step $t+2$, the input is $(\hat{X}^{t+1}, X^t)$; to predict step $t+3$, the input is $(\hat{X}^{t+2}, \hat{X}^{t+1})$). Figure 1b,c depicts this process, and see Appendix Equation A.5 for details.

[2] See ERA5 documentation: https://confluence.ecmwf.int/display/CKB/ERA5%3A+data+documentation.

[3] We follow common practice of using pressure as our vertical coordinate, instead of altitude. A "pressure level" is a field of altitudes with equal pressure. E.g., "pressure level 500 hPa" corresponds to the field of altitudes for which the pressure is 500 hPa. The relationship between pressure and altitude is determined by the geopotential variable.

### 3.2. Architecture

The core architecture of GraphCast uses GNNs in an "encode-process-decode" configuration (Battaglia et al., 2018), as depicted in Figure 1d,e,f. GNN-based learned simulators are very effective at learning complex physical dynamics of fluids and other materials (Pfaff et al., 2021; Sanchez-Gonzalez et al., 2020), as the structure of their representations and computations are analogous to learned finite element solvers (Alet et al., 2019). A key advantage of GNNs is that the input graph's structure determines what parts of the representation interact with one another via learned message-passing, allowing arbitrary patterns of spatial interactions over any range. By contrast, a convolutional neural network (CNN) is restricted to computing interactions within local patches (or, in the case of dilated convolution, over regularly strided longer ranges). And while Transformers (Vaswani et al., 2017) can also compute arbitrarily long-range computations, they do not scale well with very large inputs (e.g., the 1 million-plus grid points in GraphCast's global inputs) because of the quadratic memory complexity induced by computing all-to-all interactions. Contemporary extensions of Transformers often sparsify possible interactions to reduce the complexity, which in effect makes them analogous to GNNs (e.g., graph attention networks (Veličković et al., 2017)).

The way we capitalize on the GNN's ability to model arbitrary sparse interactions is by introducing GraphCast's internal multi-mesh representation, which has homogeneous spatial resolution over the globe, and allows long-range interactions within few message-passing steps. The multi-mesh is constructed by first dividing a regular icosahedron (12 nodes and 20 faces) iteratively 6 times to obtain a hierarchy of icosahedral meshes with a total of 40,962 nodes and 81,920 faces on the highest resolution (see Section 3). We leveraged the fact that the coarse-mesh nodes are subsets of the fine-mesh nodes, which allowed us to superimpose edges from all levels of the mesh hierarchy onto the finest-resolution mesh. This procedure yields a multi-scale set of meshes, with coarse edges bridging long distances at multiple scales, and fine edges capturing local interactions. Figure 1g shows each individual refined mesh, and Figure 1e shows the full multi-mesh.

GraphCast's encoder (Figure 1d) first maps the input data, from the original latitude-longitude grid, into learned features on the multi-mesh, using a GNN with directed edges from the grid points to the multi-mesh. The processor (Figure 1e) then uses a 16-layer deep GNN to perform learned message-passing on the multi-mesh, allowing efficient propagation of information across space due to the long-range edges. The decoder (Figure 1f) then maps the final multi-mesh representation back to the latitude-longitude grid using a GNN with directed edges, and combines this grid representation, $\hat{Y}^{t+k}$ with the input state, $\hat{X}^{t+k}$, to form the output prediction, $\hat{X}^{t+k+1} = \hat{X}^{t+k} + \hat{Y}^{t+k}$.

The encoder and decoder do not require the raw data to be arranged in a regular rectilinear grid, and can also be applied to arbitrary mesh-like state discretizations (Alet et al., 2019). The general architecture builds on various GNN-based learned simulators which have been successful in many complex fluid systems and other physical domains (Fortunato et al., 2022; Pfaff et al., 2021; Sanchez-Gonzalez et al., 2020). Keisler (2022) used similar approaches in weather forecasting, with promising results.

On a single Cloud TPU v4 device[4], GraphCast can generate a 0.25° resolution, 10-day forecast (at 6-hour steps) in under 60 seconds. For comparison, ECMWF's IFS system runs on a 11,664-core cluster, and generates a 0.1° resolution, 10-day forecast (released at 1-hour steps for the first 90 hours, 3-hour steps for hours 93-144, and 6-hour steps from 150-240 hours[5]) in about an hour (Rasp et al., 2020). For full details of GraphCast, see Appendix A.2.

[4]Cloud TPU v4 device specifications: https://cloud.google.com/tpu/docs/system-architecture-tpu-vm.
[5]See the HRES release details here: https://www.ecmwf.int/en/forecasts/datasets/set-i.

### 3.3. Training procedure

GraphCast was trained to minimize an objective function over 12-step forecasts (3 days) against ERA5 targets, using gradient descent. The objective function was,

$$\mathcal{L}_{\text{MSE}} = \underbrace{\frac{1}{|D_{\text{batch}}|}\sum_{d_0 \in D_{\text{batch}}}}_{\text{forecast date-time}} \underbrace{\frac{1}{T}\sum_{\tau \in 1:T_{\text{train}}}}_{\text{lead time}} \underbrace{\frac{1}{|G_{0.25^\circ}|}\sum_{i \in G_{0.25^\circ}}}_{\text{spatial location}} \underbrace{\sum_{j \in J}}_{\text{variable-level}} s_j w_j a_i \underbrace{(\hat{x}_{i,j}^{d_0+\tau} - x_{i,j}^{d_0+\tau})^2}_{\text{squared error}} \tag{2}$$

which averages the squared errors over forecast date-times, lead times, spatial locations, variables and levels, where

- $d_0 \in D_{\text{batch}}$ represent forecast initialization date-times in a batch of forecasts in the training set,
- $\tau \in 1 : T_{\text{train}}$ are the lead times that correspond to the $T_{\text{train}}$ autoregressive steps during training,
- $i \in G_{0.25^\circ}$ are the spatial latitude and longitude coordinates in the grid,
- $j \in J$ indexes the variable and level, e.g., $J = \{\text{z1000}, \text{z850}, \ldots, \text{2t}, \text{msl}\}$,
- $\hat{x}_{j,i}^{d_0+\tau}$ and $x_{j,i}^{d_0+\tau}$ are predicted and target values for some variable-level, location, and lead time,
- $s_j$ is the per-variable-level inverse variance of single-timestep differences,
- $w_j$ is the per-variable-level loss weight,
- $a_i$ is the normalized area of the latitude-longitude grid cell, which varies with latitude.

The quantities $s_j = \mathbb{V}_{i,t}\left[x_{i,j}^{t+1} - x_{i,j}^{t}\right]^{-1}$ are per-variable-level inverse variance estimates of the time differences. The $w_j$ are per-variable-level loss weights we specified in a simple way, to control how heavily different target variables are weighed during optimization. The $a_i$ weight depends on latitude, and weights the errors proportionally to the area of their corresponding grid cells. See the Appendix A.3 for full details of these symbols and indices.

We found that using an autoregressive, multi-step loss is effective at making the model minimize error accumulation over long forecasts, as described in Appendix A.3. The gradient of the objective with respect to the network weights is computed by backpropagating through the entire autoregressive prediction sequence in Appendix Equation A.5.

During model development and training, we only used ERA5 data from 1979-2017, while we reserved the 2018 ERA5 data for testing only. We maintained a strict protocol whereby data from 2018 onward was never observed by our research team or training procedures, until after our final GraphCast was frozen and the evaluation phase began (see Appendix A.1.3). As part of our preliminary experiments, we trained models up through 2015, validated them on 2016-2017 to select the best one, re-trained that model up through 2017, and tested on 2018. We also tested the model trained through 2015 on 2018 test data, and found its performance was not as good on 2018 data as the model re-trained on data through 2017. In other preliminary experiments, we found similar improvements on test performance for models trained up to an earlier test period (2015), versus those whose training data did not include years immediately preceding that test period. This may be due to the data being non-stationary across years (i.e., due to the ENSO cycle, climate change, or other multi-year weather patterns), or a result of having additional training data. In principle, GraphCast can be retrained (or fine-tuned) regularly, on the most recent weather data, to potentially reap these benefits.

Training GraphCast took about 3 weeks on 32 Cloud TPU v4 devices using batch parallelism. To reduce the memory footprint, we used sophisticated gradient-checkpoint strategies and low-precision numerics. To reduce training time, we applied a training schedule that ramped up the number of autoregressive steps, as detailed in Appendix A.3.

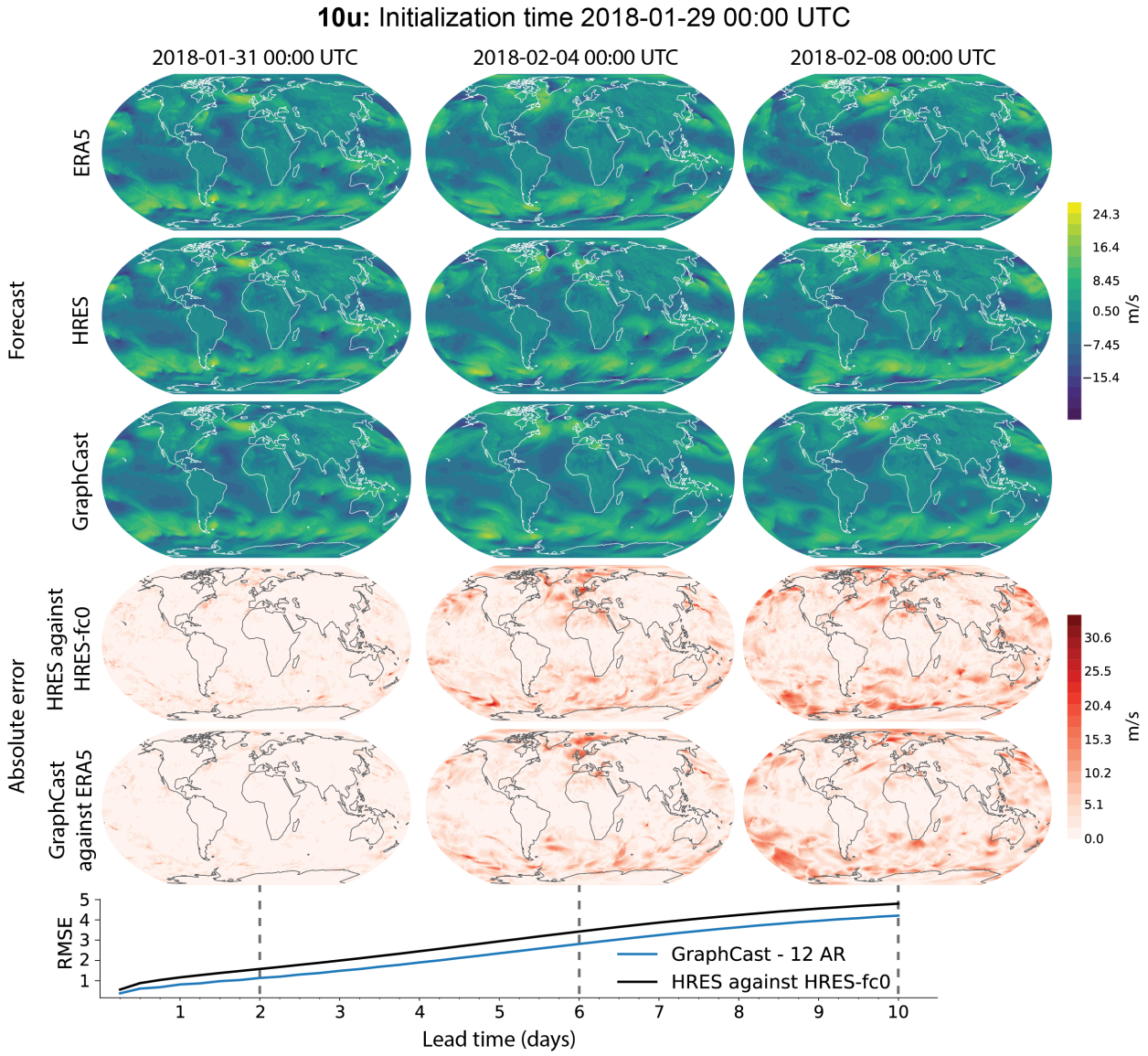


Figure 2 | **Forecast images and errors for 10u.** The plot shows the true and forecasted weather states for 10u. The columns are 3 lead times, 2, 6, and 10 days, with the dates indicated in the column headers. Row 1 shows ERA5, row 2 shows HRES, and row 3 shows GraphCast. Rows 4 and 5 are maps of the absolute value of the error between HRES and HRES-fc0, and GraphCast and ERA5, respectively. The line plot on the bottom shows the RMSE skill for HRES and GraphCast, as a function of lead time, for that specific forecast. The dashed lines show which lead times on the line plot correspond to the forecast images. The forecast initialization date-time (2018-01-20_00:00) was chosen by selecting the forecast which had median skill score for GraphCast versus HRES (normalized RMSE difference, averaged over lead times), across all forecasts in the 2018 test set, so that it is a representative example.

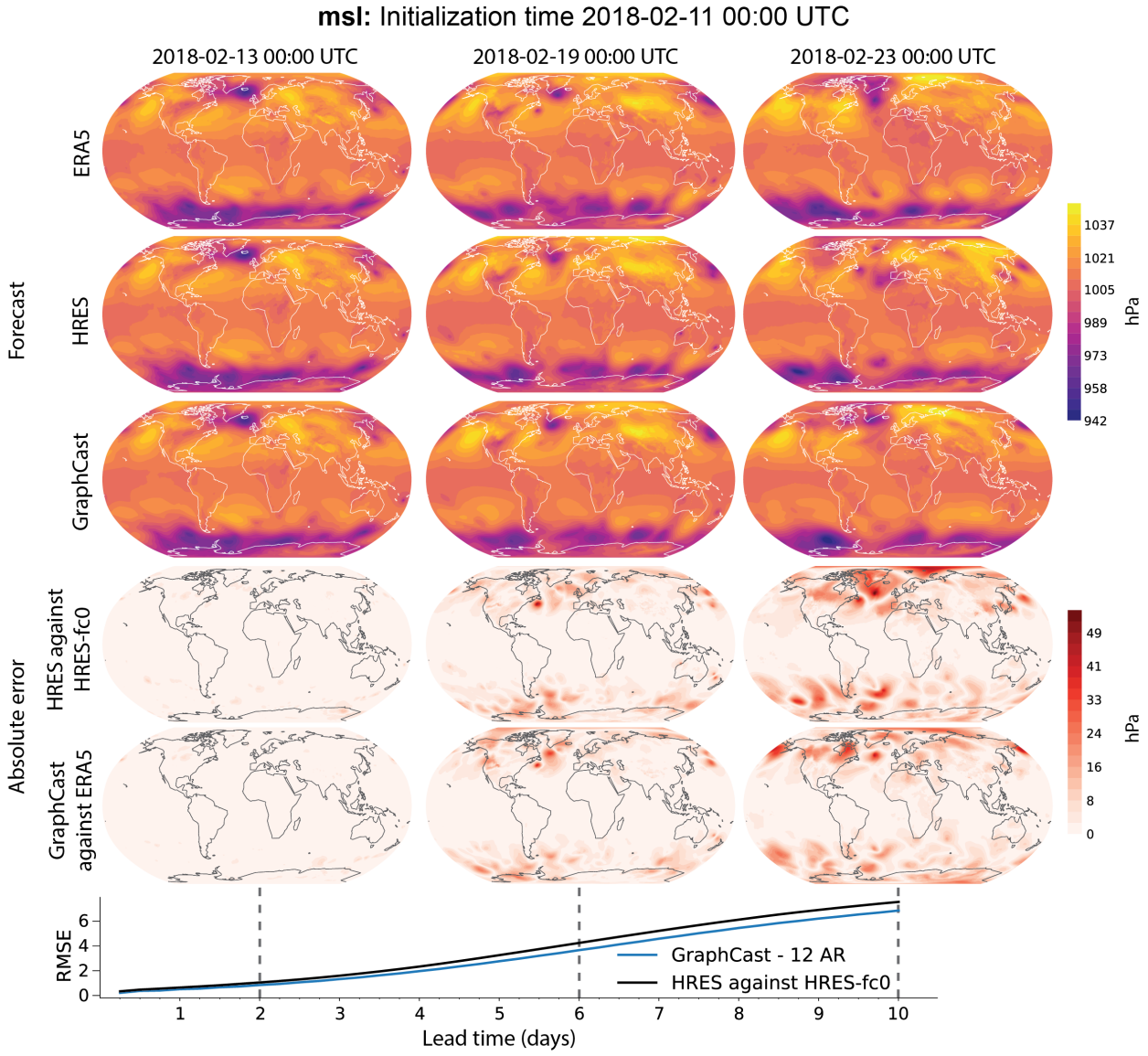


Figure 3 | **Forecast images and errors for MSL.** The plot shows the true and forecasted weather states for MSL, for forecast initialization date-time 2018-02-11_00:00. See the caption of Figure 2 for details.

