Graph Neural Networks

Mihai Alexe

ECMWF Bonn

first.lastname@ecmwf.int

GNNs are **neural networks** built to operate on **graph data**.

arXiv > cs > arXiv:2212.12794

Search...

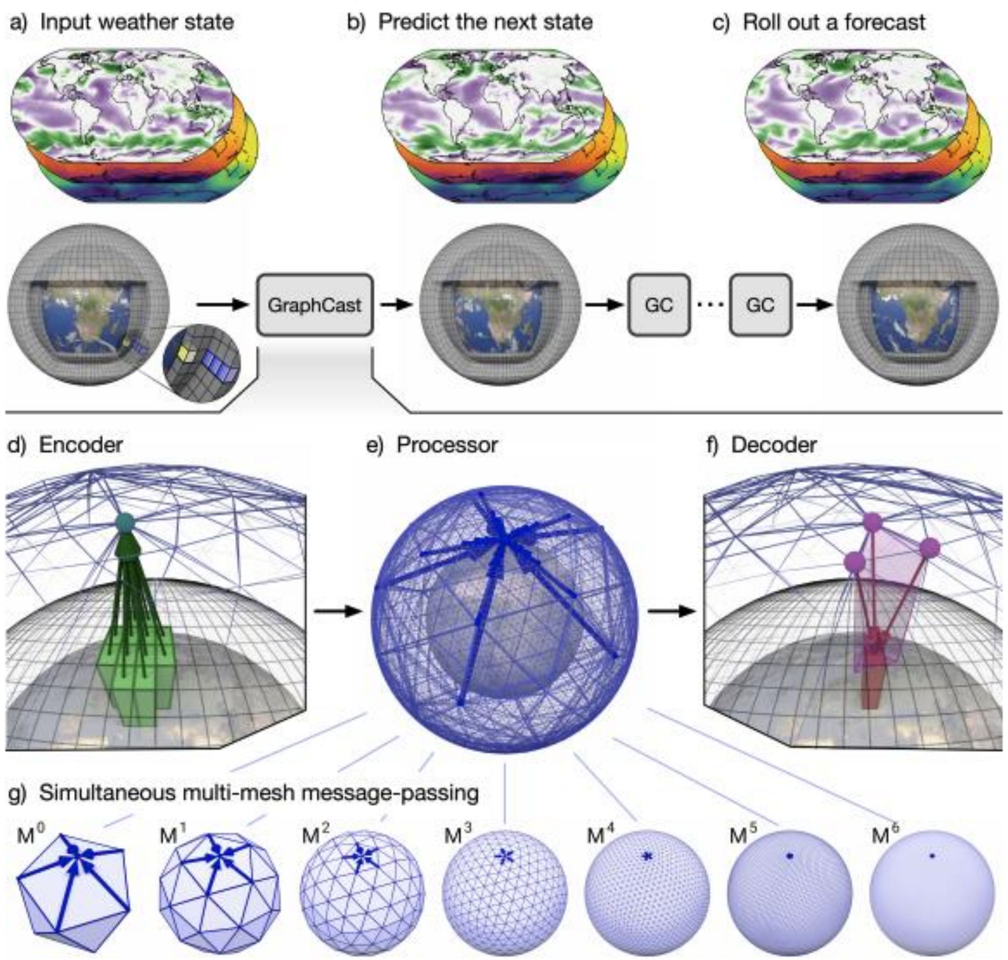
Help | Adva

Computer Science > Machine Learning

[Submitted on 24 Dec 2022 (v1), last revised 4 Aug 2023 (this version, v2)]

GraphCast: Learning skillful medium-range global weather forecasting

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



ECMWF

https://www.science.org/doi/10.1126/science.adi2336

https://arxiv.org/abs/2212.12794

arXiv > physics > arXiv:2406.01465

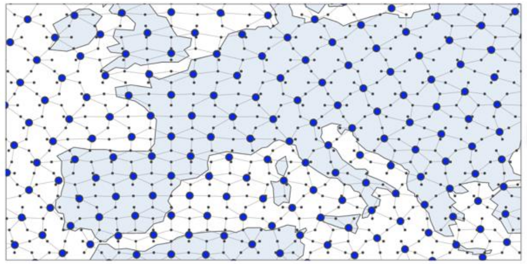
Search
Help

Physics > Atmospheric and Oceanic Physics

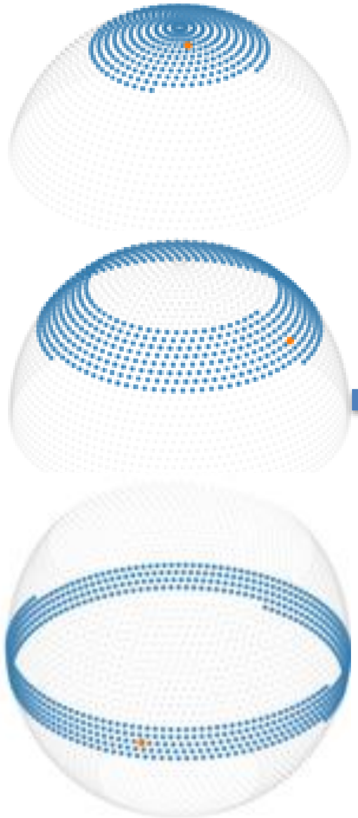
[Submitted on 3 Jun 2024 (v1), last revised 7 Aug 2024 (this version, v2)]

AIFS -- ECMWF's data-driven forecasting system

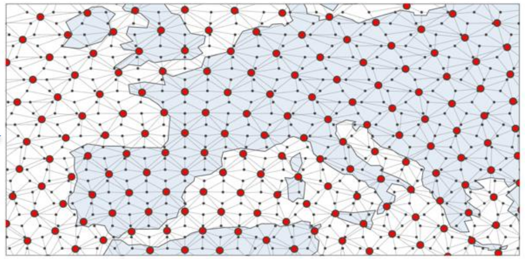
Simon Lang, Mihai Alexe, Matthew Chantry, Jesper Dramsch, Florian Pinault, Baudouin Raoult, Mariana C. A. Clare, Christian Lessig, Michael Maier-Gerber, Linus Magnusson, Zied Ben Bouallègue, Ana Prieto Nemesio, Peter D. Dueben, Andrew Brown, Florian Pappenberger, Florence Rabier



Encoder GNN



Transformer processor



Decoder GNN

ECMWF

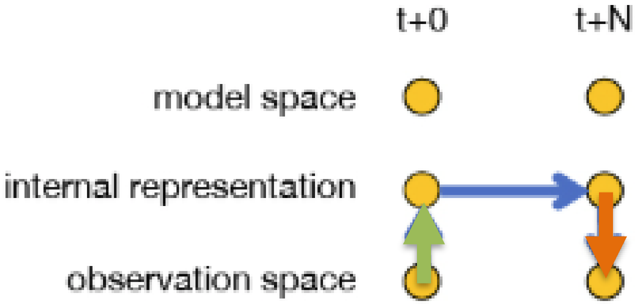
[Submitted on 20 Dec 2024]

GraphDOP: Towards skilful data-driven medium-range weather forecasts learnt and initialised directly from observations

