OGS National Institute of Oceanography and Applied Geophysics

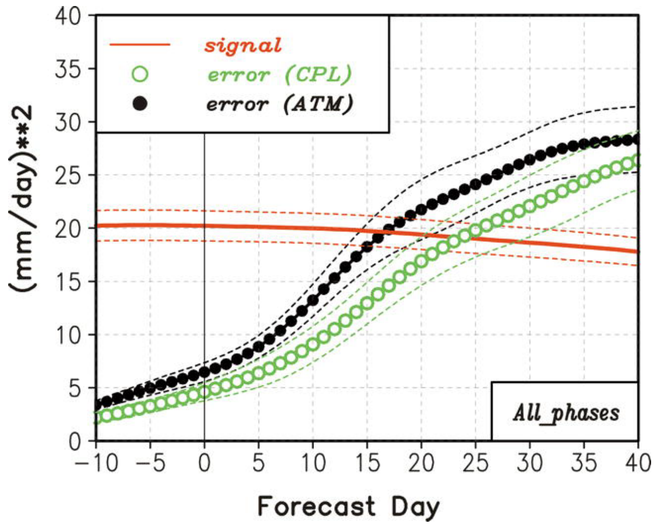
Lessons from Weather

Adapting GraphCast for Global Ocean Forecasting

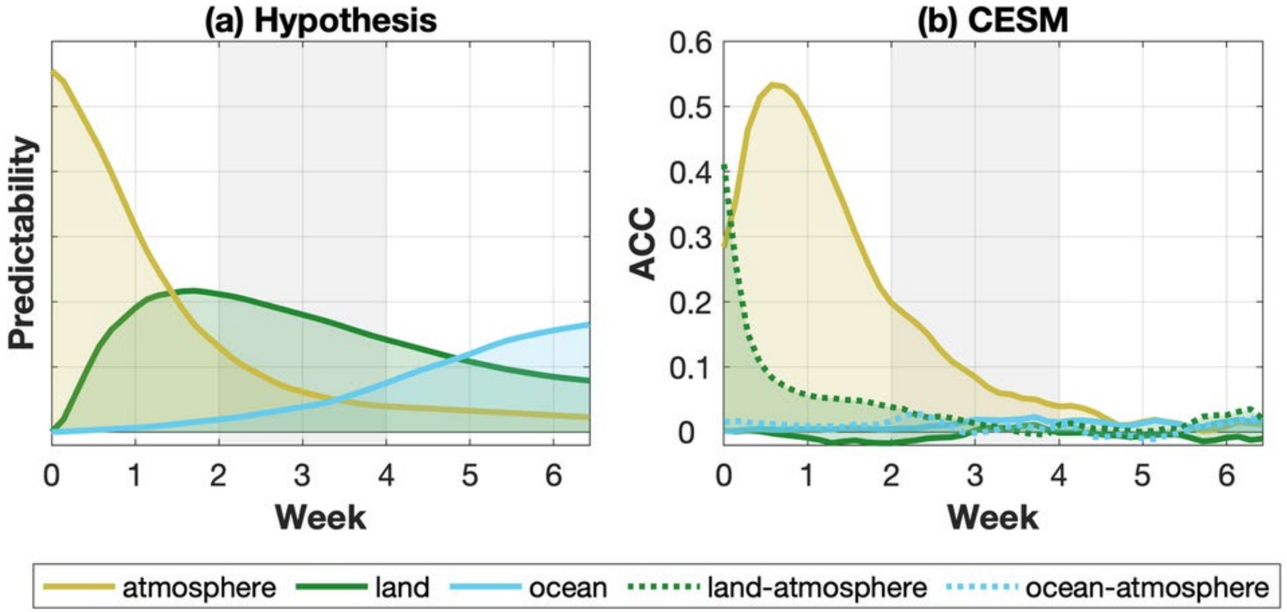
Stefano Campanella

Stefano Salon, Stefano Querin, Luca Bortolussi

Motivation for global ocean modelling at S2S timescales



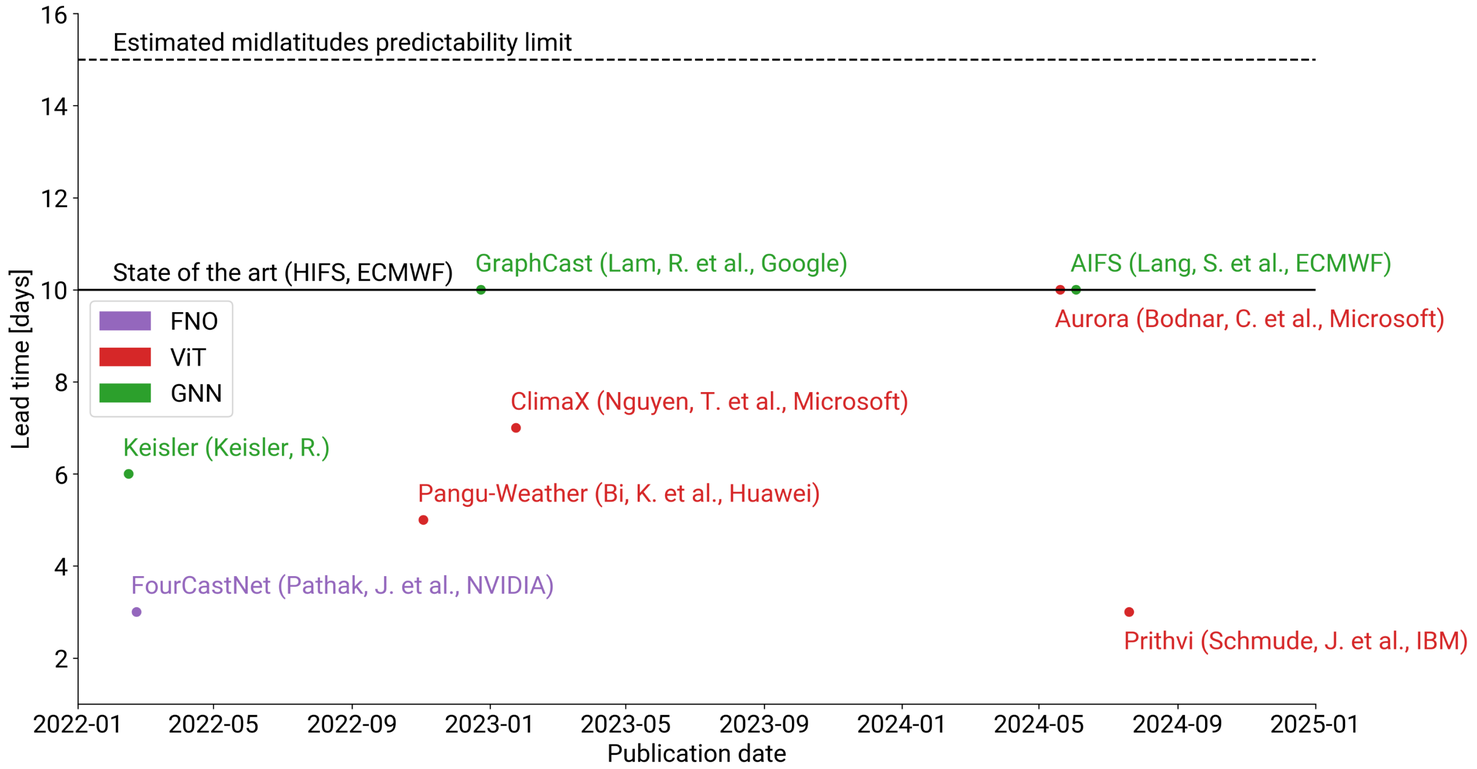
The rainfall signals and forecast errors (mm day^{-1})^2 as function of lead time averaged in the eastern Indian Ocean for MISO events using an atmosphere–ocean coupled model (CPL) and an atmosphere-only model (ATM).