## 4. Model evaluation

We quantify the skillfulness[6] of GraphCast, other ML models, and HRES using the root mean square error (RMSE) and the anomaly correlation coefficient (ACC), which are both computed against ground truth data. The RMSE, $\mathcal{L}^{j,\tau}_{\text{RMSE}}$, measures the magnitude of the differences between forecasts and ground truth (see Appendix Equation A.20). The ACC, $\mathcal{L}^{j,\tau}_{\text{ACC}}$, is defined in Appendix Equation A.21 and measures how well the model forecasts differences from climatology, i.e., the average weather for a location and date, correlate with the same quantity computed for the ground truth. For skill scores[6] we use the normalized RMSE difference between model $A$ and baseline $B$ as $(\text{RMSE}_A - \text{RMSE}_B)/\text{RMSE}_B$, and the normalized ACC difference as $(\text{ACC}_A - \text{ACC}_B)/(1 - \text{ACC}_B)$.

GraphCast was trained to predict ERA5 data and we therefore report errors that were computed against ERA5 as ground truth. The HRES model, however, uses HRES analysis as input. We therefore built a separate dataset, termed "HRES-fc0", that serves as ground truth for computing HRES errors (see Appendix A.1.2). This dataset comprises the initial time step of HRES forecasts. In other words, HRES errors were evaluated by comparing its forecasts to the HRES inputs for future forecasts. In the ECMWF Scorecard, the skill of HRES forecasts is also measured against HRES analysis. Our overall aim is to use a fair comparison between models trained on ERA5, versus HRES. See also Appendix A.5.

In the Results, we highlight 10 "headline" variables chosen from the ECMWF Scorecard:

- `z500`, `z850`: Geopotential, at 500 and 850 hPa, respectively.
- `t500`, `t850`: Temperature, at 500 and 850 hPa, respectively.
- `q700`: Specific humidity, at 700 hPa.
- `u500`, `v850`: `u` and `v` components of wind at 500 and 850 hPa, respectively. The error in the wind components is always highly correlated (e.g., average Pearson's $r = 0.99981$ across levels for `u` and `v`), so we only show one at each level.
- `2t`: 2 metre temperature (surface temperature).
- `10u`: 10 metre vector wind (surface wind). Again, we exclude `10v` because it is highly correlated with `10u` ($r = 0.99995$).
- `msl`: Mean sea level pressure.

These variables were selected because they comprise all the variables the Scorecard shows for the comparison-with-analysis section, which GraphCast models. We chose those pressure levels which were at or closer to the surface than the 500 hPa level, for each variable.

In our Results below, we also present a scorecard with GraphCast's performance versus HRES for all 69 variable-level combinations in our evaluation set. The pressure levels we evaluated were the 13 used in WeatherBench (Rasp et al., 2020): 50, 100, 150, 200, 250, 300, 400, 500, 600, 700, 850, 925, 1000 hPa. The variables we evaluated were those listed above, except vertical velocity (`w`) and total precipitation (`tp`), which are diagnostic variables in ERA5 and HRES (i.e., they are calculated from other variables, rather than modeled explicitly). This leaves 69 total evaluation variables (4 surface variables, plus 5 atmospheric variables at 13 pressure levels), at 40 lead times, for a total of 2760.

Figure 2 and Figure 3 show sequences of states from ERA5, an HRES forecast, and a GraphCast forecast (top three rows). The fourth row shows the absolute value of the difference between HRES's forecasted state and the HRES-fc0 ground truth. The fifth row shows the absolute value of the difference between GraphCast's forecasted state and the ERA5 ground truth. The line plot shows the RMSE between the forecasts and their respective ground truths, with dashed lines indicating the lead times to which the forecast images correspond. The figure demonstrates the quality of GraphCast's forecasts, and provides intuition for how forecasts are used to compute skill metrics.

[6] For definitions of "skill" and "skill score": https://en.wikipedia.org/wiki/Forecast_skill.

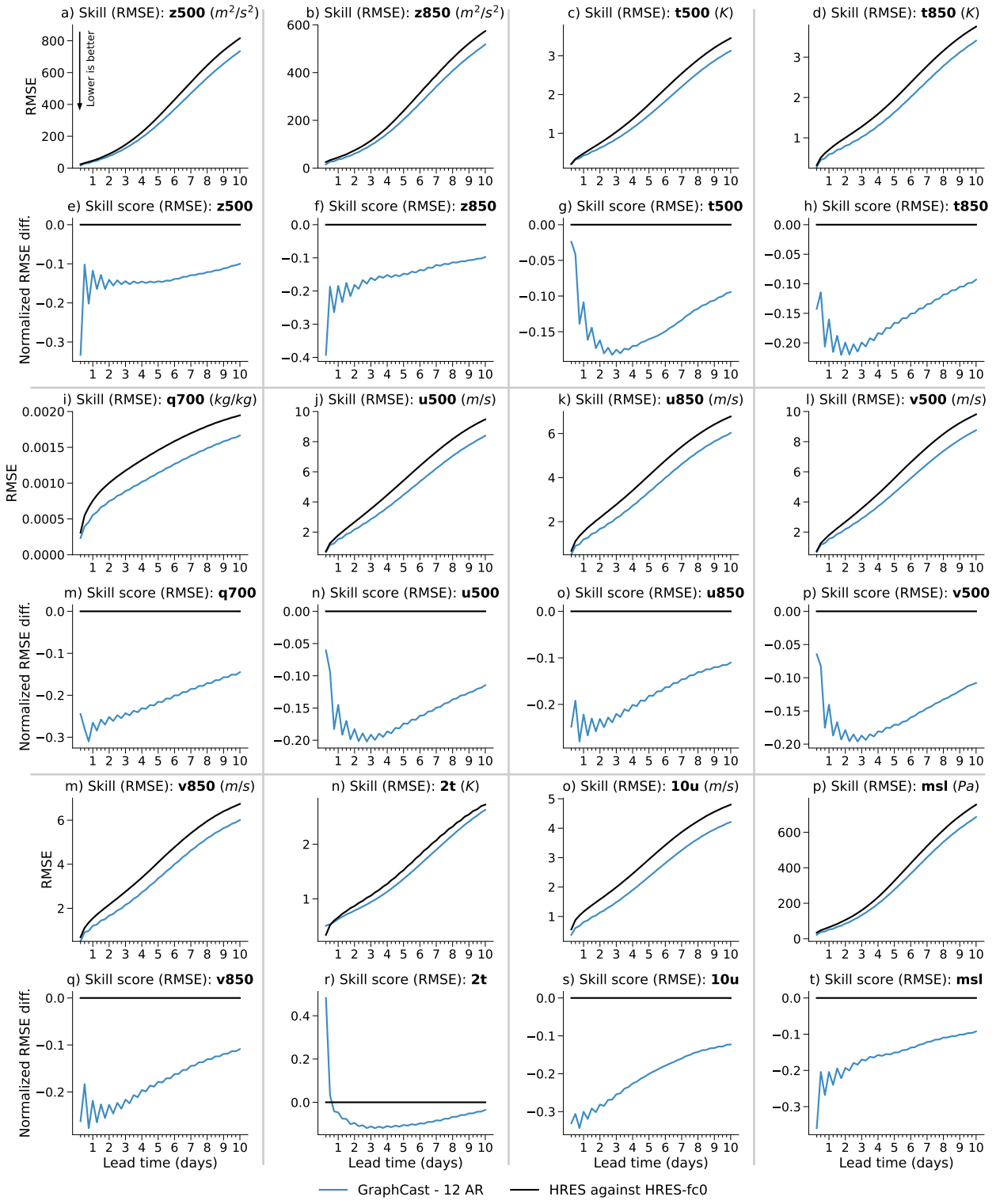


Figure 4 | **GraphCast's RMSE skill versus HRES in 2018** *(lower is better)*. Rows 1 and 3 show absolute RMSE for GraphCast (blue lines) and HRES (black lines); rows 2 and 4 show normalized RMSE differences between GraphCast's RMSE and HRES's. GraphCast's RMSE is computed against ERA5; HRES's RMSE is computed against HRES-fc0 (see Section 4). Each subplot represents a single variable (and pressure level), as indicated in the subplot titles. The x-axis represents lead time, at 6-hour steps over 10 days. The y-axis represents (absolute or normalized) RMSE. We did not include 10v because it was highly correlated with 10u. Note, there are small oscillations visible in the skill of GraphCast, as a function of lead time, which we speculate may result from reporting metrics only for 00z and 12z initialization times (to be comparable to HRES), and small differences in region-specific forecast skill that varies over the diurnal cycle (see Appendix Figures A.4 to A.5).

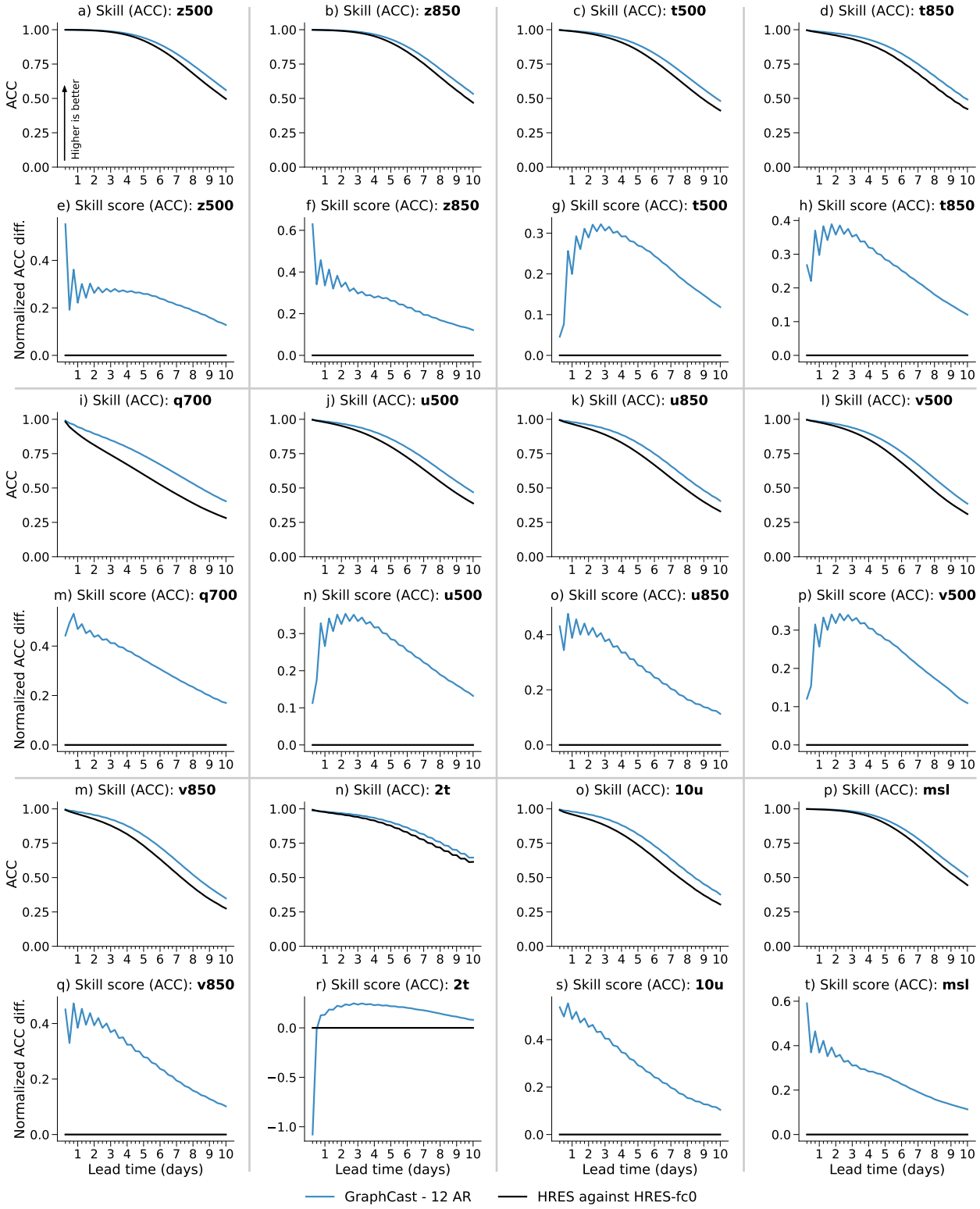


Figure 5 | **GraphCast's ACC skill versus HRES in 2018** *(higher is better)*. This plots skill and skill scores for ACC and normalized ACC differences, respectively. See the caption of Figure 4 for all other details.