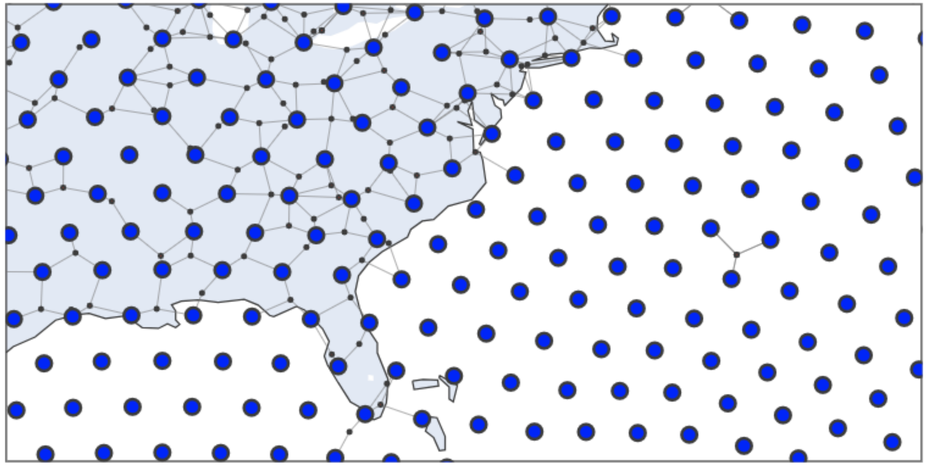
Mihai Alexe, Eulalie Boucher, Peter Lean, Ewan Pinnington, Patrick Laloyaux, Anthony McNally, Simon Lang, Matthew Chantry, Chris Burrows, Marcin Chrust, Florian Pinault, Ethel Villeneuve, Niels Bormann, Sean Healy

5 Predict future observations from observations

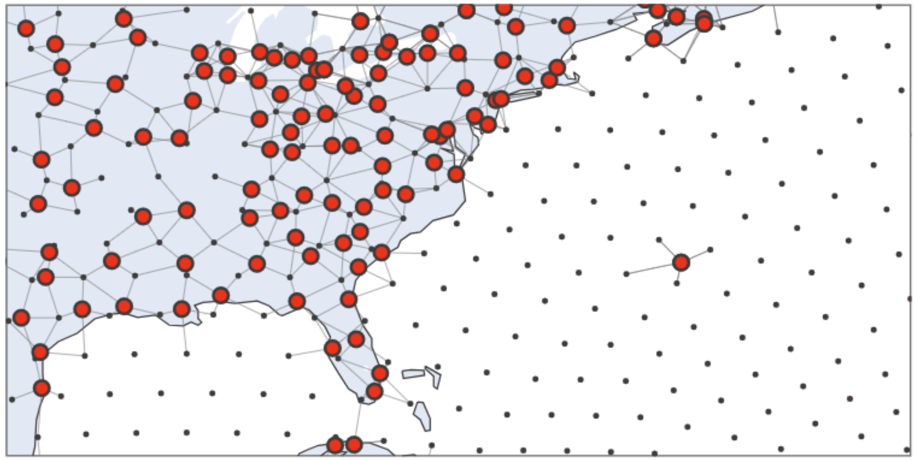
make predictions in observation space, use observations as truth



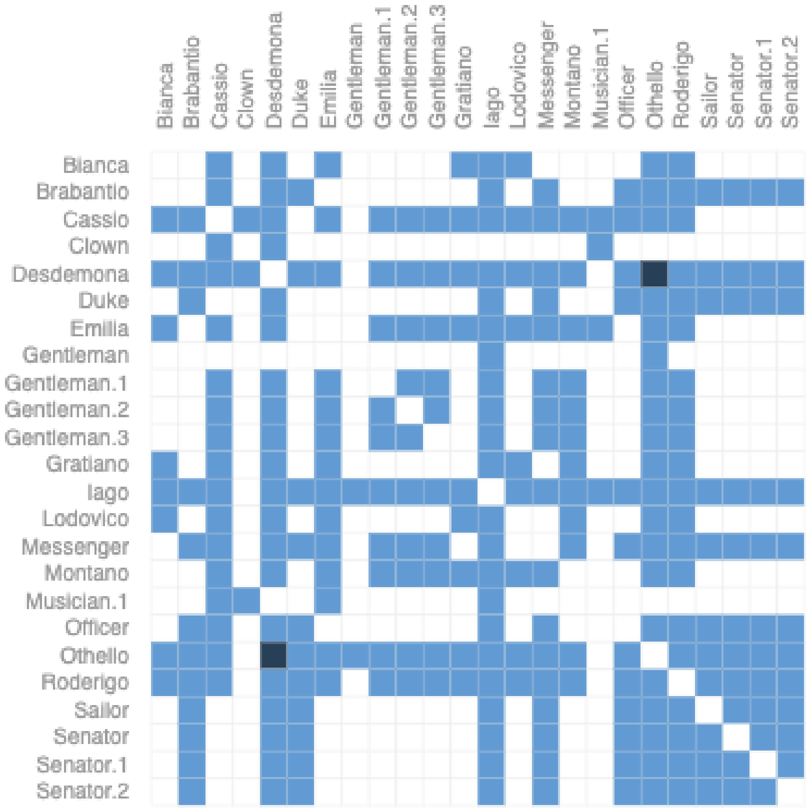
OBS to H graph

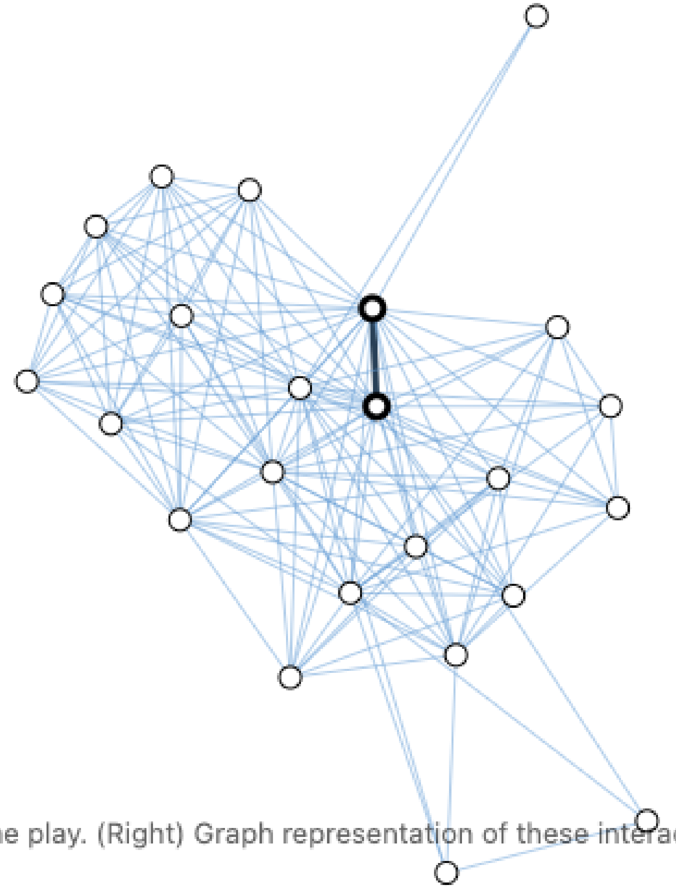


H to OBS graph



Graphs: vertices (nodes), edges (links), connectivity (adjacency) …





(Left) Image of a scene from the play "Othello". (Center) Adjacency matrix of the interaction between characters in the play. (Right) Graph representation of these interactions.