Richter, J.H., Glanville, A.A., King, T. et al. Quantifying sources of subseasonal prediction skill in CESM2. npj Clim Atmos Sci 7, 59 (2024). https://doi.org/10.1038/s41612-024-00595-4

Fu, X., et al., 2007: Impact of Atmosphere–Ocean Coupling on the Predictability of Monsoon Intraseasonal Oscillations. J. Atmos. Sci., 64, 157–174, https://doi.org/10.1175/JAS3830.1.

Data-driven weather model zoo



Def^n *ML timeseries forecasting models trained on a reanalysis. Diverse architectures, no clear "winner".*

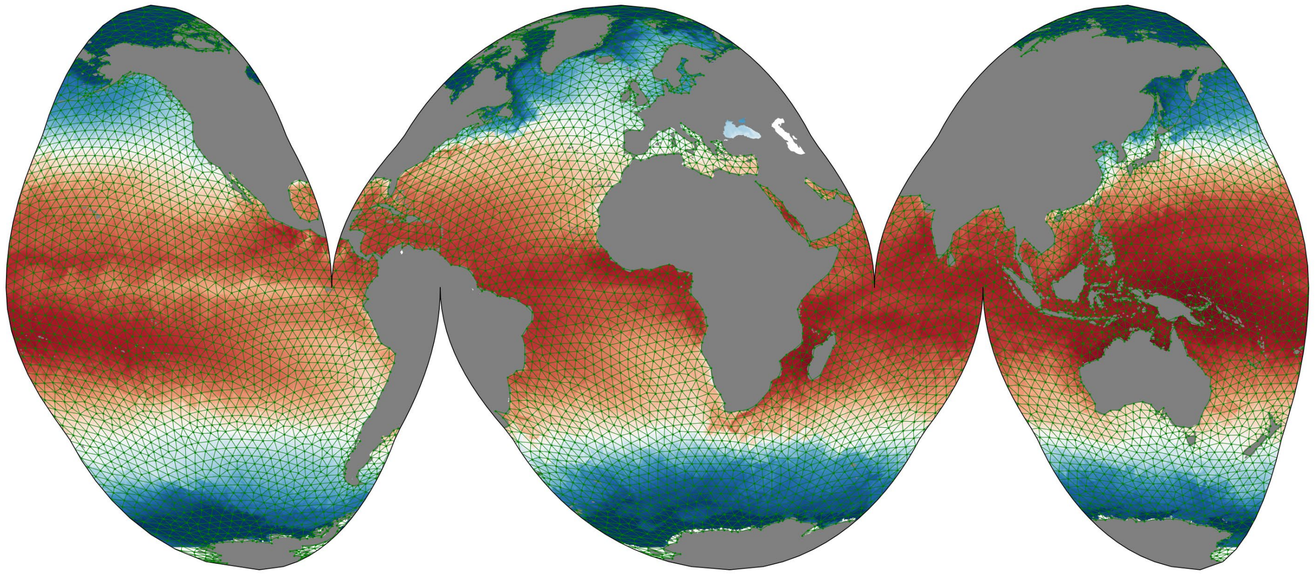
Advantages

- Skills (should) scales with amt. of historical data.
- Subgrid-scale (might) be learnt implicitly.
- Extreme speedup: allowing for large ensembles.

GraphCast is peer-reviewed, open-source, fairly-well documented, still state-of-the-art, no "atmospheric inductive bias".

This suggested the idea of adapting to ocean modelling.

ARCO-OCEAN



Dataset derived from **GLORYS12**, **WAVERYS**, **ERA5**, and **GLOFAS**.

It contains physical properties of the **ocean, sea ice, and waves**, for a period between the 1st of January 1993 and the 30th of June 2021.

The dataset includes also **atmospheric and hydrological variables** that would be needed as boundary conditions and used to drive a numerical simulation.

Horizontal resolution of **0.25°**[1], 10 depth levels between 0 and 1000m.

[1] There are plans to extend the dataset to resolution of 2°, 1°, and 0.5°

Check it out at https://registry.opendata.aws/ogs-arco-ocean/!