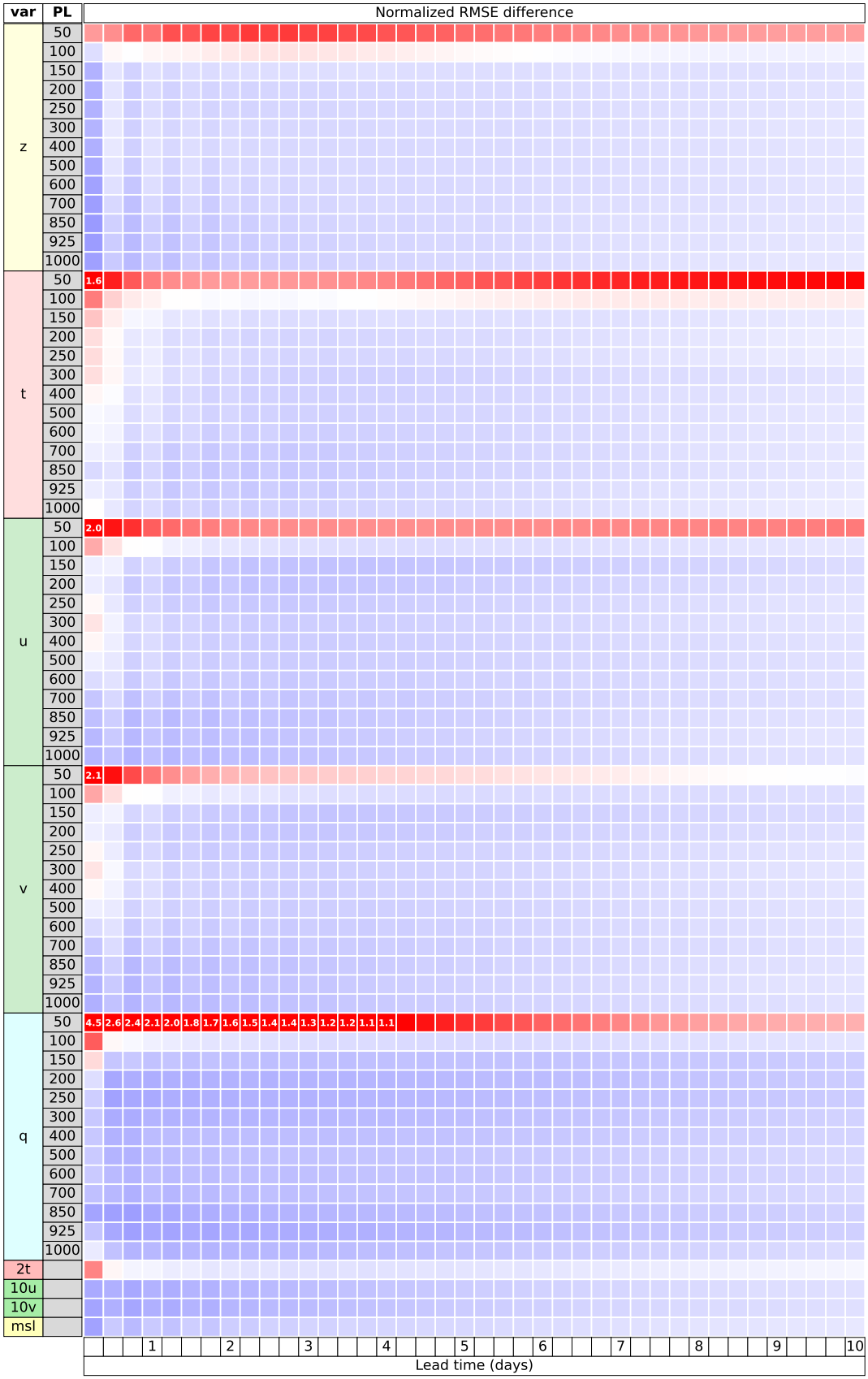


Figure 6 | **Scorecard of GraphCast's skill score versus HRES in 2018.** Skill score was measured by normalized RMSE differences, where colors are proportional to the value: blue is negative (better), red is positive (worse). Each cell represents one of the 2760 evaluation variable-level-lead-time combinations. Each row is a single variable ("var") at a single pressure level ("PL"). Each column is a single lead time ("Forecast days"), at 6 hour steps, over 10 days. Normalized RMSE difference has a lower bound of -1, but no upper bound, so we clipped (red) colors above 1, and overlaid the numeric value on those cells which were clipped.

# 5. Results

## 5.1. Skill of GraphCast versus HRES

Our results show GraphCast comprehensively outperforms HRES's weather forecasting skill across 10-day forecasts, at 0.25° horizontal resolution.

Figure 4 shows how GraphCast (blue lines) clearly outperforms HRES (black lines) for our 10 headline surface and atmospheric variables (see Section 3 for evaluation protocol), chosen from the ECMWF Scorecard for pressure levels closest to the surface. Each subplot corresponds to a variable (and pressure level, for atmospheric variables) and skill (y-axis) is plotted at 6-hour steps over 10 day horizons (x-axis). Rows 1 and 3 show absolute RMSE, and rows 2 and 4 show the corresponding normalized differences in RMSE. From the initial 6-hour step, through all 40 steps over 10 days, GraphCast almost always has higher skill, and the normalized differences are often 30% lower error than HRES at early lead times, while typically plateauing to around 10 – 15% after 10 days. We found similar results when evaluating ACC, as shown in Figure 5. We also performed a regional analysis, which indicates that these results are consistent across the entire globe (see Appendix Figure A.3, Appendix Figure A.4, and Appendix Figure A.5).

Figure 6 summarizes the normalized differences (i.e., Figure 4's rows 2 and 4) for all variables and pressure levels, across the 10 day forecasts, in a format analogous to the ECMWF Scorecard. Each row represents a single variable (left row label) at a single level (right row label), where each colored square's color is proportional to the normalized difference (scaled between -1 and 1), where blue indicates GraphCast had lower error than HRES (negative normalized difference), with -1 (solid blue) meaning zero error, and red indicates it had higher error (positive normalized differences), with 1 (solid red) meaning twice the error. GraphCast outperformed HRES on 90.0% of the 2760 variables, levels, and lead times in our evaluation set (4 surface variables, plus 5 atmospheric variables × 13 levels, over 10 days with 4 steps per day). We note that HRES tended to have superior performance than GraphCast on the upper atmospheric levels, especially pressure level 50 hPa, which is not surprising because the total training loss weight applied to pressure levels at or below 50 hPa was only 0.66% of the total loss weight across all variables and levels (see Section 3 for details). When excluding the 50 hPa level, the percentage of the 2240 targets on which GraphCast outperforms HRES is 96.6%; when excluding levels 50 and 100 hPa, the percentage of the 1720 targets is 99.2%.

| | Skill score (normalized RMSE difference) | |
|---|---|---|
| Lead time | Mean | Median |
| 1-day | -0.092 | -0.15 |
| 3-day | -0.13 | -0.19 |
| 5-day | -0.12 | -0.17 |

Table 1 | **Summary of skill score between GraphCast and HRES, for key lead times.** The means and medians (columns) are computed across the normalized RMSE differences for our 69 evaluation variable-level combinations, for lead times of 1, 3, and 5 days (the rows).

Together these results show that GraphCast has substantially greater skill than HRES across the variables, levels, and lead times we tested. Table 1 shows the mean and median RMSE skill scores between GraphCast and HRES, averaged over all 69 variables and levels, for the key lead times of 1, 3, and 5 days. For context, the improvement in skill GraphCast provides over HRES appears to be considerably greater than typical improvements between successive cycles of HRES releases, and roughly corresponds to the improvements in skill over several years.

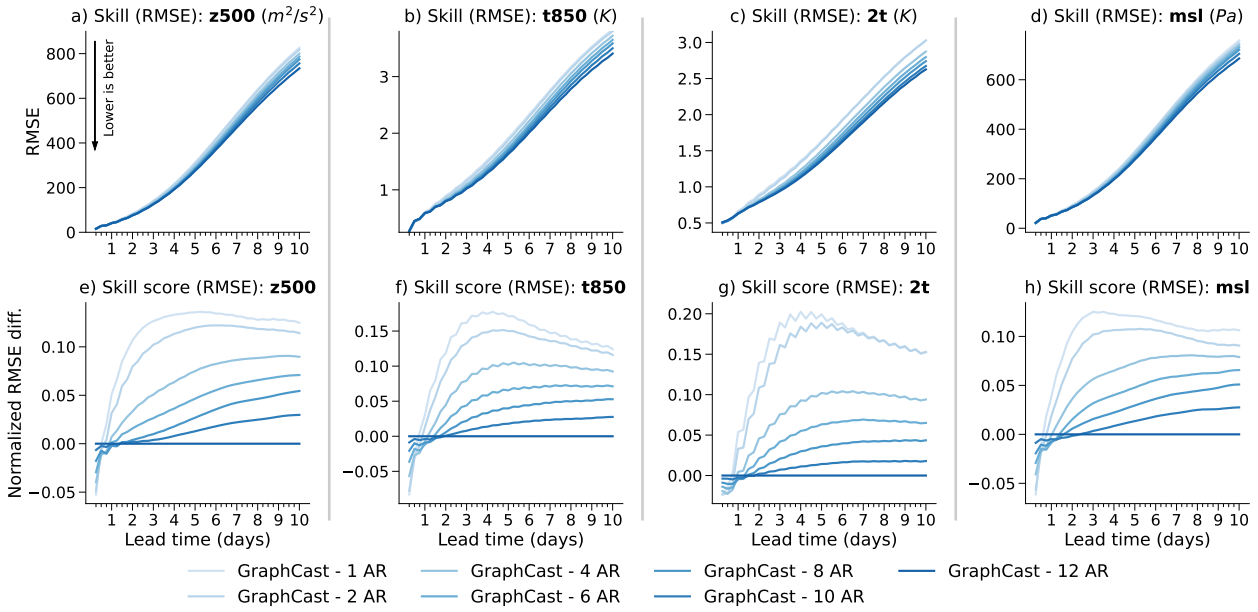


Figure 7 | **Effects of autoregressive training.** Each line in the plots represents GraphCast, fine-tuned with different numbers of autoregressive steps, where increasing numbers of steps are represented with darker shades of blue. The top row shows absolute RMSE for GraphCast. The bottom row shows normalized RMSE differences, with respect to our full 12 autoregressive-step GraphCast. Each column shows a different variable, as indicated in the subplot titles. The x-axis represents lead time, at 6-hour steps over 10 days. The y-axis represents (absolute or normalized) RMSE.

### 5.2. How autoregressive training affects forecast skill

Figure 7 shows how the forecast performance varies with the number of autoregressive (AR) steps used to train our model. When trained with fewer autoregressive steps, the model performs better at short lead times, and worse over longer lead times. As the number of autoregressive steps is increased, the performance becomes worse at short lead times, but better at longer ones[7]. These results suggest potential for combining multiple models with varying numbers of AR steps, e.g., for short, medium and long lead times, to capitalize on their respective advantages across the entire forecast period.

### 5.3. Performance of GraphCast compared to top ML forecasting models

Of the three most recent ML-based forecasting models in the past year, Keisler (2022)'s model, FCN (Kurth et al., 2022; Pathak et al., 2022), and Pangu-Weather (Bi et al., 2022), the most recent, Pangu-Weather reports results that are categorically stronger than the former two. Thus we focus our evaluation of GraphCast on Pangu-Weather, as it represents the state-of-the-art of ML-based weather forecasting. Pangu-Weather is based on a Vision Transformers (Dosovitskiy et al., 2020; Vaswani et al., 2017), which has similar patterns of computations to GNNs (e.g., Pangu-Weather's masking of non-local interactions among embedded patches means it can also be viewed as a Graph Attention Network (Veličković et al., 2017)).

[7]The way we obtained each of these models was using a curriculum where the 1 AR-step model was fine-tuned for 1000 gradient updates each, on increasing numbers of AR steps, from 2-12 (see Appendix A.3 and Appendix Figure A.2 for details). Each model shown in Figure 7 was from the end of its respective number of AR-step training. This means the higher AR-step models had slightly more training than the others, though we do not believe that each had generally converged, so training the lower AR-step models longer likely would not have made much difference.

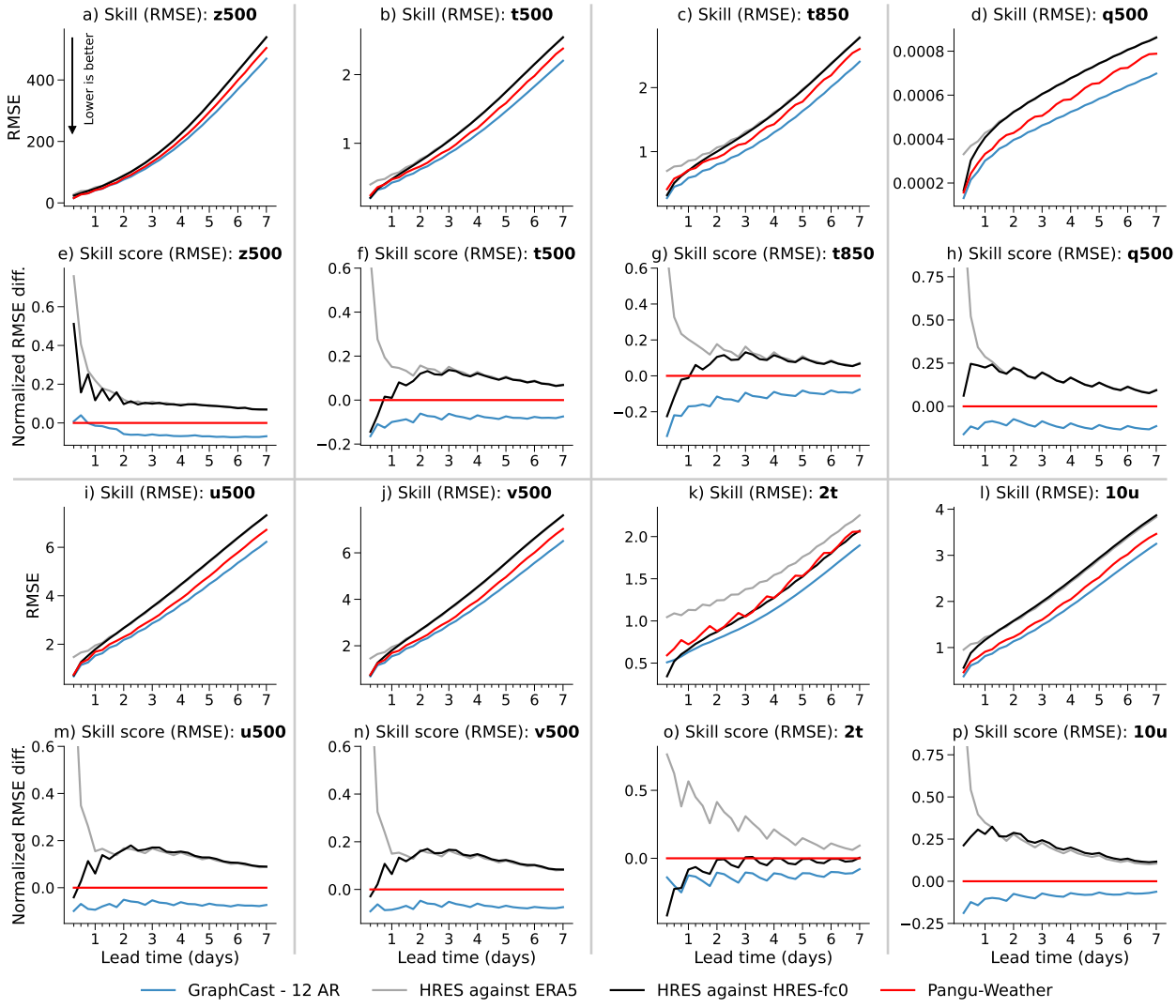


Figure 8 | **Comparison between GraphCast and Pangu-Weather, on RMSE skill.** Rows 1 and 3 show absolute RMSE for GraphCast (blue lines), Pangu-Weather (Bi et al., 2022) (red lines), HRES evaluated against HRES-fc0 (black lines), and HRES evaluated against ERA5; rows 2 and 4 show normalized RMSE differences between the models with respect to Pangu-Weather. Each subplot represents a single variable (and pressure level, for atmospheric variables), as indicated in the subplot titles. The x-axis represents lead time, at 6-hour steps over 10 days. The y-axis represents (absolute or normalized) RMSE. The variables and levels were chosen to be those reported by Bi et al. (2022). The reason HRES evaluated against ERA5 is shown is because that is what Bi et al. (2022) use as their HRES baseline, while our analyses use HRES against HRES-fc0. We did not include 10v, because 10u is already present, and the two are highly correlated.