Adjacency matrix **A**

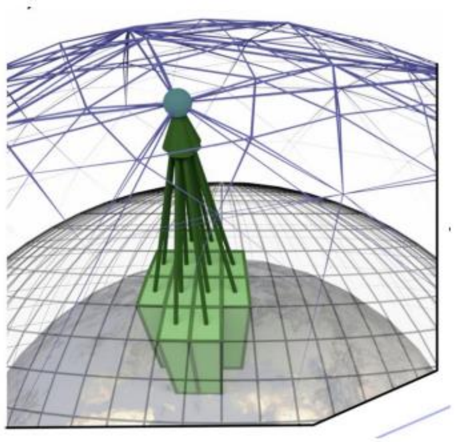
Graph **G = (V, E)**

ECMWF

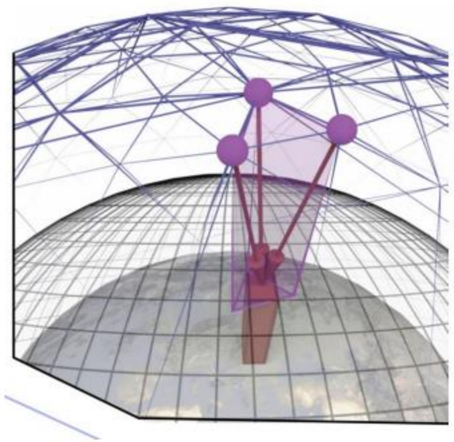
Figure from https://distill.pub/2021/gnn-intro/

Graph structure representation

Bipartite graphs: encode a relation SRC -> DST



ERA5 -> Latent



Latent -> ERA5

Your choice (of representation) matters!

```
src, dst = edge_index
x_src = x[src]  # [E, D]
out = torch.empty((N_DST, D), ...)
for e_idx, d in enumerate(dst):
    out[d] += x_src[e_idx]
```

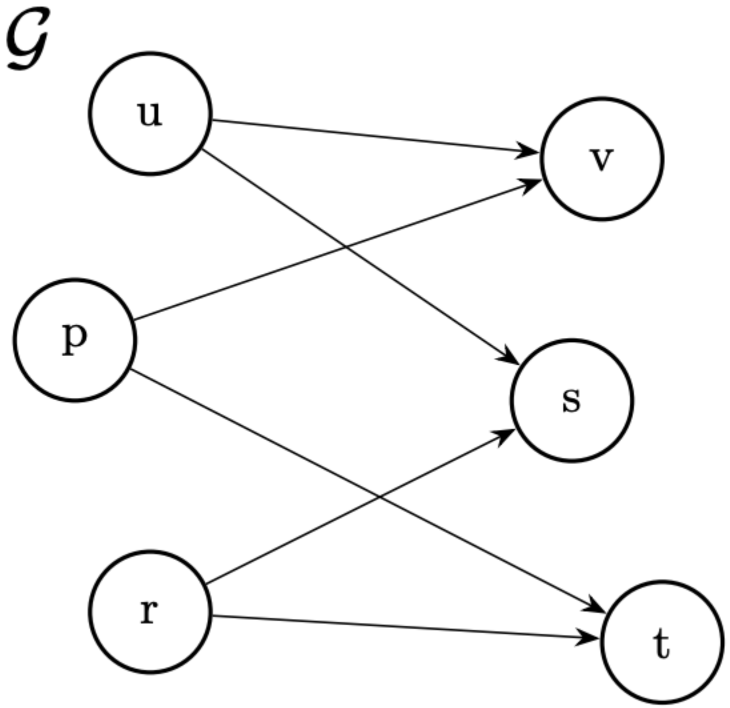
Edge list

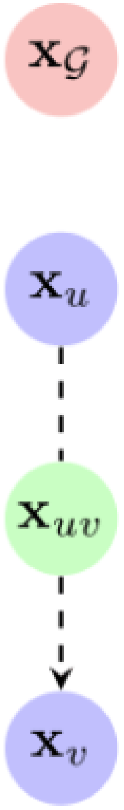
```
out = torch.empty((N_DST, D), ...)
for d in range(N_DST):
    off_start, off_end = offsets[d], offsets[d+1]
    acc = 0.0
    for e_idx in range(off_start, off_end):
        src = indices[e_idx]
        acc += out[src, :]
    out[d, :] = acc
```

Compressed format (CSC)

Pseudocode from https://events.ecmwf.int/event/460/contributions/5223/attachments/3262/5436/HPC-WS_Stadler.pdf

Graph features (= information associated with elements of our graph)





Graph-level (global) features

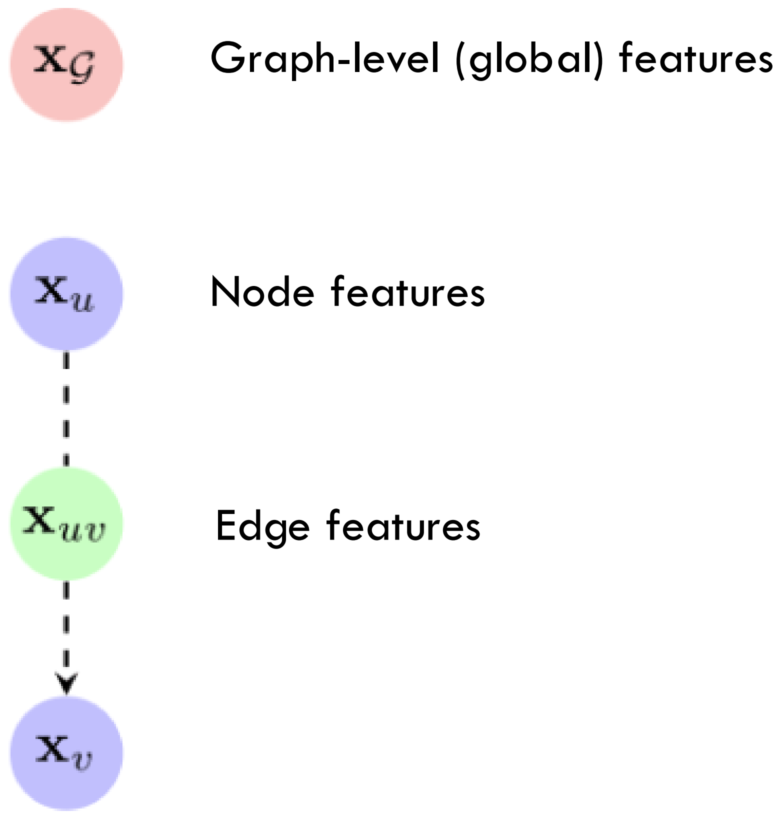
Node features

Edge features

GNN diagrams taken from (Velickovic, 2022): Geometric deep learning - lecture 7: Graphs and sets (II). https://geometricdeeplearning.com/lectures/

Graph neural networks

GNNs are **neural networks** built to operate on **graph data**.



Quick detour: MLPs

Multi-layer perceptrons

```
from torch import nn

def generate_mlp_module(num_inputs: int = 32, hidden_dim: int = 64, num_outputs: int = 32):
    mlp = nn.Sequential(
        nn.Linear(num_inputs, hidden_dim),
        nn.LeakyReLU(0.1),
        nn.Linear(hidden_dim, hidden_dim),
        nn.LeakyReLU(0.1),
        nn.Linear(hidden_dim, num_outputs),
        nn.LeakyReLU(0.1),
        nn.LayerNorm(num_outputs)
    )
    return mlp
```

(…., **???**) = **MLP** (…, **???**)

MLPs will be denoted by Greek letters ϕ, ψ and ρ

Before we mathematically define a GNN layer…

Q: What **inductive biases** should a GNN have?