Technical interlude: storing a large dataset for ML

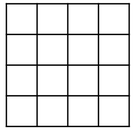
Zarr

```
<xarray.Dataset> Size: 2TB
Dimensions:    (time: 10408, latitude: 721, longitude: 1440, depth: 10)
Coordinates:
  * depth      (depth) float32 40B 0.494 5.078 11.4 21.6 ... 318.1 643.6 902.3
  * latitude   (latitude) float32 3kB -90.0 -89.75 -89.5 ... 89.5 89.75 90.0
  * longitude  (longitude) float32 6kB 0.0 0.25 0.5 0.75 ... 359.2 359.5 359.8
  * time       (time) datetime64[ns] 83kB 1993-01-01 1993-01-02 ... 2021-06-30
Data variables:
```

Lustre filesystem

```
lfs setstripe -c 4 -S 4M data
```

Individual chunks



Xarray

```
xr.open_dataset("/localfs/path/to/arco/dataset", engine="zarr")
```

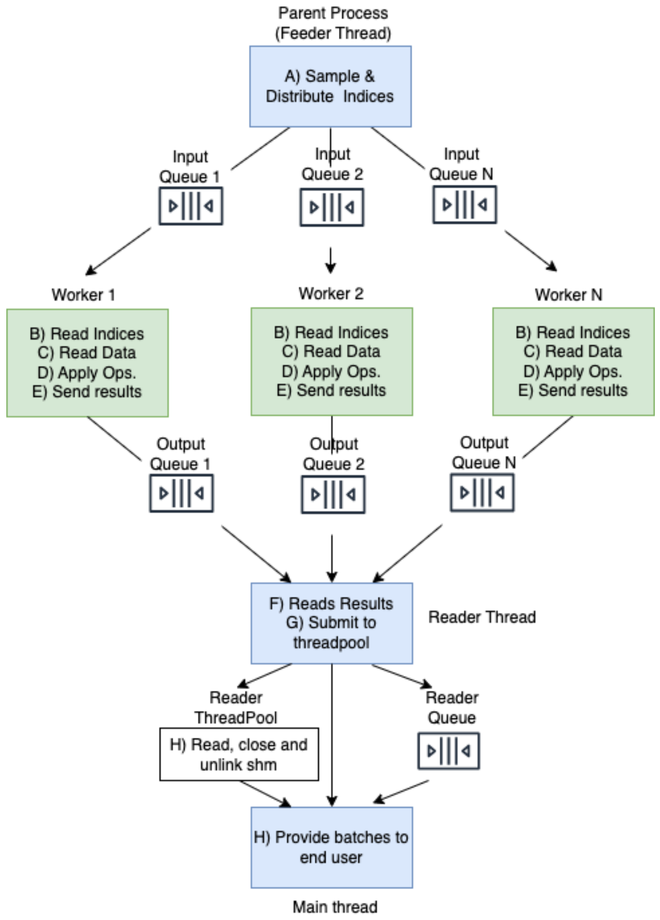
```
<xarray.DataArray 'thetao' (time: 10408, depth: 10, latitude: 721,
                            longitude: 1440)> Size: 428GB
dask.array<concatenate, shape=(10408, 10, 721, 1440), dtype=float32, chunksize=(1, 10, 721,
1440), chunktype=numpy.ndarray>
Coordinates:
  * depth      (depth) float32 40B 0.494 5.078 11.4 21.6 ... 318.1 643.6 902.3
  * latitude   (latitude) float32 3kB -90.0 -89.75 -89.5 ... 89.5 89.75 90.0
  * longitude  (longitude) float32 6kB 0.0 0.25 0.5 0.75 ... 359.2 359.5 359.8
  * time       (time) datetime64[ns] 83kB 1993-01-01 1993-01-02 ... 2021-06-30
Attributes:
    long_name:      Temperature
    standard_name:  sea_water_potential_temperature
    unit_long:      Degrees Celsius
    units:          degrees_C
    valid_max:      21306
    valid_min:      -32766
```

Encoding

```
{'chunks': (1, 10, 721, 1440),
 'preferred_chunks': {'time': 1,
  'depth': 10,
  'latitude': 721,
  'longitude': 1440},
 'compressor': Blosc(cname='lz4', clevel=1, shuffle=SHUFFLE, blocksize=0),
 'filters': None,
 '_FillValue': nan,
 'dtype': dtype('float32')}
```

https://github.com/google-research/arco-era5

Technical interlude: feeding models



During training, feeding the model was the main bottleneck (step bubble).

Current dataloader implementation based on **Grain** and **XArray**, main features:

1. Deterministic.
2. Preemptible (i.e., checkpointable).
3. Data prefetching.
4. Multithreaded (multi-processing still requires polishing).

Original GraphCast training: 32 TPU v4, 4 weeks / 314’000 optimization steps **≈ 7.70 s/step.**
Our training: 8 Leonardo Booster nodes: **≈ 2.0 s/step**

Host memory is the main constraint to large prefetch queues (32 items per process, 8 threads).

https://google-grain.readthedocs.io/en/latest/behind_the_scenes.html

GraphCast in a nutshell: an autoregressive model

GraphCast can be seen as an **autoregressive**, **nonlinear**, **error-in-variable** model, fitted minimizing weighted least squares iteratively using stochastic optimization.

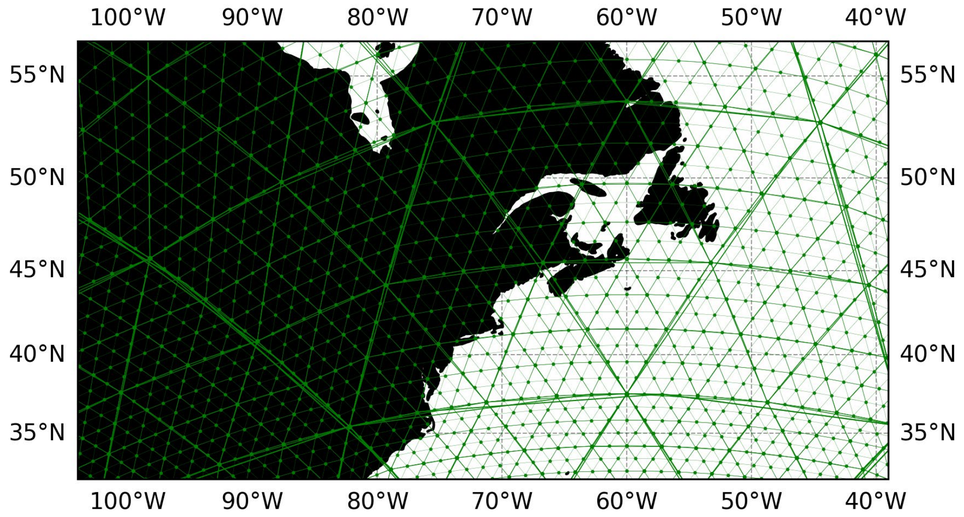
$$\begin{cases} \boldsymbol{x}_n^* = \boldsymbol{x}_{n-1}^* + \boldsymbol{s}\, f_\theta(\boldsymbol{x}_{n-1}^*, \boldsymbol{x}_{n-2}^*, \dots, \boldsymbol{x}_{n-p}^*, n) \\ \boldsymbol{x}_n = \boldsymbol{x}_n^* + \boldsymbol{\varepsilon}_n \end{cases}, \qquad \hat{\theta}(\boldsymbol{x}_n, \boldsymbol{x}_{n+1}, \boldsymbol{x}_{n+2}, \dots, \boldsymbol{x}_{n+k}) = \operatorname{argmin}_\theta \sum_{j=n}^{n+k} \sum_{\alpha_i} \omega_{\alpha_i} \left(\frac{\left(x_j^{\alpha_i} - x_{j-1}^{\alpha_i}\right)}{s^{\alpha_i}} - f_\theta^{\alpha_i}\left(\boldsymbol{x}_{j-1}, \boldsymbol{x}_{j-2}, \dots, \boldsymbol{x}_{j-p}, j\right) \right)^2,$$