Bi et al. (2022) report Pangu-Weather's 7-day forecast accuracy (RMSE and ACC) on: z500, t500, t850, q500, u500, v500, 2t, 10u, 10v, and msl. They did not provide the raw values, so we used a software tool to extract the values from their paper's plots. As shown in Figure 8, GraphCast (blue lines) outperforms Pangu-Weather (red lines) on 99.2% of targets reported by Bi et al. (2022). The only 2 (of the 252 total) metrics on which Pangu-Weather outperformed GraphCast was z500, at lead times 6 and 12 hours, where GraphCast had 1.7% higher average RMSE (Figure 8a,e).

Our evaluation approach for comparison with HRES differs from that used in Bi et al. (2022). As described in Section 4, we measured HRES's skill by using HRES-fc0 as ground truth. In Bi et al. (2022), HRES's skill is computed by using ERA5 as ground truth. As detailed in Appendix A.5, this HRES-against-ERA5 approach can lead to systematically worse skill estimates. As shown in Figure 8, we computed HRES's skill both ways: against HRES-fc0 (black lines), and against ERA5 (grey lines). HRES-against-ERA5 is clearly worse at early lead times, and in some cases, especially for temperature, HRES-against-HRES-fc0 has better skill than Pangu-Weather.

## 6. Discussion

We show that our GraphCast model outperforms the most accurate deterministic operational system, ECMWF's HRES, on 10-day forecasts, at 6-hour steps, and 0.25° latitude-longitude resolution. We evaluated GraphCast's skill on a comprehensive set of 2760 variable, pressure level, and lead time combinations, and our results showed our model had lower RMSE than HRES on 90.0% of the metrics. When we excluded the upper atmosphere fields from 100 hPa and above, GraphCast outperformed HRES on 99.2% of the 1760 targets. GraphCast also outperformed the best previous ML baseline, Pangu-Weather (Bi et al., 2022), on 99.2% of the 252 metrics which Bi et al. (2022) reported. A key innovation of GraphCast was its novel "multi-mesh" representation, which allows it to capture much longer-range spatial interactions than in traditional NWP methods, and thus support much coarser native timesteps. This is part of why GraphCast can generate an accurate 10-day weather forecast, at 6 hour steps, in under 60 seconds on a single Cloud TPU v4 device.

One important caveat to our work is that we focused on deterministic forecasts, and compared GraphCast's skill only to HRES. While HRES is ECMWF's highest fidelity single forecast, the other pillar of IFS, the ensemble forecasting system, ENS, is of comparable importance, especially for forecast horizons in the 5-15 day range. Because weather dynamics are highly nonlinear, and the analysis provided as input to weather models has inherent uncertainty, as a forecast's lead time increases, it becomes increasingly difficult to make accurate point-wise predictions of weather trajectories, and thus modeling the uncertainty becomes increasingly important. Over longer time horizons, we notice GraphCast's forecasts become somewhat more blurry than HRES's. This is to be expected, because it was trained to optimize a weighted mean squared error. Thus, the way it expresses uncertainty over longer lead times is by producing a forecast closer to the mean. This contrasts with traditional deterministic NWPs, which will make high-resolution, but sometimes incorrect, predictions. However these predictions may be useful for some purposes, such as predicting temperature extremes, or the chance of storms. It also contrasts with ensemble forecast models, which produce multiple forecasts from a sample of initial conditions which approximate the uncertainty in the true initial conditions. Statistical measures of the ensemble forecasts are used to quantify uncertainty. Ensemble forecasting offers key advantages, such as estimating the distribution more explicitly, however it comes at the cost of requiring many expensive forecasts to be generated. Building models that model uncertainty, and can be comprehensively evaluated against ensemble systems, is a crucial next step.

Another caveat is that we focused on 0.25° latitude-longitude resolution, while HRES operates on 0.1°. This choice was based on the fact that ERA5 is available only at 0.25°, and because there are

substantial engineering challenges in working with large 0.25° state representations. Nevertheless, there is no principled or technical reason why our approach will not scale to higher resolution, given sufficient data and engineering advances.

We also invite the meteorological community to join us in defining new, broader skill metrics that go beyond the ECMWF Scorecard (which was designed to evaluate IFS Cycles against one another). For example, storm tracking, preparing for extreme weather, etc. are important applications of weather forecasts, which would benefit from concrete metrics and benchmarks on which ML-based methods can be trained and evaluated.

Our contributions represent a significant step forward for ML-based weather modeling, and can, in principle, apply to a much wider range of environmental and other geo-spatiotemporal forecasting problems. Key application areas include modeling other weather variables, seasonal and climate forecasting, wildfires, deforestation, animal and human activity, etc. However, our approach should not be regarded as a replacement for traditional weather forecasting methods, which have been developed for decades, rigorously tested in many real-world contexts, and provide probabilistic predictions (Palmer, 2018). Rather, our work should be interpreted as clear evidence that ML-based simulation is able to scale effectively to the challenges of real-world forecasting problems, and has potential to complement and transform the current best methods. More broadly, by training on complex, real-world data, and outperforming traditional numerical approaches, this work helps realize the promise of ML-based simulation in the physical sciences.

**Acknowledgements** In alphabetical order, we thank Kelsey Allen, Charles Blundell, Matt Botvinick, Zied Ben Bouallegue, Michael Brenner, Rob Carver, Matt Chantry, Marc Deisenroth, Peter Deuben, Marta Garnelo, Ryan Keisler, Dmitrii Kochkov, Chris Mattern, Piotr Mirowski, Peter Norgaard, Ilan Price, Chongli Qin, Stephan Rasp, Yulia Rubanova, Kunal Shah, and Daniel Worrall for advice and feedback on our work. We also thank ECMWF for providing invaluable datasets to the research community.

## References

F. Alet, A. K. Jeewajee, M. B. Villalonga, A. Rodriguez, T. Lozano-Perez, and L. Kaelbling. Graph element networks: adaptive, structured computation and memory. In *International Conference on Machine Learning*, pages 212–222. PMLR, 2019.

K. R. Allen, Y. Rubanova, T. Lopez-Guevara, W. Whitney, A. Sanchez-Gonzalez, P. Battaglia, and T. Pfaff. Learning rigid dynamics with face interaction graph networks. *arXiv preprint arXiv:2212.03574*, 2022.

J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv*, 2016. doi: 10.48550/ARXIV.1607.06450.

I. Babuschkin, K. Baumli, A. Bell, S. Bhupatiraju, J. Bruce, P. Buchlovsky, D. Budden, T. Cai, A. Clark, I. Danihelka, C. Fantacci, J. Godwin, C. Jones, R. Hemsley, T. Hennigan, M. Hessel, S. Hou, S. Kapturowski, T. Keck, I. Kemaev, M. King, M. Kunesch, L. Martens, H. Merzic, V. Mikulik, T. Norman, J. Quan, G. Papamakarios, R. Ring, F. Ruiz, A. Sanchez, R. Schneider, E. Sezener, S. Spencer, S. Srinivasan, L. Wang, W. Stokowiec, and F. Viola. The DeepMind JAX Ecosystem, 2020. URL http://github.com/deepmind.

P. Battaglia, R. Pascanu, M. Lai, D. Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. *Advances in neural information processing systems*, 29, 2016.

P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

P. Bauer, A. Thorpe, and G. Brunet. The quiet revolution of numerical weather prediction. *Nature*, 525, 2015. doi: https://doi.org/10.1038/nature14956.

S. G. Benjamin, J. M. Brown, G. Brunet, P. Lynch, K. Saito, and T. W. Schlatter. 100 years of progress in forecasting and nwp applications. *Meteorological Monographs*, 59:13–1, 2019.

K. Bi, L. Xie, H. Zhang, X. Chen, X. Gu, and Q. Tian. Pangu-weather: A 3d high-resolution model for fast and accurate global weather forecast. *arXiv preprint arXiv:2211.02556*, 2022.

J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

T. Chen, B. Xu, C. Zhang, and C. Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.

M. Cranmer, S. Greydanus, S. Hoyer, P. Battaglia, D. Spergel, and S. Ho. Lagrangian neural networks. *arXiv preprint arXiv:2003.04630*, 2020.

A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

L. Espeholt, S. Agrawal, C. Sønderby, M. Kumar, J. Heek, C. Bromberg, C. Gazen, R. Carver, M. Andrychowicz, J. Hickey, et al. Deep learning for twelve hour precipitation forecasts. *Nature communications*, 13(1):1–10, 2022.

M. Fortunato, T. Pfaff, P. Wirnsberger, A. Pritzel, and P. Battaglia. Multiscale meshgraphnets. *arXiv preprint arXiv:2210.00612*, 2022.

J. Godwin*, T. Keck*, P. Battaglia, V. Bapst, T. Kipf, Y. Li, K. Stachenfeld, P. Veličković, and A. Sanchez-Gonzalez. Jraph: A library for graph neural networks in jax., 2020. URL http://github.com/deepmind/jraph.

J. Guibas, M. Mardani, Z. Li, A. Tao, A. Anandkumar, and B. Catanzaro. Adaptive fourier neural operators: Efficient token mixers for transformers. *arXiv preprint arXiv:2111.13587*, 2021.

T. Haiden, M. Janousek, J. Bidlot, R. Buizza, L. Ferranti, F. Prates, and F. Vitart. *Evaluation of ECMWF forecasts, including the 2018 upgrade*. European Centre for Medium Range Weather Forecasts Reading, UK, 2018.

T. Haiden, M. Janousek, F. Vitart, Z. Ben-Bouallegue, L. Ferranti, and F. Prates. Evaluation of ECMWF forecasts, including the 2021 upgrade. *ECMWF Technical Memoranda*, September 2021.

T. Hennigan, T. Cai, T. Norman, and I. Babuschkin. Haiku: Sonnet for JAX, 2020. URL http://github.com/deepmind/dm-haiku.

H. Hersbach, B. Bell, P. Berrisford, S. Hirahara, A. Horányi, J. Muñoz-Sabater, J. Nicolas, C. Peubey, R. Radu, D. Schepers, et al. The era5 global reanalysis. *Quarterly Journal of the Royal Meteorological Society*, 146(730):1999–2049, 2020.

S. Hoyer and J. Hamman. xarray: N-D labeled arrays and datasets in Python. *Journal of Open Research Software*, 5(1), 2017. doi: 10.5334/jors.148. URL https://doi.org/10.5334/jors.148.

R. Keisler. Forecasting global weather with graph neural networks. *arXiv preprint arXiv:2202.07575*, 2022.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

T. Kipf, E. Fetaya, K.-C. Wang, M. Welling, and R. Zemel. Neural relational inference for interacting systems. In *International Conference on Machine Learning*, pages 2688–2697. PMLR, 2018.

T. Kurth, S. Subramanian, P. Harrington, J. Pathak, M. Mardani, D. Hall, A. Miele, K. Kashinath, and A. Anandkumar. Fourcastnet: Accelerating global high-resolution weather forecasting using adaptive fourier neural operators. *arXiv preprint arXiv:2208.05419*, 2022.

Y. Li, J. Wu, R. Tedrake, J. B. Tenenbaum, and A. Torralba. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. *arXiv preprint arXiv:1810.01566*, 2018.

Y. Li, J. Wu, J.-Y. Zhu, J. B. Tenenbaum, A. Torralba, and R. Tedrake. Propagation networks for model-based control under partial observation. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 1205–1211. IEEE, 2019.

Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.

I. Lopez-Gomez, A. McGovern, S. Agrawal, and J. Hickey. Global extreme heat forecasting using neural weather models. *arXiv preprint arXiv:2205.10972*, 2022.

I. Loshchilov and F. Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Met Office, 2021. URL https://www.metoffice.gov.uk/about-us/what/technology/supercomputer.

D. Mrowca, C. Zhuang, E. Wang, N. Haber, L. F. Fei-Fei, J. Tenenbaum, and D. L. Yamins. Flexible neural representation for physics prediction. *Advances in neural information processing systems*, 31, 2018.

T. Palmer. The ECMWF ensemble prediction system: Looking back (more than) 25 years and projecting forward 25 years. *Quarterly Journal of the Royal Meteorological Society*, 145(S1):12–24, Sept. 2018. doi: 10.1002/qj.3383.

J. Pathak, S. Subramanian, P. Harrington, S. Raja, A. Chattopadhyay, M. Mardani, T. Kurth, D. Hall, Z. Li, K. Azizzadenesheli, et al. Fourcastnet: A global data-driven high-resolution weather model using adaptive fourier neural operators. *arXiv preprint arXiv:2202.11214*, 2022.

T. Pfaff, M. Fortunato, A. Sanchez-Gonzalez, and P. Battaglia. Learning mesh-based simulation with graph networks. In *International Conference on Learning Representations*, 2021.

P. Ramachandran, B. Zoph, and Q. V. Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

S. Rasp and N. Thuerey. Data-driven medium-range weather prediction with a resnet pretrained on climate simulations: A new model for weatherbench. *Journal of Advances in Modeling Earth Systems*, 13(2):e2020MS002405, 2021.

S. Rasp, P. D. Dueben, S. Scher, J. A. Weyn, S. Mouatadid, and N. Thuerey. Weatherbench: a benchmark data set for data-driven weather forecasting. *Journal of Advances in Modeling Earth Systems*, 12 (11):e2020MS002203, 2020.

S. Ravuri, K. Lenc, M. Willson, D. Kangin, R. Lam, P. Mirowski, M. Fitzsimons, M. Athanassiadou, S. Kashem, S. Madge, et al. Skilful precipitation nowcasting using deep generative models of radar. *Nature*, 597(7878):672–677, 2021.

Y. Rubanova, A. Sanchez-Gonzalez, T. Pfaff, and P. Battaglia. Constraint-based graph network simulator. *arXiv preprint arXiv:2112.09161*, 2021.

A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell, and P. Battaglia. Graph networks as learnable physics engines for inference and control. In *International Conference on Machine Learning*, pages 4470–4479. PMLR, 2018.

A. Sanchez-Gonzalez, V. Bapst, K. Cranmer, and P. Battaglia. Hamiltonian graph networks with ode integrators. *arXiv preprint arXiv:1909.12790*, 2019.

A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, and P. Battaglia. Learning to simulate complex physics with graph networks. In *International Conference on Machine Learning*, pages 8459–8468. PMLR, 2020.

X. Shi, Z. Gao, L. Lausen, H. Wang, D.-Y. Yeung, W.-k. Wong, and W.-c. Woo. Deep learning for precipitation nowcasting: A benchmark and a new model. *Advances in neural information processing systems*, 30, 2017.

C. K. Sønderby, L. Espeholt, J. Heek, M. Dehghani, A. Oliver, T. Salimans, S. Agrawal, J. Hickey, and N. Kalchbrenner. Metnet: A neural weather model for precipitation forecasting. *arXiv preprint arXiv:2003.12140*, 2020.