Locality

We want the GNN signal to be stable under small domain deformations.

Standard deep NNs (e.g., CNNs) build large-scale ops from small-scale building blocks (e.g., 3x3 convolutions).

GNN layers operate locally, too - over *neighborhoods*.

We can extract neighborhood features and define **local functions** ϕ (**MLPs**) operating on them:

$$\mathbf{X}_{\mathcal{N}_i} = \{\{\, \mathbf{x}_j : j \in \mathcal{N}_i \,\}\}$$

$$\phi(\mathbf{x}_i, \mathbf{X}_{\mathcal{N}_i})$$

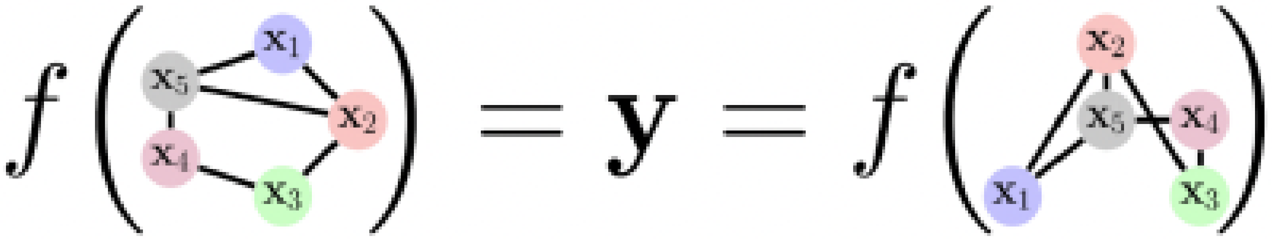
Permutation invariance and equivariance

For certain applications, the specific ordering of nodes and edges should not matter!

Invariance

$$f(PX, PAP^T) = f(X, A)$$

A = adjacency matrix



Examples: **max, sum, min, avg**

$\bigoplus$ = any permutation-invariant aggregation op acting on one or more graph nodes / edges

Permutation equivariance

What if we wanted to distinguish between outputs at different nodes? E.g.: node classification

A permutation-invariant aggregator would not allow us to do that ☹

Instead, we may use functions that **don't change the node ordering.**

That is, if we permute nodes using a permutation matrix P, it doesn't matter if we do it before or after F! ☺

$$\mathbf{F}(\mathbf{X},\mathbf{A}) = \begin{bmatrix} - & \phi(\mathbf{x}_1, \mathbf{X}_{\mathcal{N}_1}) & - \\ - & \phi(\mathbf{x}_2, \mathbf{X}_{\mathcal{N}_2}) & - \\ & \vdots & \\ - & \phi(\mathbf{x}_n, \mathbf{X}_{\mathcal{N}_n}) & - \end{bmatrix}$$

$$F(P\mathbf{X}, PAP^T) = PF(X, A)$$

If ϕ is permutation <u>invariant</u> over the neighborhood , $\mathbf{X}_{\mathcal{N}_u}$ **F** is permutation <u>equivariant</u>!

We stack multiple equivariant GNN layers to build large-scale operators:

$$\mathbf{F}(\mathbf{X}, \mathbf{A}) := \phi\left(\bigoplus_{v \in \mathcal{N}_u} \psi(\mathbf{x}_u, \mathbf{x}_v, \mathbf{x}_{uv})\right)$$

$\bigoplus$ = sum, average … or any *permutation-invariant* aggregation op acting on one or more graph nodes / edges in a neighborhood

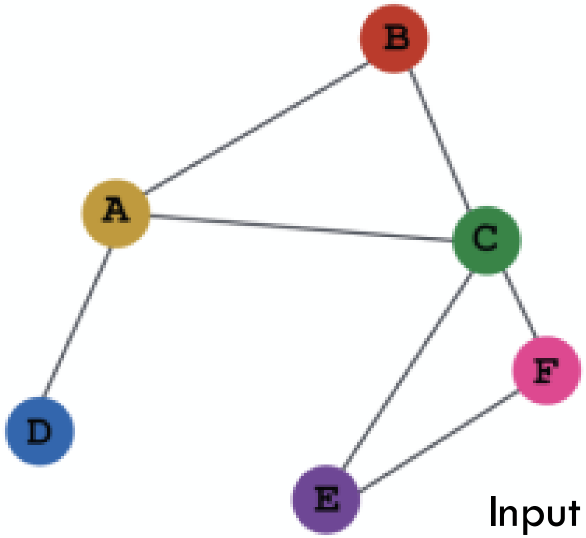
We've just defined a GNN layer!

$$\mathbf{F}(\mathbf{X}, \boldsymbol{A}) := \phi\left(\bigoplus_{v \in \mathcal{N}_u} \psi(\boldsymbol{x}_u, \boldsymbol{x}_v, \boldsymbol{x}_{uv})\right)$$

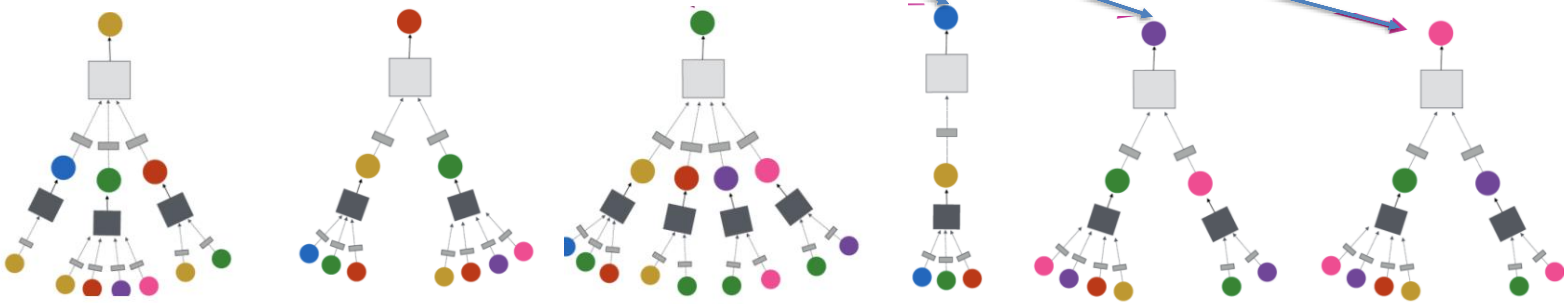
Trainable, shared MLPs

GNN layers are defined by the shared application of local, differentiable and permutation equivariant MLPs

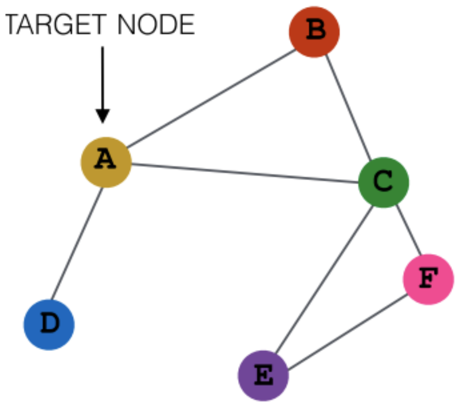
Each node in the graph has its own computational graph, defined by the edge connectivity