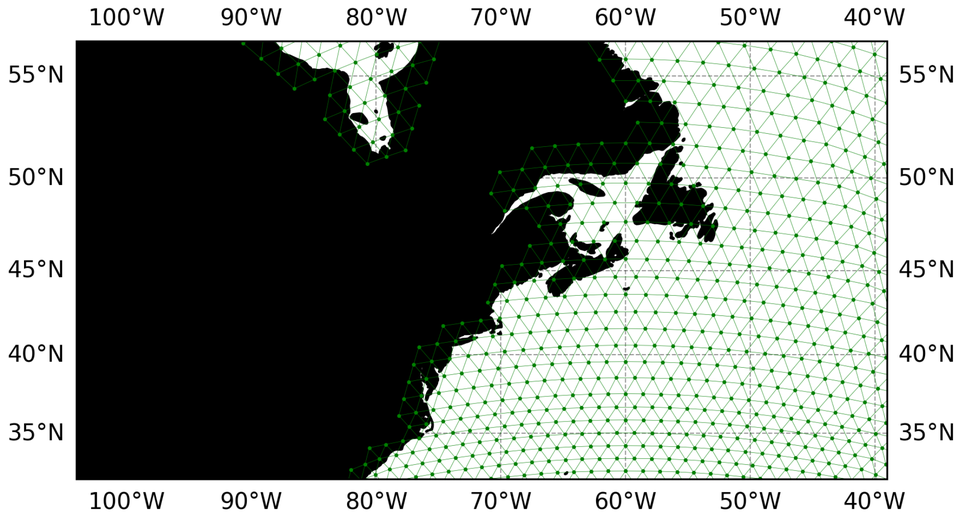
where:

- f_θ is a (message-passing) Graph Neural Network (GNN), and $\boldsymbol{s} = \sqrt{\frac{1}{n}\sum_n (\boldsymbol{x}_n - \boldsymbol{x}_{n-1})^2}$ is an estimate of the variance of the increments,
- $\boldsymbol{x}_n$ is the system state at time n, and $\boldsymbol{x}_n^*$ is the (un-observed) true state at time n, subject to a reconstruction error $\boldsymbol{\varepsilon}_n$,
- ω_{α_i} are weights proportional to the area of a cell (varying with latitude) and to the pressure of the level (adjusted for surface variables with ad-hoc values).

Lam, R., et al. Learning skillful medium-range global weather forecasting. Science382,1416-1421(2023). https://doi.org/10.1126/science.adi2336

Masking the icosahedral multi-mesh



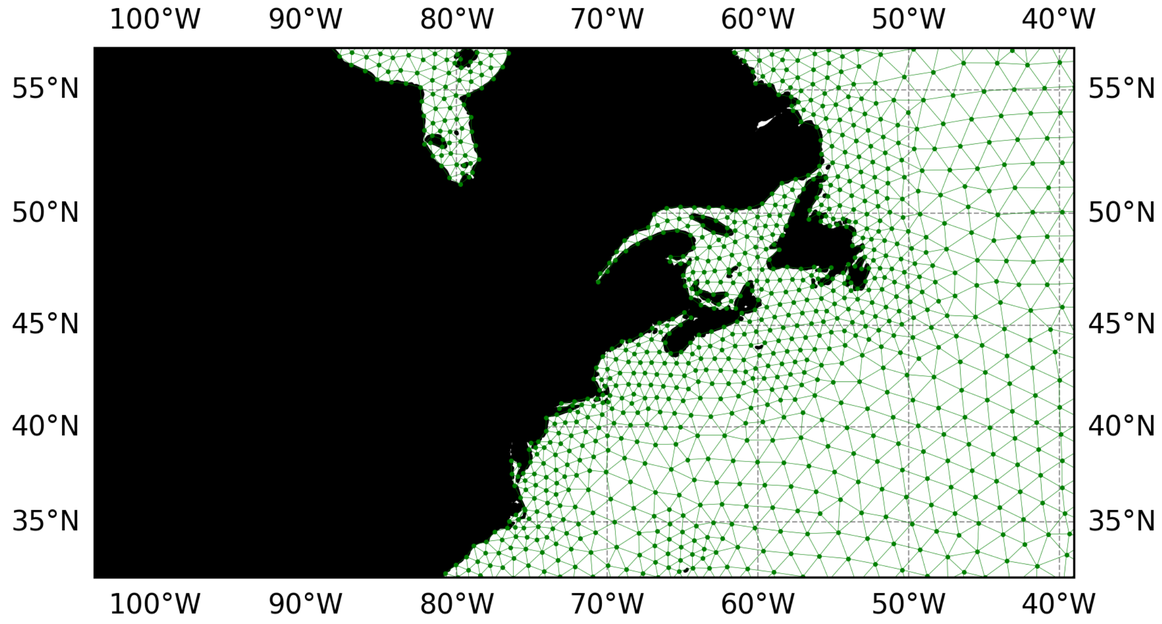


Filter the mesh to include only vertices connected to a cell in the interior of the domain or belonging to a triangle with at least one vertex with that property. The resulting graph has 31,775 vertices and 248,696 edges.