C. Sun, P. Karlsson, J. Wu, J. B. Tenenbaum, and K. Murphy. Stochastic prediction of multi-agent interactions from partial observations. *arXiv preprint arXiv:1902.09641*, 2019.

R. Swinbank, M. Kyouda, P. Buchanan, L. Froude, T. M. Hamill, T. D. Hewson, J. H. Keller, M. Matsueda, J. Methven, F. Pappenberger, et al. The tigge project and its achievements. *Bulletin of the American Meteorological Society*, 97(1):49–67, 2016.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

J. A. Weyn, D. R. Durran, and R. Caruana. Can machines learn to predict weather? using deep learning to predict gridded 500-hpa geopotential height from historical weather data. *Journal of Advances in Modeling Earth Systems*, 11(8):2680–2693, 2019.

J. A. Weyn, D. R. Durran, and R. Caruana. Improving data-driven global weather prediction using deep convolutional neural networks on a cubed sphere. *Journal of Advances in Modeling Earth Systems*, 12(9):e2020MS002109, 2020.

# Appendix

## A.1. ECMWF data and forecasts

Here we summarize ECMWF's ERA5 data that we used to train and evaluate our learned model (Appendix A.1.1), as well as the data we used to evaluate ECMWF's HRES forecasts, which is our main NWP baseline (Appendix A.1.2). We constructed multiple internal datasets that all comprise subsets of the original data provided by ECMWF. Whenever appropriate, we distinguish between ECMWF's datasets, which we refer to as ECMWF's "archive", versus the datasets we have built from these archives, which we refer to as "datasets".

| Type | Variable name | Short name | ECMWF Parameter ID | Role (accumulation period, if applicable) |
|---|---|---|---|---|
| Atmospheric | Geopotential | z | 129 | Input/Predicted |
| Atmospheric | Specific humidity | q | 133 | Input/Predicted |
| Atmospheric | Temperature | t | 130 | Input/Predicted |
| Atmospheric | U component of wind | u | 131 | Input/Predicted |
| Atmospheric | V component of wind | v | 132 | Input/Predicted |
| Atmospheric | Vertical velocity | w | 135 | Input/Predicted |
| Single | 2 metre temperature | 2t | 167 | Input/Predicted |
| Single | 10 metre u wind component | 10u | 165 | Input/Predicted |
| Single | 10 metre v wind component | 10v | 166 | Input/Predicted |
| Single | Mean sea level pressure | msl | 151 | Input/Predicted |
| Single | Total precipitation | tp | 28 | Input/Predicted (6h) |
| Single | TOA incident solar radiation | tisr | 212 | Input (1h) |
| Static | Geopotential at surface | z | 129 | Input |
| Static | Land-sea mask | lsm | 172 | Input |
| Static | Latitude | n/a | n/a | Input |
| Static | Longitude | n/a | n/a | Input |
| Clock | Local time of day | n/a | n/a | Input |
| Clock | Elapsed year progress | n/a | n/a | Input |

Table A.1 | **ECMWF variables used in our datasets.** The "Type" column indicates whether the variable represents a *static* property, a time-varying *single*-level property (e.g., surface variables are included), or a time-varying *atmospheric* property. The "Variable name" and "Short name" columns are ECMWF's labels. The "ECMWF Parameter ID" column is a ECMWF's numeric label, and can be used to construct the URL for ECMWF's description of the variable, by appending it as suffix to the following prefix, replacing "ID" with the numeric code: `https://apps.ecmwf.int/codes/grib/param-db/?id=ID`. The "Role" column indicates whether the variable is something our model takes as input and predicts, or only uses as input context (the double horizontal line separates predicted from input-only variables, to make the partitioning more visible.

### A.1.1. ERA5

For training and evaluating our ML models, we built a dataset from ECMWF's ERA5 archive (Hersbach et al., 2020)[8], which is a large corpus of data that represents the global weather from 1959 to the

[8]See ERA5 documentation: `https://confluence.ecmwf.int/display/CKB/ERA5%3A+data+documentation`.

present, at 0.25° latitude/longitude resolution, and 1 hour increments, for hundreds of static, surface, and atmospheric variables. The ERA5 archive is based on *reanalysis*, which uses ECMWF's NWP model that was operational for most of 2016[9] to assimilate 12 hour windows of observations, from 21z-09z and 09z-21z, as well as previous forecasts, into a dense representation of the weather's state, for each historical date and time.

Our ERA5 dataset contains a subset of available variables in ECMWF's ERA5 archive (Appendix Table A.1), on 37 pressure levels:

$$1, 2, 3, 5, 7, 10, 20, 30, 50, 70, 100, 125, 150, 175, 200, 225, 250, 300, 350, 400, 450,$$
$$500, 550, 600, 650, 700, 750, 775, 800, 825, 850, 875, 900, 925, 950, 975, 1000$$

The range of years included was 1979-2018, and were downsampled to 6 hour time intervals (corresponding to 00z, 06z, 12z and 18z each day). The downsampling is performed simply by subsampling, except for the total precipitation, which is accumulated for the 6 hours leading up to the corresponding downsampled time.

### A.1.2. HRES

Evaluating the HRES model baseline requires two separate sets of data, namely the forecast data and the ground truth data[10], which are summarized in the subsequent sub-sections. The HRES versions which were operational during our test year of 2018 were Cycle 43r3 and Cycle 45r1[9].

**HRES operational forecasts** HRES is the most accurate deterministic NWP-based weather model in the world, so to evaluate the HRES baseline, we built a dataset of HRES's archived historical forecasts. HRES is regularly updated by ECMWF, so these forecasts represent the latest HRES model at the time the forecasts were made. The forecasts were downloaded at 0.1°, then spatially downsampled to 0.25° (to match ERA5's resolution) using ECMWF's Metview library, with default `regrid` parameters. Each forecast consists of a 10-day prediction with 6h timesteps, initialized from HRES analysis at 00z or 12z each day[11].

**HRES-fc0 analysis** For evaluating the skill of the HRES operational forecasts, we constructed a ground truth dataset, "HRES-fc0", based on ECMWF's HRES operational forecast archive. This dataset comprises the initial time step of each HRES forecast, at initialization times 00z, 06z, 12z, and 18z. The HRES-fc0 analysis data is similar to the ERA5 data, but it is assimilated using the latest ECMWF NWP model at the forecast time, and assimilates observations from ±3 hours around the corresponding date and time, along with previous forecasts.

[9] HRES Cycle 41r2 was operational from `2016-03-08` to `2016-11-21`. HRES Cycle 43r3 was operational from `2017-07-11` to `2018-06-04`. HRES Cycle 45r1 was operational from `2018-06-05` to `2019-06-10`. See `https://www.ecmwf.int/en/forecasts/documentation-and-support/changes-ecmwf-model` for the full cycle release schedule.

[10] Note, ECMWF also provides an archive of "HRES Analysis" data, which is not to be confused with our HRES-fc0 dataset. Our understanding is the HRES Analysis dataset includes atmospheric and land surface analyses, but is not the input which is provided to the HRES forecasts, therefore we do not use it because it would disadvantage HRES at short lead times.

[11] Our HRES dataset was missing several surface variables (`10U`, `10V`, `MSL`, `TP`) on 11 dates and times:

1. `2018-10-22_12:00:00`
2. `2018-10-28_12:00:00`
3. `2018-11-06_12:00:00`
4. `2018-11-09_12:00:00`
5. `2018-11-12_12:00:00`
6. `2018-11-18_12:00:00`
7. `2018-11-21_12:00:00`
8. `2018-11-23_12:00:00`
9. `2018-11-24_12:00:00`
10. `2018-11-30_12:00:00`
11. `2018-12-04_12:00:00`

because the download requests failed with the message, `"Requested data is on one or more damaged tape"`. For evaluation of those variables we excluded these forecasts and ground truth data from our analyses.

### A.1.3. Training, validation, and test splits

We separated the ERA5 data used to develop the model, i.e., to train and validate it iteratively, from the data used for evaluating test performance. This means that neither the researchers, nor the model training software, ever had access to data from the "test set" until we had developed and frozen our final model solely based on the "development set". We then evaluated the frozen model on the test set to assess its performance, and compare it against baselines. We chose to split the data "causally" by which we mean that the test set only contained dates later than those in the training and validation sets. Our test set includes all data from 2018; our "development set" contains data from 2017 and earlier. This prevented our model development and training protocols from being able to exploit any information from the test set (either by directly optimizing on the training set, or by somehow fitting to the validation set)[12]. A further benefit of this protocol, as compared to other choices in the literature, e.g., Bi et al. (2022); Keisler (2022), is that it reflects actual deployment conditions, in which the forecast cannot depend on information from the future.

Within our development set, we further split the data into a "training set" comprising the years 1979-2015[13], and a "validation set" that includes 2016-2017. We used the training set as training data for our models and the validation set for hyperparameter optimization and model selection, i.e., to decide on the best-performing model. We then froze that model, including the architecture and all the training choices, and re-trained the architecture from scratch on the full development set, combining training and validation data (1979-2017), to leverage all the available development data. That final, trained model marks the transition from the development phase to the evaluation phase.

## A.2. Model details

### A.2.1. Time notation

Forecasting involves a number of different time symbols, e.g., to denote the initial forecast time, validity time, forecast horizon, etc. We therefore start by introducing the relevant terms and notation for clarity and simplicity. We refer to a particular point in time as "date-time", indicated by calendar date and UTC time. For example, `2018-06-21_18:00:00` means June 21, 2018, at 18:00 UTC. For shorthand, we also sometimes use the Zulu convention, i.e., 00z, 06z, 12z, 18z mean `00:00`, `06:00`, `12:00`, `18:00` UTC, respectively. We further define the following symbols:

- $t$: Forecast time step index, which indexes the number of steps since the forecast was initialized.
- $T$: Forecast horizon, which represents the total number of steps in a forecast.
- $d$: Validity time, which indicates the date-time of a particular weather state.
- $d_0$: Forecast initialization time, indicating the validity time of a forecast's initial inputs.
- $\Delta d$: Forecast step duration, indicating how much time elapses during one forecast step.
- $\tau$: Forecast lead time, which represents the elapsed time in the forecast (i.e., $\tau = t\Delta d$).

### A.2.2. General forecasting problem statement

Let $Z^d$ denote the true state of the global weather at time $d$. The time evolution of the true weather can be represented by an underlying discrete-time dynamics function, $\Phi$, which generates the state at the next time step ($\Delta d$ in the future) based on the current one, i.e., $Z^{d+\Delta d} = \Phi(Z^d)$. We then obtain

[12] Of course, we experienced the weather in 2018, but we assume this had negligible effect on the choices we made to develop the model.

[13] In preliminary work, we also explored earlier data from 1959-1978, but found it had little impact on training performance, so in the final phases of our work we excluded 1959-1978 for simplicity.

a trajectory of $T$ future weather states by applying $\Phi$ autoregressively $T$ times,

$$Z^{d+\Delta d:d+T\Delta d} = (\underbrace{\Phi(Z^d), \Phi(Z^{d+\Delta d}), \ldots, \Phi(Z^{d+(T-1)\Delta d})}_{1 \ldots T \text{ autoregressive iterations}}) \quad . \tag{A.1}$$

Our goal is to find an accurate and efficient model, $\phi$, of the true dynamics function, $\Phi$, that can efficiently forecast the state of the weather over some forecast horizon, $T\Delta d$. We assume that we cannot observe $Z^d$ directly, but instead only have some partial observation $X^d$, which is an incomplete representation of the state information required to predict the weather perfectly. Because $X^d$ is only an approximation of the instantaneous state $Z^d$, we also provide $\phi$ with one or more past states, $X^{d-\Delta d}, X^{d-2\Delta d}, \ldots$, in addition to $X^d$. The model can then, in principle, leverage this additional context information to approximate $Z^d$ more accurately. Thus $\phi$ predicts a future weather state as,

$$\hat{X}^{d+\Delta d} = \phi(X^d, X^{d-\Delta d}, \ldots) \quad . \tag{A.2}$$

Analogous to Appendix Equation A.1, the prediction $\hat{X}^{d+\Delta d}$ can be fed back into $\phi$ to autoregressively produce a full forecast,

$$\hat{X}^{d+\Delta d:d+T\Delta d} = (\underbrace{\phi(X^d, X^{d-\Delta d}, \ldots), \phi(\hat{X}^{d+\Delta d}, X^d, \ldots), \ldots, \phi(\hat{X}^{d+(T-1)\Delta d}, \hat{X}^{d+(T-2)\Delta d}, \ldots)}_{1 \ldots T \text{ autoregressive iterations}}) \quad . \tag{A.3}$$

We assess the forecast quality, or skill, of $\phi$ by quantifying how well the predicted trajectory, $\hat{X}^{d+\Delta d:d+T\Delta d}$, matches the ground-truth trajectory, $X^{d+\Delta d:d+T\Delta d}$. However, it is important to highlight again that $X^{d+\Delta d:d+T\Delta d}$ only comprises our observations of $Z^{d+\Delta d:d+T\Delta d}$, which itself is unobserved. We then measure the consistency between forecasts and ground truth with an objective function,

$$\mathcal{L}\left(\hat{X}^{d+\Delta d:d+T\Delta d}, X^{d+\Delta d:d+T\Delta d}\right) \quad ,$$

which is described explicitly in section Appendix A.4.

### A.2.3. Modeling ECMWF weather data

For training and evaluating models, we treat our ERA5 dataset as the ground truth representation of the surface and atmospheric weather state. As described in Appendix A.1.2, we used the HRES-fc0 dataset as ground truth for evaluating the skill of HRES.

In our work, the temporal resolution of data and forecasts was always $\Delta d = 6$ hours with a maximum forecast horizon of 10 days, corresponding to a total of $T = 40$ steps. Because $\Delta d$ is a constant throughout this paper, we can simplify the notation using $(X^t, X^{t+1}, \ldots, X^{t+T})$ instead of $(X^d, X^{d+\Delta d}, \ldots, X^{d+T\Delta d})$, to index time with an integer instead of a specific date-time.

An ERA5 weather state $X^t$ comprises all variables in Appendix Table A.1[14], at a 0.25° horizontal latitude-longitude resolution with a total of $721 \times 1440 = 1,038,240$ grid points and 37 vertical pressure levels. The set of (horizontal) grid points is given by

$$G_{0.25^\circ} = \{-90.0, -89.75, \ldots, 90.0\} \times \{-179.75, -179.5, \ldots, 180.0\} \quad , \tag{A.4}$$

[14]Note, only the "predicted" variables are output by our model, because the "input"-only variables are forcings that are known apriori, and simply appended to the state on each time-step. We ignore them in the description for simplicity, so in total there are 5 surface variables and 6 atmospheric variables.

and atmospheric variables are defined at all pressure levels. We refer to the subset of variables in $X^t$ that correspond to a particular grid point $i$ (1,038,240 in total) as $\mathbf{x}_i^t$, and to each variable $j$ of the 227 target variables as $x_{i,j}^t$. The full state representation $X^t$ therefore contains a total of $721 \times 1440 \times (5 + 6 \times 37) = 235,680,480$ values[15].