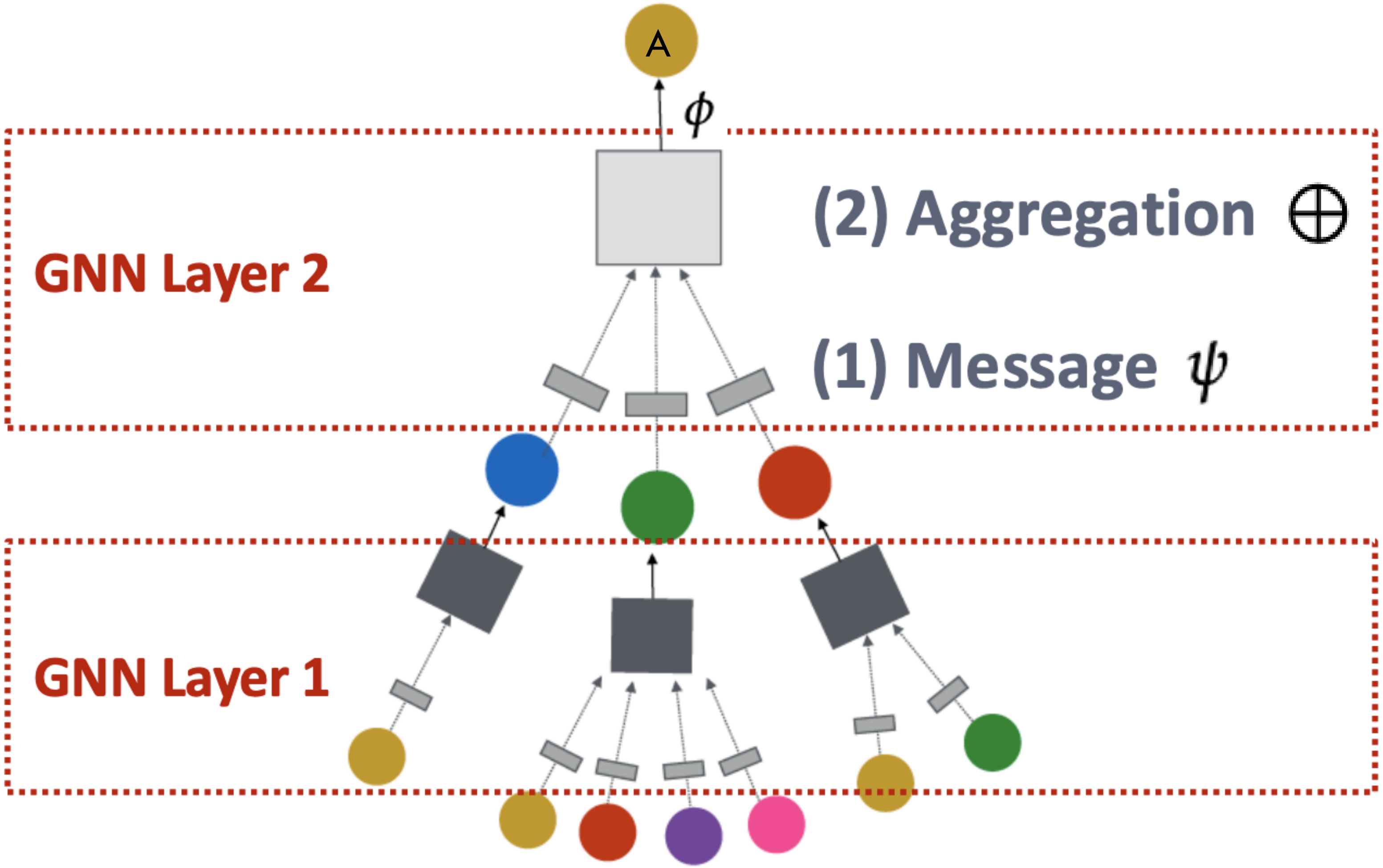
Input graph



Diagrams from Jure Leskovec, Stanford CS224W: Machine Learning with Graphs, http://cs224w.stanford.edu



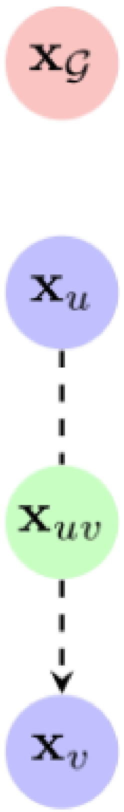
$$\mathbf{F}(\boldsymbol{X}, \boldsymbol{A}) := \phi\left(\bigoplus_{v \in \mathcal{N}_u} \psi(\boldsymbol{x}_u, \boldsymbol{x}_v, \boldsymbol{x}_{uv})\right)$$



Diagrams from Jure Leskovec, Stanford CS224W: Machine Learning with Graphs, http://cs224w.stanford.edu

Quiz time

$$\mathbf{F}(\mathbf{X},\mathbf{A}) := \phi\left(\bigoplus_{v\in\mathcal{N}_u} \psi(\mathbf{x}_u, \mathbf{x}_v, \mathbf{x}_{uv})\right)$$



Graph-level (global) features

Node features

Edge features

How would the **graph-level feature(s)** $\mathbf{x}_\mathcal{G}$ in this framework?

Quiz time

Inductive bias	Task
Locality	**All** (operators act over neighborhoods)
Invariance	**Graph classification** (e.g. “bad” vs “good” protein structure) **Graph regression** (e.g. molecular energy prediction) **Edge (link) prediction**
Equivariance	**Node classification / regression** **3D molecular dynamics** (rotation/translation)

Can we (and should we?) break permutation equivariance?

Quick detour: MLPs

```
from torch import nn

def generate_mlp_module(num_inputs: int = 32, hidden_dim: int = 64, num_outputs: int = 32):
    mlp = nn.Sequential(
        nn.Linear(num_inputs, hidden_dim),
        nn.LeakyReLU(0.1),
        nn.Linear(hidden_dim, hidden_dim),
        nn.LeakyReLU(0.1),
        nn.Linear(hidden_dim, num_outputs),
        nn.LeakyReLU(0.1),
        nn.LayerNorm(num_outputs)
    )
    return mlp
```

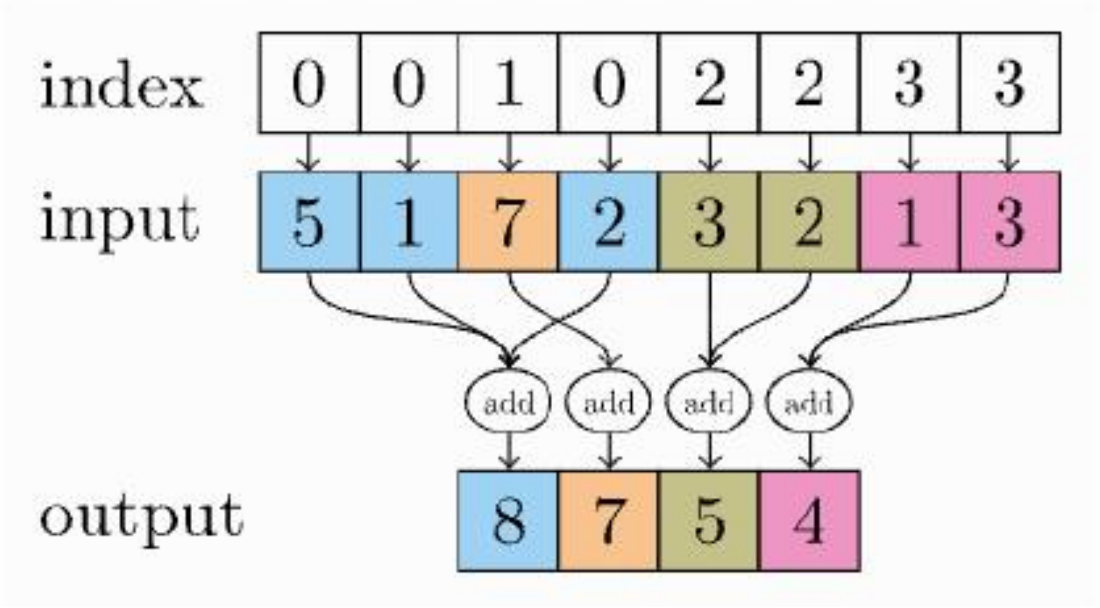
(…, num_neighbors, **num_outputs**) = **MLP** (…, num_neighbors, **num_inputs**)

Recall: MLPs (1) act on neighborhoods and (2) share the weights.