Advantages: non-parametric pruning of the graph, minimal code modifications needed.

Disadvantages: loosing many “hubs” and long-range connections, poor modelling of the coastline, varying mesoscale size not considered.

Using a full-fledged ocean triangle mesh



Isotropic target mesh size fields

$$\delta(\boldsymbol{x}) = \min\left(\delta_1(\boldsymbol{x}), \delta_2(\boldsymbol{x}), \dots\right)$$

$$\alpha_i(\boldsymbol{x}) = \begin{cases} 0 & \text{if} \quad f_i(\boldsymbol{x}) \le f_i^{\min} \\ \dfrac{f_i(\boldsymbol{x}) - f_i^{\min}}{f_i^{\max} - f_i^{\min}} & \text{if} \quad f_i^{\min} < f_i(\boldsymbol{x}) < f_i^{\max} \\ 1 & \text{if} \quad f_i(\boldsymbol{x}) \ge f_i^{\max} \end{cases}$$

$$\delta_i(\boldsymbol{x}) = \delta_i^{\text{small}} + \alpha_i(\boldsymbol{x}) \left(\delta_i^{\text{large}} - \delta_i^{\text{small}}\right)$$

In the mesh on the left we used a single criterion field, the distance from the coast, with lower and upper extremes respectively of 20Km and 200Km.

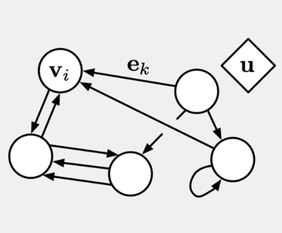
The domain is **meshed in stereographic projection** (2D), then projected back onto the geoid (our fork of Python `seamsh`).

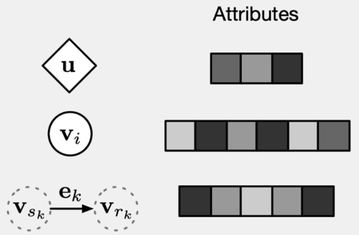
Advantages: accurate description of coastlines, varying mesoscale size

Disadvantages: parameters tuning, **no long-range interactions**, edges with different scales, poorer aspect ratio of elements ⇒ increased graph size

Lambrechts, J., Comblen, R., Legat, V., Geuzaine, C. & Remacle, J.-F. Multiscale mesh generation on the sphere. Ocean Dynamics 58, 461–473 (2008).

Brief recap on GNNs





Here we use "graph" to mean a directed, attributed multi-graph with a global attribute. In our terminology, a node is denoted as $\mathbf{v}_i$, an edge as $\mathbf{e}_k$, and the global attributes as $\mathbf{u}$. We also use s_k and r_k to indicate the indices of the sender and receiver nodes (see below), respectively, for edge k. To be more precise, we define these terms as:

Directed : one-way edges, from a "sender" node to a "receiver" node.
Attribute : properties that can be encoded as a vector, set, or even another graph.
Attributed : edges and vertices have attributes associated with them.
Global attribute : a graph-level attribute.
Multi-graph : there can be more than one edge between vertices, including self-edges.

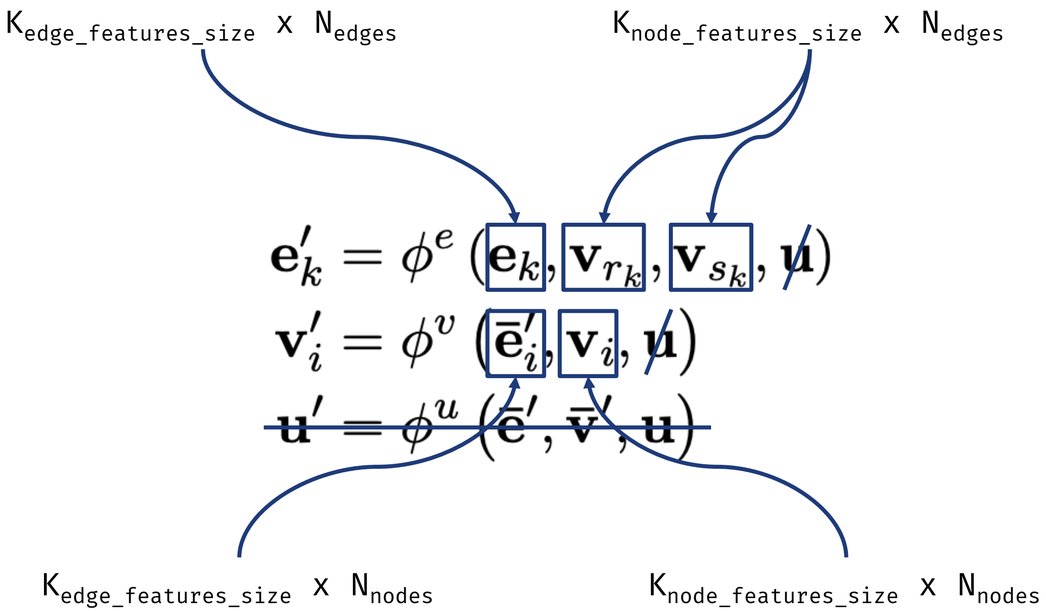
function GRAPHNETWORK(E, V, $\mathbf{u}$)
  for $k \in \{1 \dots N^e\}$ **do**
    $\mathbf{e}'_k \leftarrow \phi^e(\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}, \mathbf{u})$ ▷ 1. Compute updated edge attributes
  end for
  for $i \in \{1 \dots N^n\}$ **do**
    let $E'_i = \{(\mathbf{e}'_k, r_k, s_k)\}_{r_k=i,\ k=1:N^e}$
    $\bar{\mathbf{e}}'_i \leftarrow \rho^{e\to v}(E'_i)$ ▷ 2. Aggregate edge attributes per node
    $\mathbf{v}'_i \leftarrow \phi^v(\bar{\mathbf{e}}'_i, \mathbf{v}_i, \mathbf{u})$ ▷ 3. Compute updated node attributes
  end for
  let $V' = \{\mathbf{v}'\}_{i=1:N^v}$
  let $E' = \{(\mathbf{e}'_k, r_k, s_k)\}_{k=1:N^e}$
  $\bar{\mathbf{e}}' \leftarrow \rho^{e\to u}(E')$ ▷ 4. Aggregate edge attributes globally
  $\bar{\mathbf{v}}' \leftarrow \rho^{v\to u}(V')$ ▷ 5. Aggregate node attributes globally
  $\mathbf{u}' \leftarrow \phi^u(\bar{\mathbf{e}}', \bar{\mathbf{v}}', \mathbf{u})$ ▷ 6. Compute updated global attribute
  return $(E', V', \mathbf{u}')$
end function