While a regular latitude-longitude grid is a straightforward way to discretize the sphere, it induces a non-uniform distribution of grid points. This is likely sub-optimal for learning useful representations and also inefficient from a computational perspective, because the unnecessarily high resolution at the poles requires a lot of processing. It is therefore desirable to discretize the sphere more uniformly, e.g., by using a suitable mesh instead of a latitude-longitude grid. As detailed below, a central component of our model is an internal representation that uses a multi-scale icosahedral mesh, which we refer to as "multi-mesh". We construct this mesh by refining a base icosahedral mesh consisting of 12 nodes iteratively six times, which yields a total of 40,962 nodes. A key feature of the multi-mesh is that we preserve the edges at all refinement levels (see Figure 1g).

### A.2.4. GraphCast

Our GraphCast model is defined as a one-step learned simulator that takes the role of $\phi$ in Appendix Equation A.2 and predicts the next step based on two consecutive input states, as defined in Equation (1) in the main text,

$$\hat{X}^{t+1} = \text{GraphCast}(X^t, X^{t-1}) \quad .$$

As in Appendix Equation A.3, we can apply GraphCast iteratively to produce a forecast

$$\hat{X}^{t+1:t+T} = \underbrace{(\text{GraphCast}(X^t, X^{t-1}), \text{GraphCast}(\hat{X}^{t+1}, X^t), \ldots, \text{GraphCast}(\hat{X}^{t+T-1}, \hat{X}^{t+T-2}))}_{1 \ldots T \text{ autoregressive iterations}} \tag{A.5}$$

of arbitrary length, $T$.

GraphCast is implemented using GNNs in an "encode-process-decode" configuration (Battaglia et al., 2018; Pfaff et al., 2021; Sanchez-Gonzalez et al., 2020), where the encoder maps (surface and atmospheric) features on the input latitude-longitude grid to a multi-mesh, the processor performs many rounds of message-passing on the multi-mesh, and the decoder maps the multi-mesh features back to the output latitude-longitude grid (see Figure 1).

The model operates on a graph $\mathcal{G}(\mathcal{V}^{\text{G}}, \mathcal{V}^{\text{M}}, \mathcal{E}^{\text{M}}, \mathcal{E}^{\text{G2M}}, \mathcal{E}^{\text{M2G}})$ which is defined in detail in the subsequent paragraphs.

**Grid nodes** $\mathcal{V}^{\text{G}}$ represents the set containing each of the grid nodes $v_i^{\text{G}}$. Each grid node represents a vertical slice of the atmosphere at a given latitude-longitude point, $i$. The features associated with each grid node $v_i^{\text{G}}$ are $\mathbf{v}_i^{\text{G,features}} = [\mathbf{x}_i^{t-1}, \mathbf{x}_i^t, \mathbf{f}_i^{t-1}, \mathbf{f}_i^t, \mathbf{f}_i^{t+1}, \mathbf{c}_i]$, where $\mathbf{x}_i^t$ is the time-dependent weather state $X^t$ corresponding to grid node $v_i^{\text{G}}$ and includes all the predicted data variables for all 37 atmospheric levels as well as surface variables. The forcing terms $\mathbf{f}^t$ consist of time-dependent features that can be computed analytically, and do not require to be predicted by GraphCast. They include the total incident solar radiation at the top of the atmosphere, accumulated over 1 hour, the sine and cosine of the local time of day (normalized to [0, 1)), and the sine and cosine of the of year progress (normalized to [0, 1)). The constants $\mathbf{c}_i$ are static features: the binary land-sea mask, the geopotential at the surface, the cosine of the latitude, and the sine and cosine of the longitude. At 0.25° resolution, there is a total of $721 \times 1440 = 1,038,240$ grid nodes, each with (5 *surface variables* + 6 *atmospheric variables* × 37 *levels*) × 2 *steps* + 5 *forcings* × 3 *steps* + 5 *constant* = 474 input features.

[15] At the poles, the 1440 longitude points are equal, so the actual number of distinct grid points is slightly smaller.

**Mesh nodes** $\mathcal{V}^{\mathrm{M}}$ represents the set containing each of the mesh nodes $v_i^{\mathrm{M}}$. Mesh nodes are placed uniformly around the globe in a R-refined icosahedral mesh $M^R$. $M^0$ corresponds to a unit-radius icosahedron (12 nodes and 20 triangular faces) with faces parallel to the poles (see Figure 1g). The mesh is iteratively refined $M^r \rightarrow M^{r+1}$ by splitting each triangular face into 4 smaller equilateral faces, resulting in an extra node in the middle of each edge, and re-projecting the new nodes back onto the unit sphere.[16] Features $\mathbf{v}_i^{\mathrm{M,features}}$ associated with each mesh node $v_i^{\mathrm{M}}$ include the cosine of the latitude, and the sine and cosine of the longitude. GraphCast works with a mesh that has been refined $R = 6$ times, $M^6$, resulting in 40,962 mesh nodes (see Supplementary Appendix Table A.2), each with 3 input features.

| Refinement | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| Num Nodes | 12 | 42 | 162 | 642 | 2,562 | 10,242 | 40,962 |
| Num Faces | 20 | 80 | 320 | 1,280 | 5,120 | 20,480 | 81,920 |
| Num Edges | 60 | 240 | 960 | 3,840 | 15,360 | 61,440 | 245,760 |
| Num Multilevel Edges | 60 | 300 | 1,260 | 5,100 | 2,0460 | 81,900 | 327,660 |

Table A.2 | **Multi-mesh statistics.** Statistics of the multilevel refined icosahedral mesh as function of the refinement level $R$. Edges are considered to be bi-directional and therefore we count each edge in the mesh twice (once for each direction).

**Mesh edges** $\mathcal{E}^{\mathrm{M}}$ are bidirectional edges added between mesh nodes that are connected in the mesh. Crucially, mesh edges are added to $\mathcal{E}^{\mathrm{M}}$ for all levels of refinement, i.e., for the finest mesh, $M^6$, as well as for $M^5$, $M^4$, $M^3$, $M^2$, $M^1$ and $M^0$. This is straightforward because of how the refinement process works: the nodes of $M^{r-1}$ are always a subset of the nodes in $M^r$. Therefore, nodes introduced at lower refinement levels serve as hubs for longer range communication, independent of the maximum level of refinement. The resulting graph that contains the joint set of edges from all of the levels of refinement is what we refer to as the "multi-mesh". See Figure 1e,g for a depiction of all individual meshes in the refinement hierarchy, as well as the full multi-mesh.

For each edge $e^{\mathrm{M}}_{v_{\mathrm{s}}^{\mathrm{M}} \rightarrow v_{\mathrm{r}}^{\mathrm{M}}}$ connecting a sender mesh node $v_{\mathrm{s}}^{\mathrm{M}}$ to a receiver mesh node $v_{\mathrm{r}}^{\mathrm{M}}$, we build edge features $\mathbf{e}^{\mathrm{M,features}}_{v_{\mathrm{s}}^{\mathrm{M}} \rightarrow v_{\mathrm{r}}^{\mathrm{M}}}$ using the position on the unit sphere of the mesh nodes. This includes the length of the edge, and the vector difference between the 3d positions of the sender node and the receiver node computed in a local coordinate system of the receiver. The local coordinate system of the receiver is computed by applying a rotation that changes the azimuthal angle until that receiver node lies at longitude 0, followed by a rotation that changes the polar angle until the receiver also lies at latitude 0. This results in a total of 327,660 mesh edges (See Appendix Table A.2), each with 4 input features.

**Grid2Mesh edges** $\mathcal{E}^{\mathrm{G2M}}$ are unidirectional edges that connect sender grid nodes to receiver mesh nodes. An edge $e^{\mathrm{G2M}}_{v_{\mathrm{s}}^{\mathrm{G}} \rightarrow v_{\mathrm{r}}^{\mathrm{M}}}$ is added if the distance between the mesh node and the grid node is smaller or equal than 0.6 times[17] the length of the edges in mesh $M^6$ (see Figure 1) which ensures every grid node is connected to at least one mesh node. Features $\mathbf{e}^{\mathrm{G2M,features}}_{v_{\mathrm{s}}^{\mathrm{G}} \rightarrow v_{\mathrm{r}}^{\mathrm{M}}}$ are built the same way as those for the mesh edges. This results on a total of 1,618,746 Grid2Mesh edges, each with 4 input features.

[16] Note this split and re-project mechanism leads to a small variance in triangle edge lengths across the mesh.

[17] Technically it is 0.6 times the "longest" edge in $M^6$, since there is some variance in the length of the edges caused by the split-and-reproject mechanism.

**Mesh2Grid edges** $\mathcal{E}^{\text{M2G}}$ are unidirectional edges that connect sender mesh nodes to receiver grid nodes. For each grid point, we find the triangular face in the mesh $M^6$ that contains it and add three Mesh2Grid edges of the form $e^{\text{M2G}}_{v_s^{\text{M}} \to v_r^{\text{G}}}$, to connect the grid node to the three mesh nodes adjacent to that face (see Figure 1). Features $\mathbf{e}^{\text{M2G,features}}_{v_s^{\text{M}} \to v_r^{\text{G}}}$ are built on the same way as those for the mesh edges. This results on a total of 3,114,720 Mesh2Grid edges (3 mesh nodes connected to each of the $721 \times 1440$ latitude-longitude grid points), each with four input features.

### A.2.5. Encoder

The purpose of the encoder is to prepare data into latent representations for the processor, which will run exclusively on the multi-mesh.

**Embedding the input features** As part of the encoder we first embed the features of each of the grid nodes, mesh nodes, mesh edges, grid to mesh edges, and mesh to grid edges into a latent space of fixed size using five multi-layer perceptrons (MLP),

$$\begin{aligned}
\mathbf{v}_i^{\text{G}} &= \text{MLP}^{\text{embedder}}_{\mathcal{V}^{\text{G}}}(\mathbf{v}_i^{\text{G,features}}) \\
\mathbf{v}_i^{\text{M}} &= \text{MLP}^{\text{embedder}}_{\mathcal{V}^{\text{M}}}(\mathbf{v}_i^{\text{M,features}}) \\
\mathbf{e}^{\text{M}}_{v_s^{\text{M}} \to v_r^{\text{M}}} &= \text{MLP}^{\text{embedder}}_{\mathcal{E}^{\text{M}}}(\mathbf{e}^{\text{M,features}}_{v_s^{\text{M}} \to v_r^{\text{M}}}) \\
\mathbf{e}^{\text{G2M}}_{v_s^{\text{G}} \to v_r^{\text{M}}} &= \text{MLP}^{\text{embedder}}_{\mathcal{E}^{\text{G2M}}}(\mathbf{e}^{\text{G2M,features}}_{v_s^{\text{G}} \to v_r^{\text{M}}}) \\
\mathbf{e}^{\text{M2G}}_{v_s^{\text{M}} \to v_r^{\text{G}}} &= \text{MLP}^{\text{embedder}}_{\mathcal{E}^{\text{M2G}}}(\mathbf{e}^{\text{M2G,features}}_{v_s^{\text{M}} \to v_r^{\text{G}}})
\end{aligned} \tag{A.6}$$

**Grid2Mesh GNN** Next, in order to transfer information of the state of atmosphere from the grid nodes to the mesh nodes, we perform a single message passing step over the Grid2Mesh bipartite subgraph $\mathcal{G}_{\text{G2M}}(\mathcal{V}^{\text{G}}, \mathcal{V}^{\text{M}}, \mathcal{E}^{\text{G2M}})$ connecting grid nodes to mesh nodes. This update is performed using an interaction network (Battaglia et al., 2016, 2018), augmented to be able to work with multiple node types (Allen et al., 2022). First, each of the Grid2Mesh edges are updated using information from the adjacent nodes,

$$\mathbf{e}^{\text{G2M}}_{v_s^{\text{G}} \to v_r^{\text{M}}}{}' = \text{MLP}^{\text{Grid2Mesh}}_{\mathcal{E}^{\text{G2M}}}([\mathbf{e}^{\text{G2M}}_{v_s^{\text{G}} \to v_r^{\text{M}}}, \mathbf{v}_s^{\text{G}}, \mathbf{v}_r^{\text{M}}]). \tag{A.7}$$

Then each of the mesh nodes is updated by aggregating information from all of the edges arriving at that mesh node:

$$\mathbf{v}_i^{\text{M}\prime} = \text{MLP}^{\text{Grid2Mesh}}_{\mathcal{V}^{\text{M}}}\Big([\mathbf{v}_i^{\text{M}}, \sum_{e^{\text{G2M}}_{v_s^{\text{G}} \to v_r^{\text{M}}} : v_r^{\text{M}} = v_i^{\text{M}}} \mathbf{e}^{\text{G2M}}_{v_s^{\text{G}} \to v_r^{\text{M}}}{}']\Big). \tag{A.8}$$

Each of the grid nodes are also updated, but with no aggregation, because grid nodes are not receivers of any edges in the Grid2Mesh subgraph,

$$\mathbf{v}_i^{\text{G}\prime} = \text{MLP}^{\text{Grid2Mesh}}_{\mathcal{V}^{\text{G}}}\big(\mathbf{v}_i^{\text{G}}\big). \tag{A.9}$$

After updating all three elements, the model includes a residual connection, and for simplicity of the notation, reassigns the variables,

$$\begin{aligned}
\mathbf{v}_i^{\text{G}} &\leftarrow \mathbf{v}_i^{\text{G}} + \mathbf{v}_i^{\text{G}\prime}, \\
\mathbf{v}_i^{\text{M}} &\leftarrow \mathbf{v}_i^{\text{M}} + \mathbf{v}_i^{\text{M}\prime}, \\
\mathbf{e}^{\text{G2M}}_{v_s^{\text{G}} \to v_r^{M}} &\leftarrow \mathbf{e}^{\text{G2M}}_{v_s^{\text{G}} \to v_r^{M}} + \mathbf{e}^{\text{G2M}}_{v_s^{\text{G}} \to v_r^{M}}{}'.
\end{aligned} \tag{A.10}$$

### A.2.6. Processor

The processor is a deep GNN that operates on the Mesh subgraph $\mathcal{G}_{\text{M}}(\mathcal{V}^{\text{M}}, \mathcal{E}^{\text{M}})$ which only contains the Mesh nodes and and the Mesh edges. Note the Mesh edges contain the full multi-mesh, with not only the edges of $M^6$, but all of the edges of $M^5$, $M^4$, $M^3$, $M^2$, $M^1$ and $M^0$, which will enable long distance communication.