Q: what does this imply wrt **num_neighbors**?

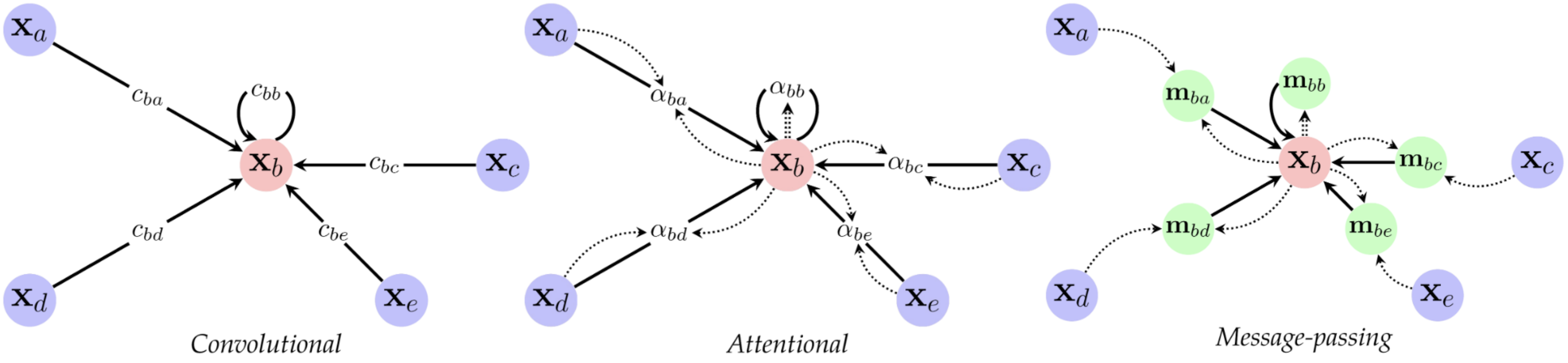
Two basic ingredients of GNNs

- **Shared MLPs** ϕ, ψ and ρ operate on node and edge features (we'll see how)
- **Sparse index-gather / -scatter operations** over graph neighborhoods (GPU-optimized)



Flavors of GNNs

$$h_i = \phi\Big(x_i, \bigoplus_{j \in \mathcal{N}_i} \alpha(x_i, x_j)\psi(x_j)\Big)$$



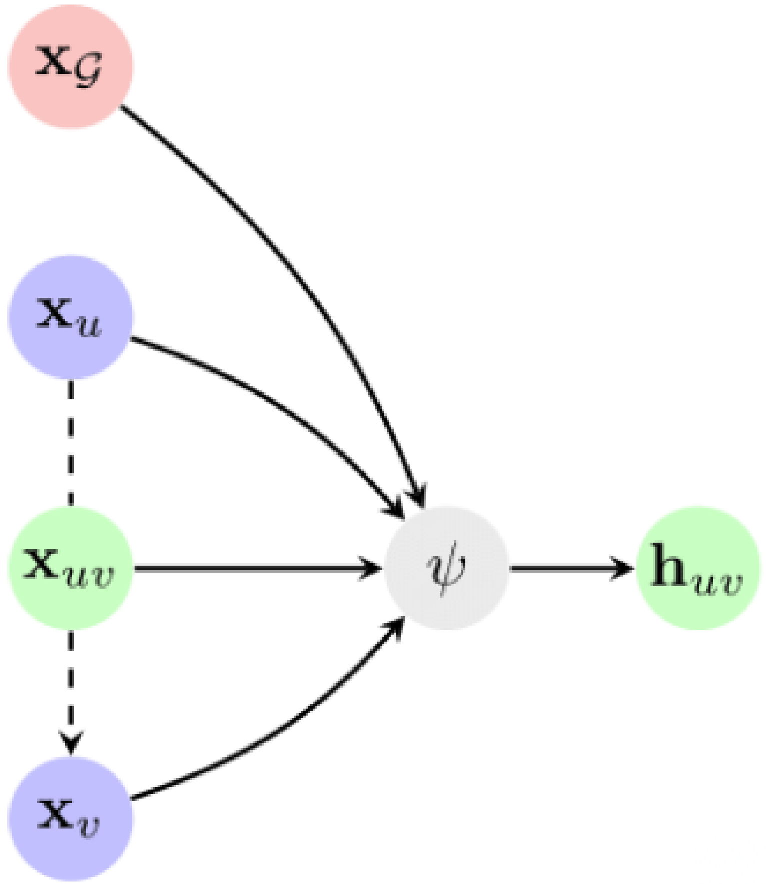
$$h_i = \phi\Big(x_i, \bigoplus_{j \in \mathcal{N}_i} c_{ij}\psi(x_j)\Big)$$

$$h_i = \phi\Big(x_i, \bigoplus_{j \in \mathcal{N}_i} \psi(x_i, x_j, e_{ij})\Big)$$