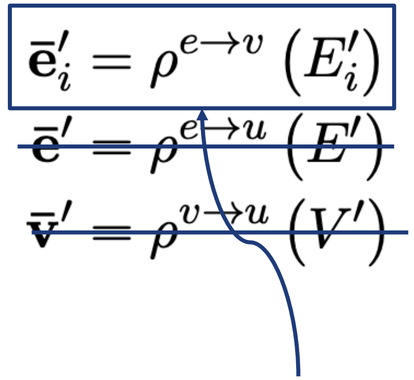
A single GNN can operate on graphs of different sizes (number of edges and nodes) and shapes (edge connectivity), and acts as a learnable graph-to-graph function approximators.

Battaglia, P. W. et al. Relational inductive biases, deep learning, and graph networks. Preprint at https://doi.org/10.48550/arXiv.1806.01261 (2018).

Challenges: input space complexity of message passing

GNNs need to allocate tensors large enough to store nodes and edges attributes.





If the graph is represented using an edges list,
it can be implemented using `jax.ops.segment_sum`

The exact numbers are implementation dependent, but one should reserve at least enough space for allocating a tensor of size:

$$\min((2 \times K_{\text{node\_features\_size}} + K_{\text{edge\_features\_size}}) \times N_{\text{edges}}, (K_{\text{node\_features\_size}} + K_{\text{edge\_features\_size}}) \times N_{\text{nodes}})$$

Challenges: network depth

GPU / TPU memory requirements for training #55

Closed as not planned

ChrisAGBlake opened on Jan 17, 2024

In the paper it states that you're using the TPU v4 chips, which have 32 GB memory accessible per TPU core I believe. When trying to train the high res version (on nvidia GPU currently) I seem to use > 48 GB of VRAM for a batch size of 1 per GPU.

I'm using code in the example notebook with minimal modification.

Are there any other things that need to be done in order to get the memory useage down below 32GB for the high res model? I note that gradient checkpointing and bfloat16 are already setup in the example notebook in this function if I'm understanding it correctly?

```
def construct_wrapped_graphcast(model_config: graphcast.ModelConfig, task_config: graphcast.TaskConfig):
        # Deeper one-step predictor.
        predictor = graphcast.GraphCast(model_config, task_config)

        # Modify inputs/outputs to `graphcast.GraphCast` to handle conversion to
        # from/to float32 to/from BFloat16.
        predictor = casting.Bfloat16Cast(predictor)

        # Modify inputs/outputs to `casting.Bfloat16Cast` so the casting to/from
        # BFloat16 happens after applying normalization to the inputs/targets.
        predictor = normalization.InputsAndResiduals(
            predictor,
            diffs_stddev_by_level=diffs_stddev_by_level,
            mean_by_level=mean_by_level,
            stddev_by_level=stddev_by_level)

        # Wraps everything so the one-step model can produce trajectories.
        predictor = autoregressive.Predictor(predictor, gradient_checkpointing=True)
        return predictor
```

Do the model inputs (xarray datasets) need to also be cast to a different precision instead of float32?

I'm unsure how to get the memory usage down further so that it could be run on the TPU v4 or something like an 40 GB A100.

mjwillson on Jan 25, 2024 Collaborator

Hi Chris, I'm afraid it requires some further tricks not included in this codebase to train a 0.25deg model unrolled to 3 days within TPU RAM limits on TPUv4. The main ones being some additional gradient checkpointing, and offloading some gradient checkpoints from device to host RAM. Our code for this is more tied to internal infrastructure and so the scope for this initial open sourcing was mainly focused on supporting inference, and basic gradient computation for versions of the model that fit more easily into RAM.

1

GraphCast is deep:

1. Embedder = 1 FFN
2. Encoder (1 msg-pass) = 2 FFN
3. Processor (16 msg-pass) = 32 FFN
4. Decoder (1 msg-pass) = 2 FFN

For a total of **37 FFN blocks**[1] (present-day LLM stack order 100 transformer blocks).

The publicly release codebase implements mixed precision, but no rematerialization:
DeepMind GraphCast cannot be trained as is due to memory requirements.

We currently **checkpoint and offload** activations at each message-passing step.

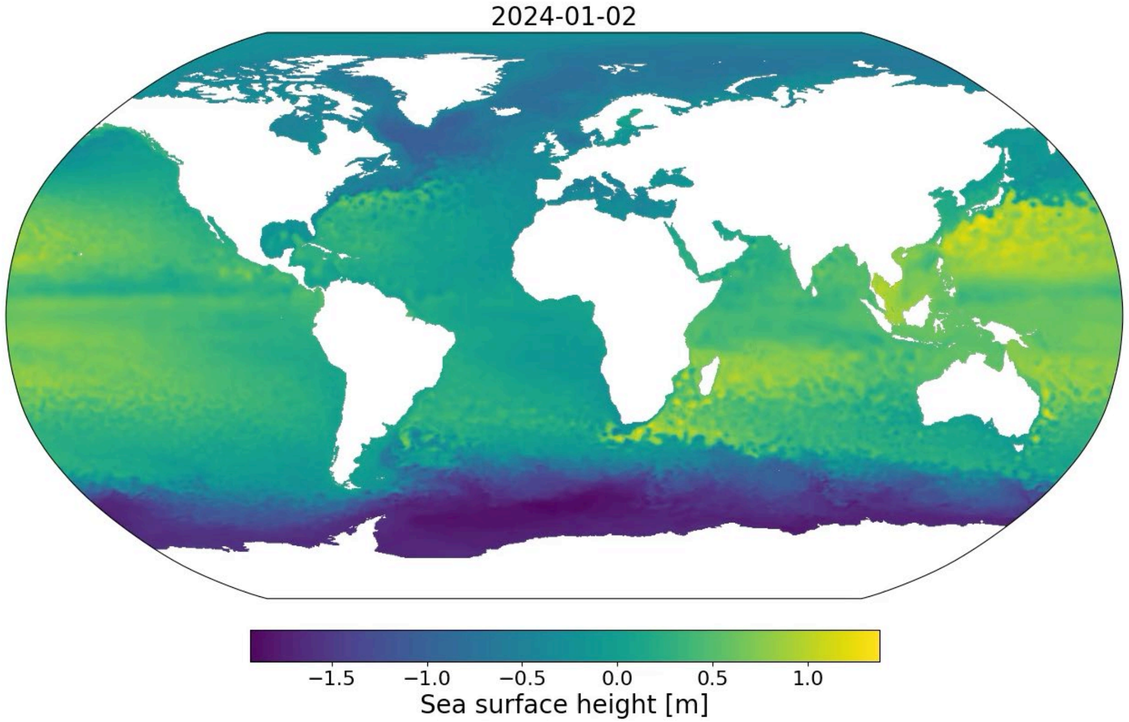
We’re considering task/pipeline parallelism