**Multi-mesh GNN** A single layer of the Mesh GNN is standard interaction network (Battaglia et al., 2016, 2018) which first updates each of the mesh edges using information of the adjacent nodes:

$$\mathbf{e}^{\text{M}}_{v^M_{\text{s}} \to v^M_{\text{r}}}{}' = \text{MLP}^{\text{Mesh}}_{\mathcal{E}^{\text{M}}}([\mathbf{e}^{\text{M}}_{v^M_{\text{s}} \to v^M_{\text{r}}}, \mathbf{v}^M_s, \mathbf{v}^M_r]). \tag{A.11}$$

Then it updates each of the mesh nodes, aggregating information from all of the edges arriving at that mesh node:

$$\mathbf{v}^{\text{M}}_i{}' = \text{MLP}^{\text{Mesh}}_{\mathcal{V}^{\text{M}}}\Big([\mathbf{v}^{\text{M}}_i, \sum_{e^{\text{M}}_{v^{\text{M}}_{\text{s}} \to v^{\text{M}}_{\text{r}}} : v^{\text{M}}_{\text{r}} = v^{\text{M}}_i} \mathbf{e}^{\text{M}}_{v^{\text{M}}_{\text{s}} \to v^{\text{M}}_{\text{r}}}{}'\Big]) \tag{A.12}$$

And after updating both, the representations are updated with a residual connection and for simplicity of the notation, also reassigned to the input variables:

$$\begin{aligned} \mathbf{v}^{\text{M}}_i &\leftarrow \mathbf{v}^{\text{M}}_i + \mathbf{v}^{\text{M}}_i{}' \\ \mathbf{e}^{\text{M}}_{v^M_{\text{s}} \to v^M_{\text{r}}} &\leftarrow \mathbf{e}^{\text{M}}_{v^N_{\text{s}} \to v^M_{\text{r}}} + \mathbf{e}^{\text{M}}_{v^M_{\text{s}} \to v^M_{\text{r}}}{}' \end{aligned} \tag{A.13}$$

The previous paragraph describes a single layer of message passing, but following a similar approach to Pfaff et al. (2021); Sanchez-Gonzalez et al. (2020) we applied this layer iteratively 16 times, using unshared neural network weights for the MLPs in each layer.

### A.2.7. Decoder

The role of the decoder is to bring back information to the grid, and extract an output.

**Mesh2Grid GNN** Analogous to the Grid2Mesh GNN, the Mesh2Grid GNN performs a single message passing over the Mesh2Grid bipartite subgraph $\mathcal{G}_{\text{M2G}}(\mathcal{V}^{\text{G}}, \mathcal{V}^{\text{M}}, \mathcal{E}^{\text{M2G}})$. The Grid2Mesh GNN is functionally equivalent to the Mesh2Grid GNN, but using the Mesh2Grid edges to send information in the opposite direction. The GNN first updates each of the Grid2Mesh edges using information of the adjacent nodes:

$$\mathbf{e}^{\text{M2G}}_{v^M_{\text{s}} \to v^{\text{G}}_{\text{r}}}{}' = \text{MLP}^{\text{Mesh2Grid}}_{\mathcal{E}^{\text{M2G}}}([\mathbf{e}^{\text{M2G}}_{v^M_{\text{s}} \to v^{\text{G}}_{\text{r}}}, \mathbf{v}^M_s, \mathbf{v}^{\text{G}}_r]) \tag{A.14}$$

Then it updates each of the grid nodes, aggregating information from all of the edges arriving at that grid node:

$$\mathbf{v}^{\text{G}}_i{}' = \text{MLP}^{\text{Mesh2Grid}}_{\mathcal{V}^{\text{G}}}\Big([\mathbf{v}^{\text{G}}_i, \sum_{e^{\text{M2G}}_{v^{\text{M}}_{\text{s}} \to v^{\text{G}}_{\text{r}}} : v^{\text{G}}_{\text{r}} = v^{\text{G}}_i} \mathbf{e}^{\text{M2G}}_{v^{\text{M}}_{\text{s}} \to v^{\text{G}}_{\text{r}}}{}'\Big]). \tag{A.15}$$

In this case we do not update the mesh nodes, as they won't play any role from this point on.

Here again we add a residual connection, and for simplicity of the notation, reassign the variables, this time only for the grid nodes, which are the only ones required from this point on:

$$\mathbf{v}^{\text{G}}_i \leftarrow \mathbf{v}^{\text{G}}_i + \mathbf{v}^{\text{G}}_i{}'. \tag{A.16}$$

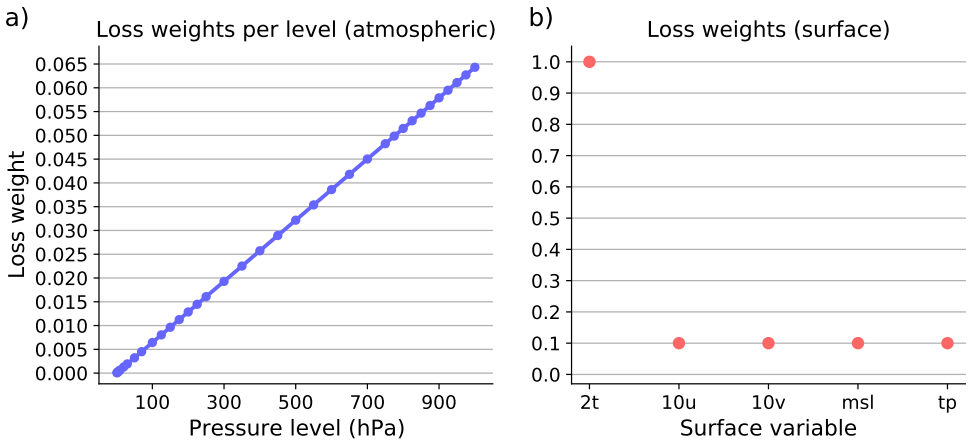


Figure A.1 | **Training loss weights.** (a) Loss weights per pressure level, for atmospheric variables. (b) Loss weights for surface variables.

**Output function** Finally the prediction $\hat{\mathbf{y}}_i$ for each of the grid nodes is produced using another MLP,

$$\hat{\mathbf{y}}_i^{\mathrm{G}} = \mathrm{MLP}^{\mathrm{Output}}_{\mathcal{V}^{\mathrm{G}}}(\mathbf{v}_i^{\mathrm{G}}) \tag{A.17}$$

which contains all 227 predicted variables for that grid node. Similar to Pfaff et al. (2021); Sanchez-Gonzalez et al. (2020), the next weather state, $\hat{X}^{t+1}$, is computed by adding the per-node prediction, $\hat{Y}^t$, to the input state for all grid nodes,

$$\hat{X}^{t+1} = \mathrm{GraphCast}(X^t, X^{t-1}) = X^t + \hat{Y}^t. \tag{A.18}$$

### A.2.8. Other model details

**Input normalization** Similar to Pfaff et al. (2021); Sanchez-Gonzalez et al. (2020), we normalized all inputs. For each physical variable, we computed the per-pressure level mean and standard deviation over 1979-2015, and used that to normalize them to zero mean and unit variance. For relative edge distances and lengths, we normalized the features to the length of the longest edge. For simplicity, we omit this output normalization from the notation.

**Output normalization** Because our model outputs a difference, $\hat{Y}^t$, which, during inference, is added to $X^t$ to produce $\hat{X}^{t+1}$, we normalized the output of the model by computing per-pressure level mean and standard deviation statistics for the time difference $Y^t = X^{t+1} - X^t$ of each variable. When the GNN produces an output, we multiply this output by this standard deviation, and add the mean to $\hat{Y}^t$ before computing $\hat{X}^{t+1}$, as in Appendix Equation A.18. For simplicity, we omit this output normalization from the notation.

**Neural network parameterizations** The neural networks within GraphCast are all MLPs, with one hidden layer, and hidden and output layers sizes of 512 (except the final layer of the Decoder's MLP, whose output size is 227, matching the number of predicted variables for each grid node). We chose the "swish" (Ramachandran et al., 2017) activation function for all MLPs. All MLPs are followed by a LayerNorm (Ba et al., 2016) layer (again, except for the Decoder's MLP).

## A.3. Training details

**Training objective** The training objective (Equation (2), repeated here for ease of reading) is defined as the mean square error (MSE) between the target output $X$ and predicted output $\hat{X}$,

$$\mathcal{L}_{\text{MSE}} = \underbrace{\frac{1}{|D_{\text{batch}}|} \sum_{d_0 \in D_{\text{batch}}}}_{\text{forecast date-time}} \underbrace{\frac{1}{T} \sum_{\tau \in 1:T_{\text{train}}}}_{\text{lead time}} \underbrace{\frac{1}{|G_{0.25^\circ}|} \sum_{i \in G_{0.25^\circ}}}_{\text{spatial location}} \underbrace{\sum_{j \in J}}_{\text{variable-level}} s_j w_j a_i \underbrace{(\hat{x}_{i,j}^{d_0+\tau} - x_{i,j}^{d_0+\tau})^2}_{\text{squared error}} \tag{A.19}$$

where

- $\tau \in 1 : T_{\text{train}}$ are the lead times that correspond to the $T_{\text{train}}$ autoregressive steps.
- $d_0 \in D_{\text{batch}}$ represent forecast initialization date-times in a batch of forecasts in the training set,
- $j \in J$ indexes the variable, and for atmospheric variables the pressure level. E.g. $J = \{\text{z1000}, \text{z850}, \ldots, \text{2t}, \text{msl}\}$,
- $i \in G_{0.25^\circ}$ are the location (latitude and longitude) coordinates in the grid,
- $\hat{x}_{j,i}^{d_0+\tau}$ and $x_{j,i}^{d_0+\tau}$ are predicted and target values for some variable-level, location, and lead time,,
- $s_j$ is the per-variable-level inverse variance of time differences,
- $w_j$ is the per-variable-level loss weight,
- $a_i$ is the area of the latitude-longitude grid cell, which varies with latitude, and is normalized to unit mean over the grid.

In order to build a single scalar loss, we took the average across latitude-longitude, pressure levels, variables, lead times, and batch size. We averaged across latitude-longitude axes, with a weight proportional to the latitude-longitude cell size (normalized to mean 1). We applied uniform averages across time and batch.

The $s_j = \mathbb{V}_{i,t}\left[x_{i,j}^{t+1} - x_{i,j}^{t}\right]^{-1}$ are per-variable-level inverse variance estimates of the time differences, which aim to standardize the targets (over consecutive steps) to unit variance. These were estimated from the training data. We then applied per-variable-level loss weights, $w_j$. For atmospheric variables, we averaged across levels, with a weight proportional to the pressure of the level (normalized to unit mean), as shown in Appendix Figure A.1a. We use pressure here as a proxy for the density (Keisler, 2022). We tuned the loss weights for the surface variables during model development, so as to produce roughly comparable validation performance across all variables: the weight on 2t was 1.0, and the weights on 10u, 10v, msl, and tp were each 0.1, as shown in Appendix Figure A.1b. The loss weights across all variables sum to 7.4, i.e., ($6 \times 1.0$ for the atmospheric variables, plus $(1.0 + 0.1 + 0.1 + 0.1 + 0.1)$ for the surface variables listed above, respectively).

**Training on autoregressive objective** In order to improve our model's ability to make accurate forecasts over more than one step, we used an autoregressive training regime, where the model's predicted next step was fed back in as input for predicting the next step. The final GraphCast version was trained on 12 autoregressive steps, following a curriculum training schedule (see Appendix A.3). The optimization procedure computed the loss on each step of the forecast, with respect to the corresponding ground truth step, error gradients with respect to the model parameters were backpropagated through the full unrolled sequence of model iterations (i.e., using backpropagation-through-time).

**Optimization** The training objective function was minimized using gradient descent, with mini-batches. We sampled ground truth trajectories from our ERA5 training dataset, with replacement, for


Figure A.2 | **Training schedule.** (a) First phase of training. (b) Second phase of training. (c) Third phase of training.

batches of size 32. We used the AdamW optimizer (Kingma and Ba, 2014; Loshchilov and Hutter, 2017) with parameters $(\text{beta1} = 0.9, \text{beta2} = 0.95)$. We used weight decay of 0.1 on the weight matrices. We used gradient (norm) clipping with a maximum norm value of 32.

**Curriculum training schedule** Training the model was conducted using a curriculum of three phases, which varied the learning rates and number of autoregressive steps. The first phase consisted of 1000 gradient descent updates, with one autoregressive step, and a learning rate schedule that increased linearly from 0 to 1e−3 (Appendix Figure A.2a). The second phase consisted of 299,000 gradient descent updates, again with one autoregressive step, and a learning rate schedule that decreased back to 0 with half-cosine decay function (Appendix Figure A.2b). The third phase consisted of 11,000 gradient descent updates, where the number of autoregressive steps increased from 2 to 12, increasing by 1 every 1000 updates, and with a fixed learning rate of 3e−7 (Appendix Figure A.2c).

**Software and hardware stack** We use JAX (Bradbury et al., 2018), Haiku (Hennigan et al., 2020), Jraph (Godwin* et al., 2020), Optax, Jaxline (Babuschkin et al., 2020) and xarray (Hoyer and Hamman, 2017) to build and train our models. We distribute the batch across 32 Cloud TPU v4 devices. We leverage bfloat16 floating point precision and gradient check-pointing (Chen et al., 2016) to fit training on long trajectories into the 32GB of the TPU v4 device.

## A.4. Evaluation details

### A.4.1. Evaluation metrics

Our goal is to assess the performance of our model in a way that allows for a consistent comparison with the HRES 10-day forecast and other baseline ML models. We therefore focus on the initialization times 00z and 12z, which are when HRES's 10-day forecasts are initialized, as well as what Bi et al. (2022) use. All results are reported for the year 2018 (our test set), with the first forecast initialization time being `2018-01-01_00:00:00` and the last `2018-12-21_12:00:00`. We excluded the last 10 days of the year so that the last day of the 10-day ground truth forecast was still within 2018.

Note, using 00z and 12z forecast times means that ERA5's state is informed by observations from +9 hours ahead (because ERA5's assimilation windows are 21z-09z and 09z-21z, as described in Appendix A.1). By contrast, HRES's assimilation window is only +3 hours ahead. Therefore, the

ERA5 initial state can carry information from further into the future than HRES's. However, after 2022-11-22 HRES forecasts are based on upgraded NWPs than ERA5's version of HRES, which may balance potential advantages of using a longer assimilation window.

All metrics were computed using float32 precision and reported using the native scale of the variables without normalization.

**Root mean square error (RMSE)** We quantified forecast skill for a given variable, $x_j$, and lead time, $\tau = t\Delta d$ using a latitude-weighted root mean square error (RMSE) given by

$$\mathcal{L}^{j,\tau}_{\text{RMSE}} = \frac{1}{|D_{\text{eval}}|} \sum_{d_0 \in D_{\text{eval}}} \sqrt{\frac{1}{|G_{0.25^\circ}|} \sum_{i \in G_{0.25^\circ}} a_i \left(\hat{x}^{d_0+\tau}_{j,i} - x^{d_0+\tau}_{j,i}\right)^2} \tag{A.20}$$

where

- $d_0 \in D_{\text{eval}}$ represent forecast initialization date-times in the evaluation dataset,
- $j \in J$ indexes the variable and level, e.g., $J = \{\texttt{z1000}, \texttt{z850}, \ldots, \texttt{2t}, \texttt{msl}\}$,
- $i \in G_{0.25^\circ}$ are the location (latitude and longitude) coordinates in the grid,
- $\hat{x}^{d_0+\tau}_{j,i}$ and $x^{d_0+\tau}_{j,i}$ are predicted and target values for some variable-level, location, and lead time,
- $a_i$ is the area of the latitude-longitude grid cell (normalized to unit mean over the grid) which varies with latitude.