$$\mathbf{m}_{ij} := \psi(x_i, x_j, e_{ij}).$$

Message-passing GNNs

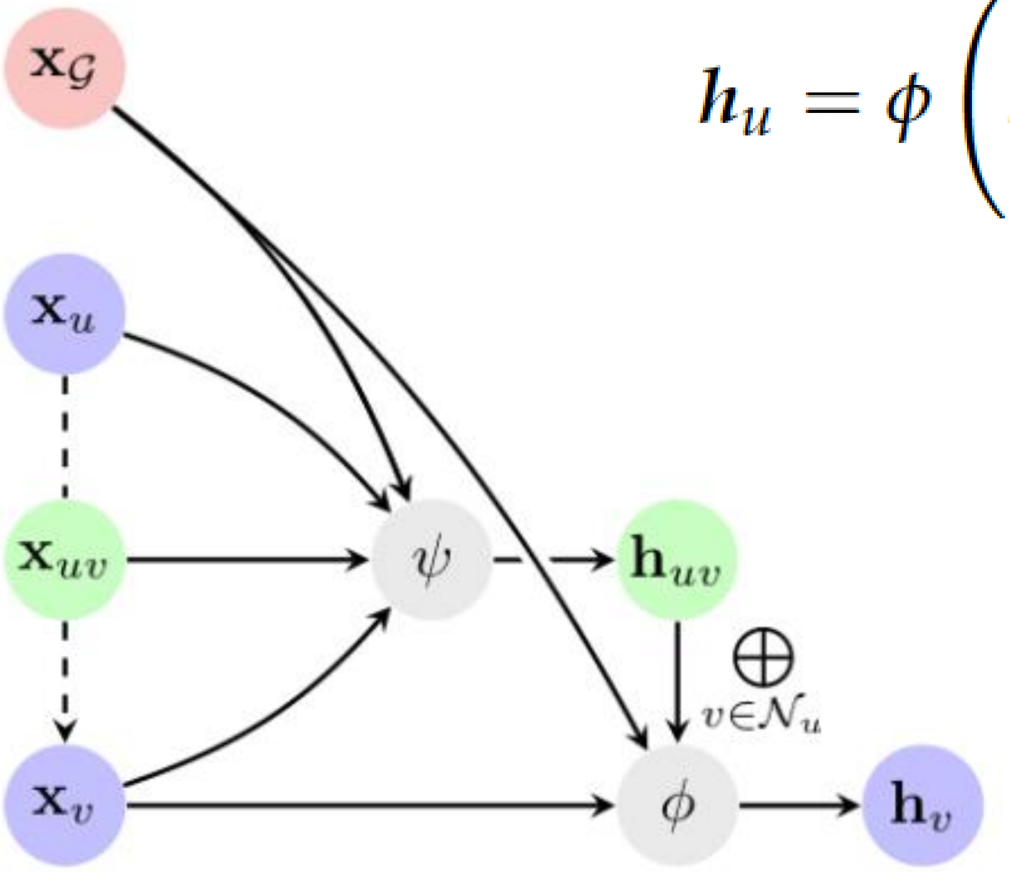
Step 1: Edge updates



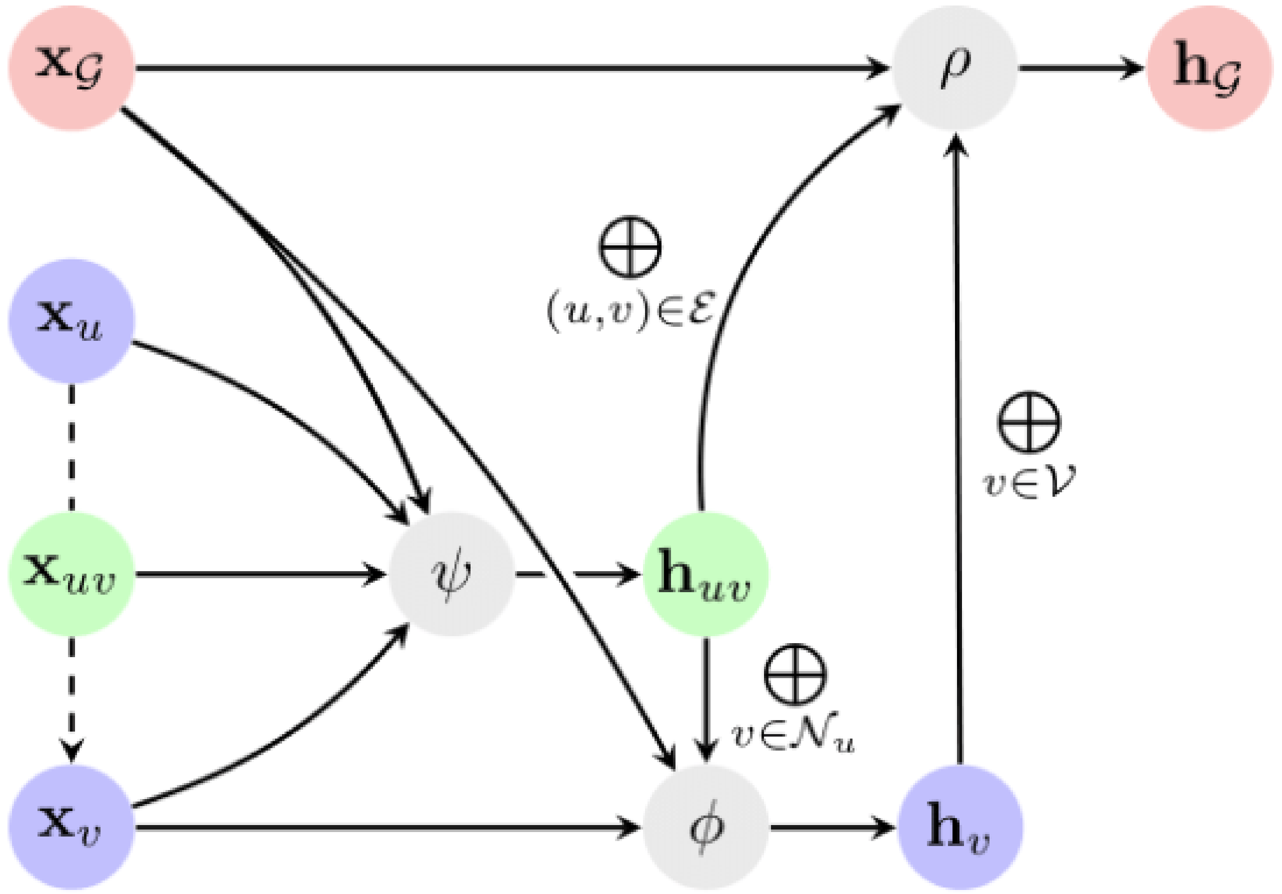
$$h_{uv} = \psi(x_u, x_v, x_{uv}, x_{\mathcal{G}})$$

Step 2: Node updates

$$h_u = \phi\left(x_u, \bigoplus_{u \in \mathcal{N}_v} h_{vu}, x_{\mathcal{G}}\right)$$