1. Natural choice for multi-step optimization.
2. For single-step optimization: the encoder/processor/decoder graphs have different sizes and use a different number of message-passing steps. How to split computation?
3. Hard to implement, development friction due to: Haiku (no Flax NNX), JAX API changes.

[1] i.e., MLP + layer normalization

https://github.com/google-deepmind/graphcast/issues/55

Towards a global ocean circulation model based on GraphCast



Our fork of `google-deepmind/graphcast`

1. Functioning environment/setup on Leonardo (fully reproducible)
2. Refactored to work with ARCO-OCEAN variables
3. Masking of land points in predictor, loss, and graphs
4. Scripted global ocean mesh utilities
5. Fully-sharded data-parallelism using JAX manual sharding
6. Prefetching dataloader (with Grain) and async checkpointing (with Orbax)
7. Ported all geolocation utilities to `gdal/proj` GIS libraries.
8. **PoC training: OceanBench target variables only, ≈ 50k optimization steps, no rollouts, 120km mesh size**

OceanBench: prognostic variables

Model	Day 1	Day 3	Day 5	Day 7	Day 10
GLO12	0.545	0.553	0.568	0.591	0.635
GLONET	0.653	0.684	0.745	0.823	0.913
WENHAI	0.637	0.725	0.834	0.956	1.144
XIHE	0.651	**0.658**	**0.693**	**0.690**	**0.792**
GraphCast	**0.612**	0.739	0.805	0.878	1.001

RMSE of surface temperature (49cm) [°C]

Model	Day 1	Day 3	Day 5	Day 7	Day 10
GLO12	0.729	0.728	0.729	0.732	0.737
GLONET	0.784	0.789	0.795	0.801	0.794
WENHAI	1.165	1.154	1.146	1.139	1.132
XIHE	**0.720**	0.731	0.726	0.706	**0.691**
GraphCast	0.762	**0.662**	**0.639**	**0.660**	0.711

RMSE of surface salinity (49cm) [PSU]

Model	Day 1	Day 3	Day 5	Day 7	Day 10
GLO12	0.069	0.070	0.073	0.077	0.082
GLONET	0.075	0.077	0.081	**0.084**	**0.089**
WENHAI	0.118	0.120	0.124	0.126	0.131
XIHE	0.079	0.084	0.085	0.088	0.091
GraphCast	**0.069**	**0.074**	**0.079**	0.085	0.097

RMSE of sea surface height [m]

Data from the OceanBench website: https://oceanbench.lab.dive.edito.eu/

OceanBench: prognostic variables

Model	Day 1	Day 3	Day 5	Day 7	Day 10
GLO12	0.114	0.119	0.126	0.134	0.145
GLONET	0.125	0.126	0.131	0.135	0.144
WENHAI	0.175	0.181	0.186	0.191	0.201
XIHE	0.125	**0.123**	**0.125**	**0.123**	**0.125**
GraphCast	**0.118**	**0.123**	0.130	0.138	0.150

RMSE of surface zonal current (49cm) [m/s]

Model	Day 1	Day 3	Day 5	Day 7	Day 10
GLO12	0.113	0.118	0.124	0.132	0.143
GLONET	0.124	0.124	0.127	0.131	0.138
WENHAI	0.169	0.173	0.174	0.174	0.178
XIHE	0.122	**0.121**	**0.121**	**0.120**	**0.121**
GraphCast	**0.117**	0.122	0.128	0.135	0.144

RMSE of surface meridional current (49cm) [m/s]

Data from the OceanBench website: https://oceanbench.lab.dive.edito.eu/

OceanBench: diagnostics variables and derived quantities

Model	Day 1	Day 3	Day 5	Day 7	Day 10
GLO12	41.511	41.344	41.588	42.001	43.219
GLONET	47.627	51.348	55.069	58.334	61.921
WENHAI	**42.584**	**46.514**	**49.838**	**52.830**	57.110
XIHE	54.538	53.652	53.472	56.484	**51.822**
GraphCast	54.522	60.960	66.263	71.077	77.206

RMSE of mixed layer depth [m]

Model	Day 2	Day 3	Day 5	Day 7	Day 9
GLO12	10.390	20.340	38.788	55.874	72.132
GLONET	**9.379**	18.268	34.954	50.804	65.985
WENHAI	11.972	23.634	45.830	67.024	87.646
XIHE	10.570	20.461	38.173	54.029	68.475
GraphCast	9.497	**18.155**	**34.213**	**49.515**	**64.521**

Average Lagrangian drift deviation [km]

Data from the OceanBench website: https://oceanbench.lab.dive.edito.eu/

OceanBench: diagnostics variables and derived quantities

Model	Day 1	Day 3	Day 5	Day 7	Day 10
GLO12	0.199	0.205	0.208	0.209	0.217
GLONET	0.267	0.332	0.391	0.443	0.526
WENHAI	1.793	1.800	1.808	1.816	1.830
XIHE	2.515	2.476	2.672	2.306	2.493
GraphCast	**0.154**	**0.197**	**0.205**	**0.214**	**0.231**