**Anomaly correlation coefficient (ACC)** We also computed the anomaly correlation coefficient for a given variable, $x_j$, and lead time, $\tau = t\Delta d$, according to

$$\mathcal{L}^{j,\tau}_{\text{ACC}} = \frac{1}{|D_{\text{eval}}|} \sum_{d_0 \in D_{\text{eval}}} \frac{\sum_{i \in G_{0.25^\circ}} a_i \left(\hat{x}^{d_0+\tau}_{j,i} - C^{d_0+\tau}_{j,i}\right)\left(x^{d_0+\tau}_{j,i} - C^{d_0+\tau}_{j,i}\right)}{\sqrt{\left[\sum_{i \in G_{0.25^\circ}} a_i \left(\hat{x}^{d_0+\tau}_{j,i} - C^{d_0+\tau}_{j,i}\right)^2\right]\left[\sum_{i \in G_{0.25^\circ}} a_i \left(x^{d_0+\tau}_{j,i} - C^{d_0+\tau}_{j,i}\right)^2\right]}} \tag{A.21}$$

where $C^{d_0+\tau}_{j,i}$ is the climatological mean for a given variable, level, latitude and longitude, and for the day-of-year containing the validity time $d_0 + \tau$. All other variables are defined as above. Climatological means were computed using ERA5 data between 1993 and 2016.

## A.5. Details of baseline comparisons

When comparing different releases of HRES to one another, ECMWF measures HRES's accuracy by computing prediction errors against HRES's own analysis. Generally, verifying a model against its own analysis gives the best skill estimates (Swinbank et al., 2016). By contrast, evaluating HRES against ERA5, for example, could lead to misinterpreting or underestimating HRES's skill, because HRES is not designed to predict ERA5 values. The reason is that HRES is initialized with its own analysis, not ERA5 values. By measuring HRES against ERA5, even if HRES were a perfect weather model, the results would still exhibit a non-zero error, simply because it is making predictions from different initial conditions. This was why we created the HRES-fc0 dataset (Appendix A.1.2), and evaluated HRES skill against that as ground truth.

Figure 8 shows the different effects of computing HRES's skill using ERA5 as ground truth (grey lines), versus using HRES-fc0 as ground truth (black lines). Using ERA5 leads to a systematically worse skill, relative to using HRES-fc0. The effect is most prominent at short lead times, and as

the lead time increases, the two skill curves converge to similar values. Bi et al. (2022) measured HRES's skill by using ERA5 as ground truth, and Figure 8 shows that in some cases Pangu-Weather outperforms HRES's skill when evaluated against ERA5, but not when evaluated against HRES-fc0. It was not specified in Kurth et al. (2022); Pathak et al. (2022) whether ERA5 was used as ground truth for their IFS skill computations. Keisler (2022) used NOAA's GFS's reported skill, which is likely an appropriate skill measure.

To determine how GraphCast's performance compares to competing ML methods, we focus on Pangu-Weather as it is the strongest baseline. Bi et al. (2022) report Pangu-Weather's 7-day forecast accuracy (RMSE and ACC) on: z500, t500, t850, q500, u500, v500, 2t, 10u, 10v, and msl. As shown in Figure 8, GraphCast (blue lines) outperforms Pangu-Weather Bi et al. (2022) (red lines) on 99.2% of targets. For the surface variables (2t, 10u, 10v, msl), GraphCast's error in the first several days is around 10-20% lower, and over the longer lead times plateaus to around 7-10% lower error. The only 2 (of the 252 total) metrics on which Pangu-Weather outperformed GraphCast was z500, at lead times 6 and 12 hours, where GraphCast had 1.7% higher average RMSE (Figure 8a,e).

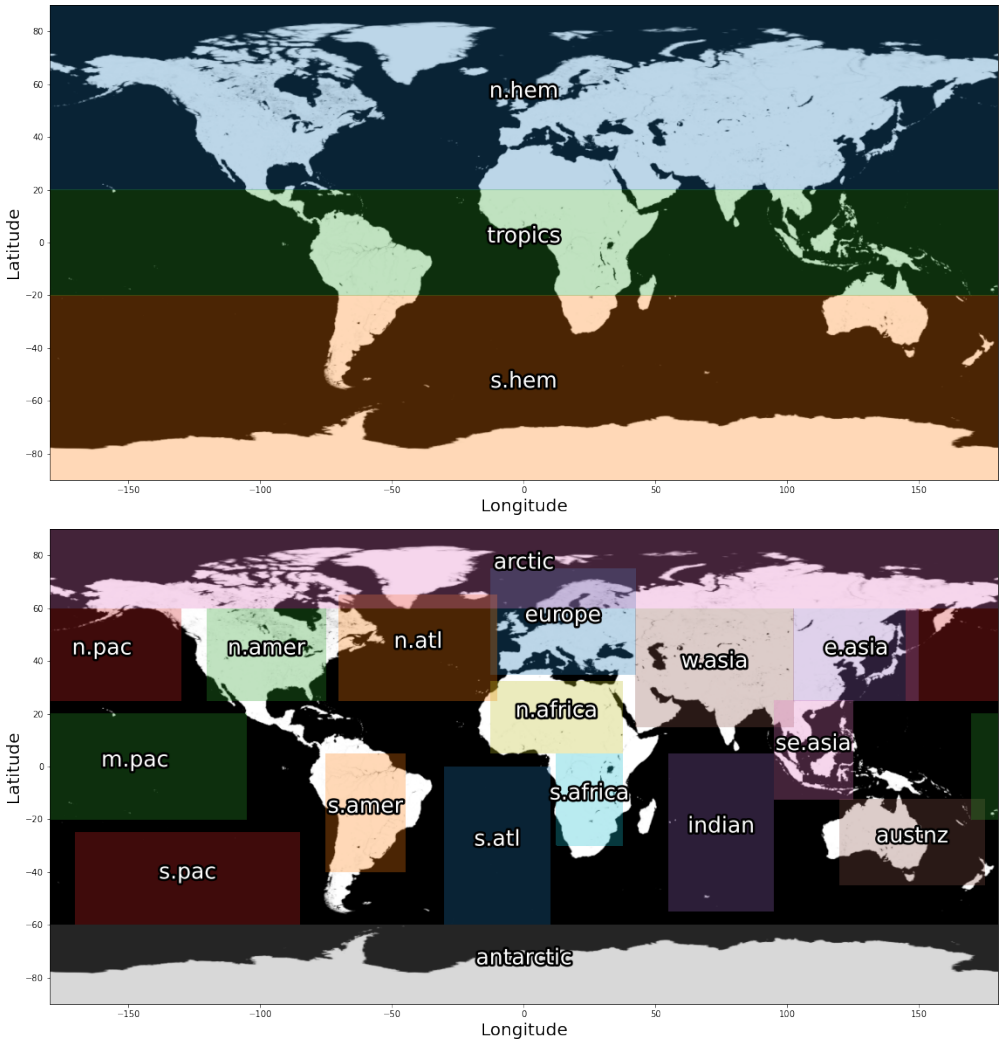


Figure A.3 | **Region specification for the regional analysis.** We use the same regions and naming convention as in the ECMWF scorecards (https://sites.ecmwf.int/ifs/scorecards/scorecards-47r3HRES.html), and add some additional regions for better coverage of the entire planet. Per-region evaluation is provided in Appendix Figure A.4 and Appendix Figure A.5.

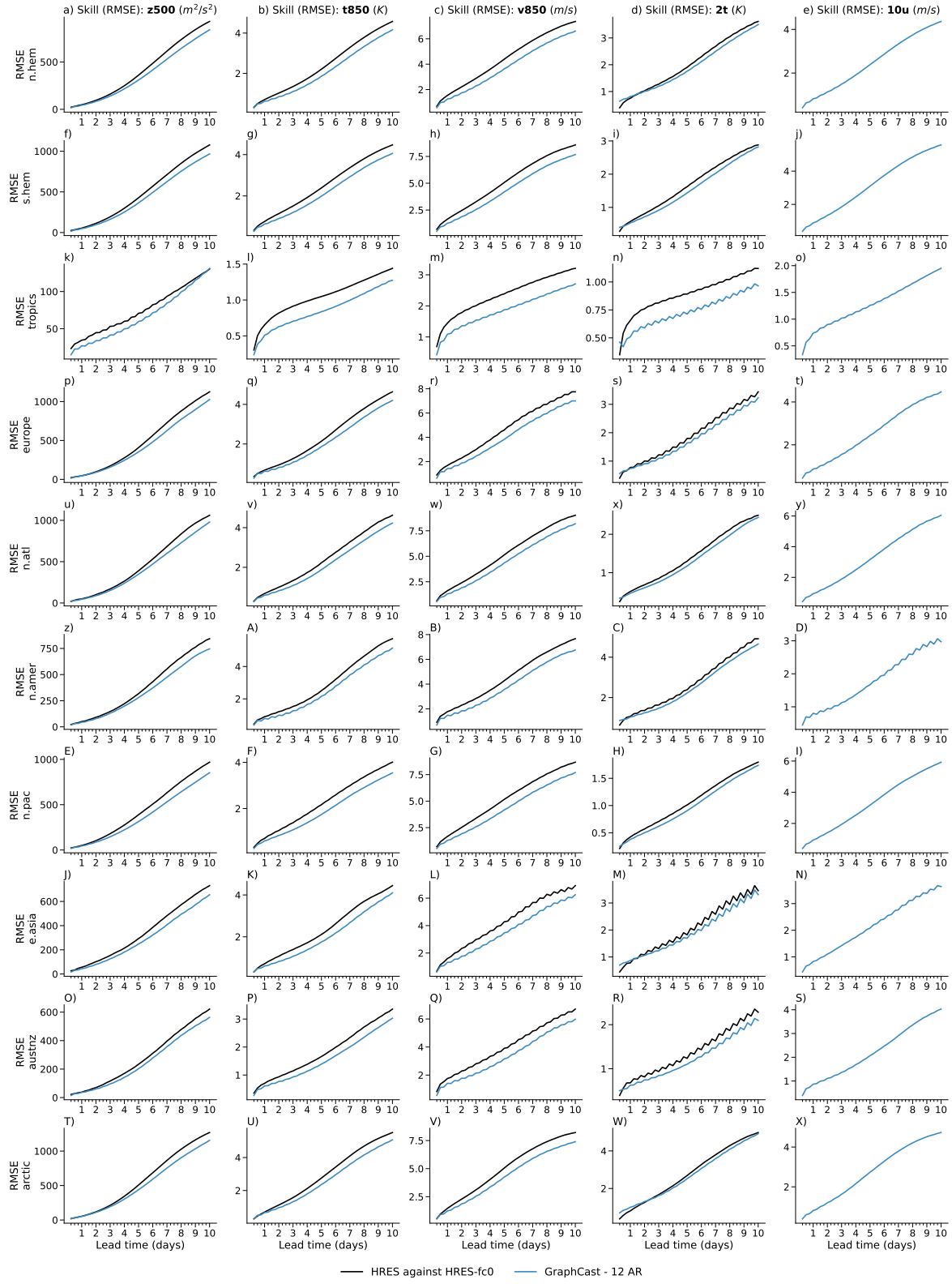


Figure A.4 | **Skill (RMSE) of GraphCast versus HRES, per-region.** Each column is a different variable (and level), for a representative set of variables. Each row is a different region. The x-axis is lead time, in days. The y-axis is RMSE, with units specified in the column titles. GraphCast's RMSEs are the blue lines, and HRES's RMSEs are the black lines. The regions are: `n.hem`, `s.hem`, `tropics`, `europe`, `n.atl`, `n.amer`, `n.pac`, `e.asia`, `austnz`, and `arctic`. See Appendix Figure A.3 for a legend of the region names.

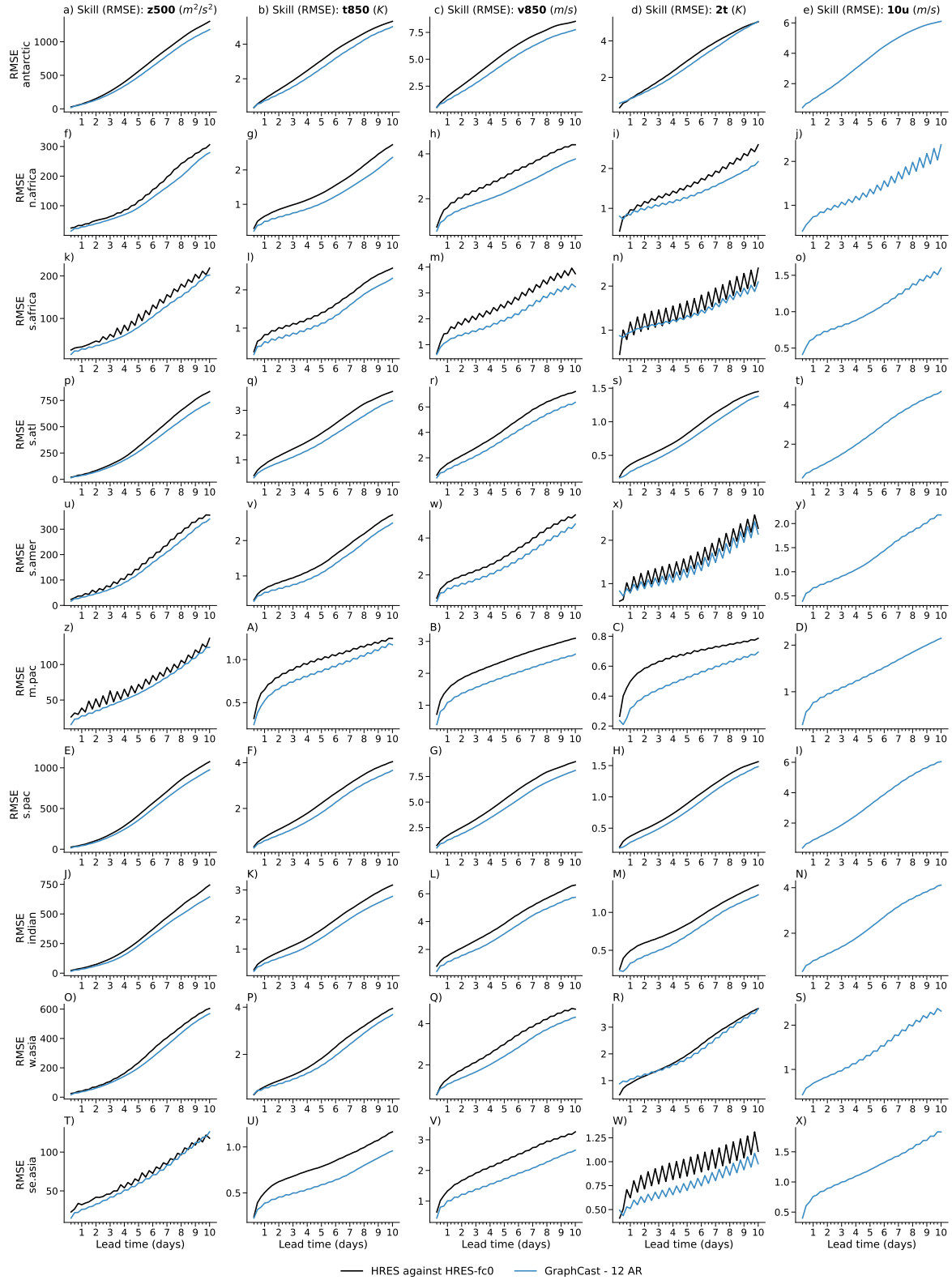


Figure A.5 | **Skill (RMSE) of GraphCast versus HRES, per-region.** This plot is the same as Appendix Figure A.4, except for regions (in Appendix Figure A.3): `antarctic`, `n.africa`, `s.africa`, `s.atl`, `s.amer`, `m.pac`, `s.pac`, `indian`, `w.asia`, and `se.asia`.