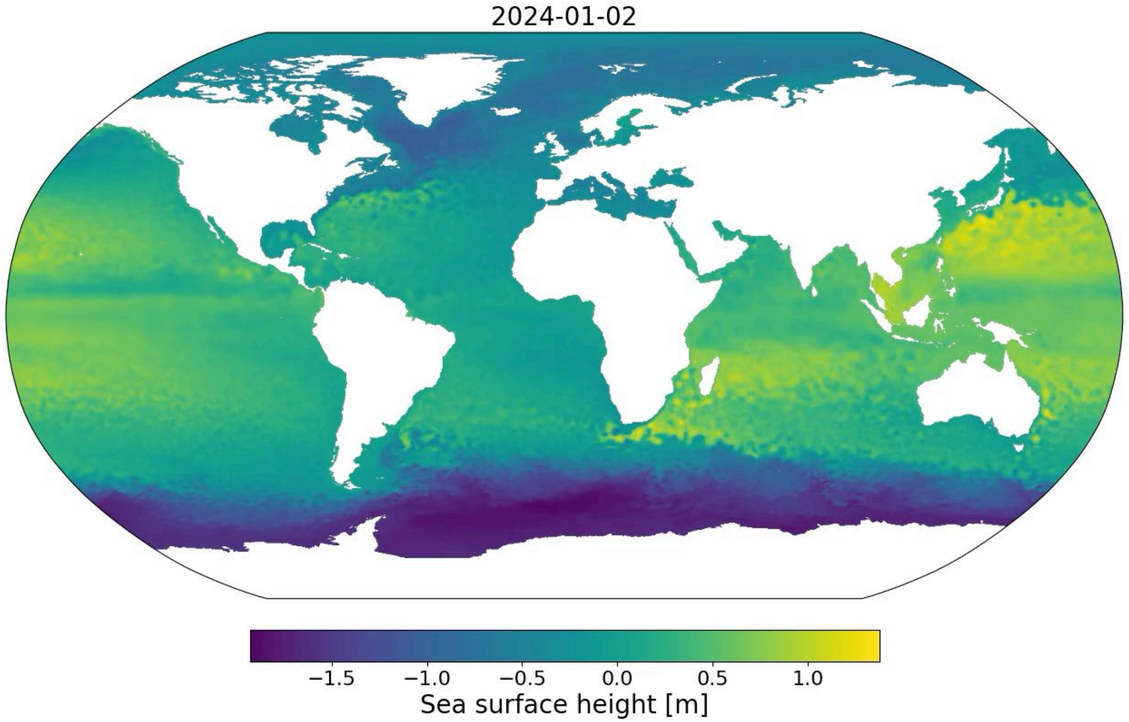
RMSE of zonal geostrophic current [m/s]

Model	Day 1	Day 3	Day 5	Day 7	Day 10
GLO12	0.173	0.177	0.180	0.185	0.189
GLONET	0.274	0.356	0.427	0.490	0.566
WENHAI	2.378	2.379	2.384	2.394	2.414
XIHE	1.993	2.074	2.068	1.990	2.050
GraphCast	**0.150**	**0.201**	**0.211**	**0.221**	**0.240**

RMSE of meridional geostrophic current [m/s]

Data from the OceanBench website: https://oceanbench.lab.dive.edito.eu/

Conclusions and final remarks

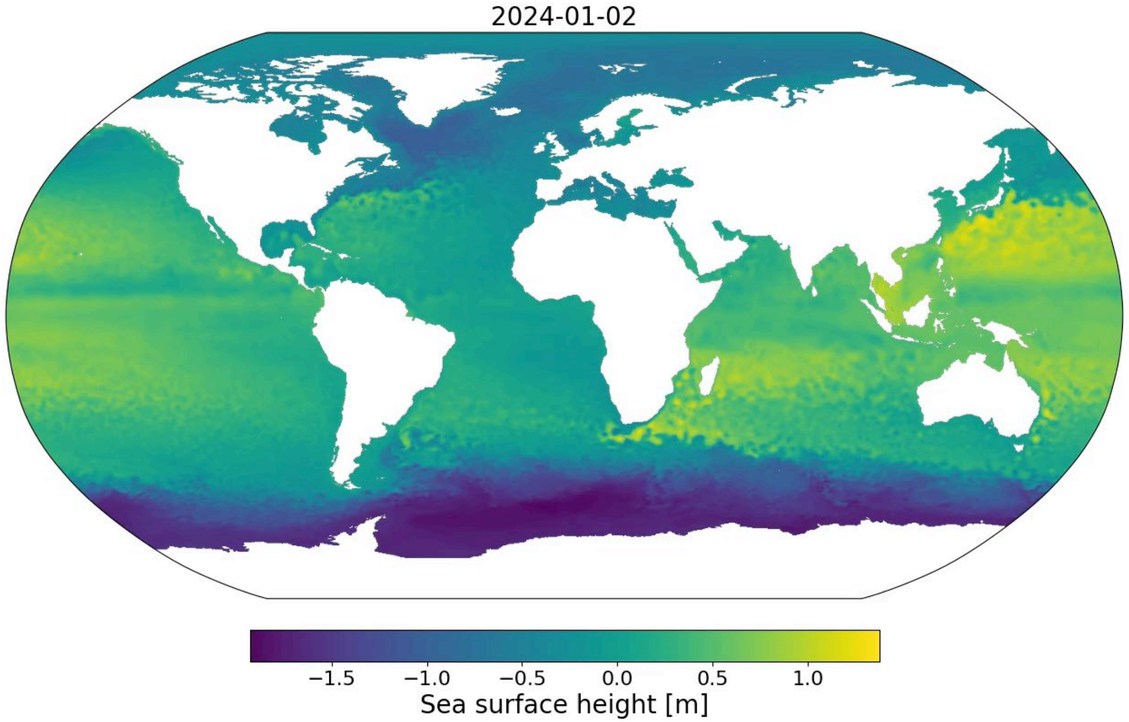


We provided evidence that **GraphCast can serve as a general architecture** for modelling other components of the Earth system.

Its ocean medium-range forecasting skill is already competitive with other machine learning models, though it has not yet reached the level of numerical models.

The engineering and computational resources required to train large GNNs may discourage researchers from experimenting with this approach.

Conclusions and final remarks



Open questions

- How do forecasting skills scale with data availability, computational budget, mesh-graph resolution, and network depth and width?
- How do spatial and temporal resolution affect model stability and divergence from observed trajectories?
- How can the model be coupled with other Earth system components, and how would such coupling affect performance?