Step 3: Graph feature updates

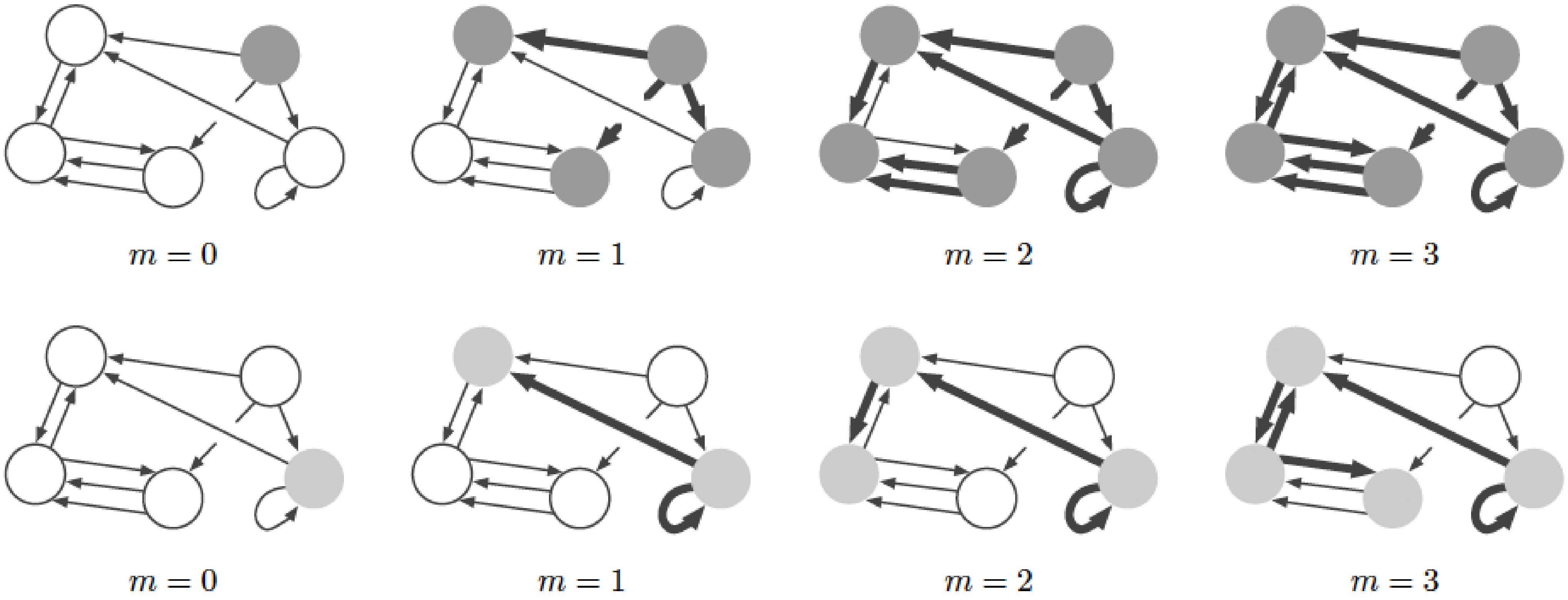


$$h_\mathcal{G} = \rho\left(\bigoplus_{u\in\mathcal{V}} h_u, \bigoplus_{(u,v)\in\mathcal{E}} h_{uv}, x_\mathcal{G}\right)$$

The message-passing algorithm

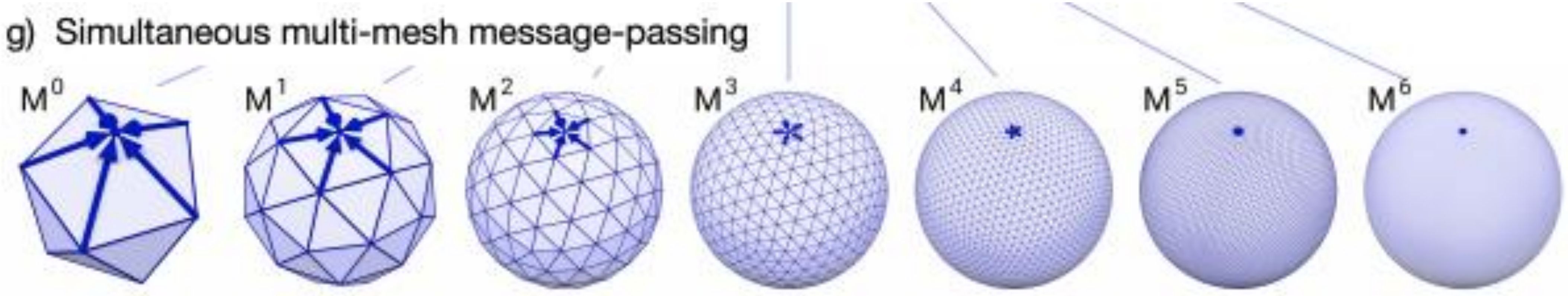
Input: Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ with $\{\boldsymbol{x}_{\mathcal{G}}, \boldsymbol{h}_u, \boldsymbol{h}_{uv}\}$.
for each edge e_{uv} **do**
    Gather sender and receiver nodes $\boldsymbol{x}_u$, $\boldsymbol{x}_v$
    Update edge $\boldsymbol{h}_{uv} \leftarrow \psi(\boldsymbol{x}_u, \boldsymbol{x}_v, \boldsymbol{x}_{uv}, \boldsymbol{x}_{\mathcal{G}})$
end for
for each node u **do**
    Aggregate all incoming edges to u: $\boldsymbol{h}_u^* := \bigoplus_{v,(v,u)\in\mathcal{E}} \boldsymbol{h}_{vu}$
    Compute node-wise features $\boldsymbol{h}_u \leftarrow \phi\left(\boldsymbol{x}_u, \boldsymbol{h}_u^*, \boldsymbol{x}_{\mathcal{G}}\right)$
end for
Aggregate all edges and nodes $\boldsymbol{u}^* := \bigoplus_{u\in\mathcal{V}} \boldsymbol{h}_u$, $\mathbf{e}^* := \bigoplus_{(u,v)\in\mathcal{E}} \boldsymbol{h}_{uv}$
Compute global features $\boldsymbol{h}_{\mathcal{G}} \leftarrow \rho(\boldsymbol{x}_{\mathcal{G}}, \boldsymbol{u}^*, \mathbf{e}^*)$
Output: Graph $\mathcal{G}$ with new $\{\boldsymbol{x}_{\mathcal{G}}, \boldsymbol{h}_u, \boldsymbol{h}_{uv}\}$.

Message passing: information propagation



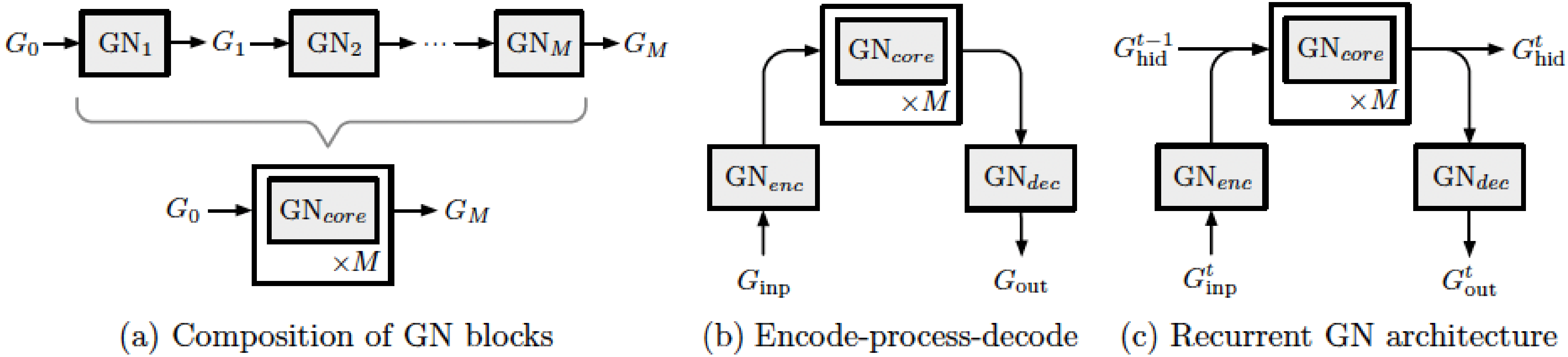
NB: This happens **simultaneously** for all nodes in the graph!

Figure from ([Battaglia et al., 2018](#))



The multi-mesh allows information to propagate faster, across longer distances

GNN block structures



GraphCast, AIFS and GraphDOP use both (a) and (b)

Figures from ([Battaglia et al., 2018](#))

Software

PyG

https://github.com/pyg-team/pytorch_geometric

anemoi graphs

https://github.com/ecmwf/anemoi-core/tree/main/graphs

anemoi graphs

https://anemoi.readthedocs.io/projects/graphs/en/latest/

https://github.com/ecmwf/anemoi-core/tree/main/graphs

- Create a new graph:

```
>>> anemoi-graphs create recipe.yaml graph.pt
```

- Describe an existing graph:

```
>>> anemoi-graphs describe graph.pt
```

- Inspect visually an existing graph:

```
>>> anemoi-graphs inspect graph.pt graph_viz/
```

```
graph_viz
  data_to_hidden.html
  distribution_edge_attributes.png
  distribution_node_adjancency.png
  distribution_node_attributes.png
  hidden_to_data.html
  hidden_to_hidden.html
  isolated_nodes.html
```

Local files generated to inspect graphs.

```
Path          : graph.pt
Format version: 0.0.1

Size        : 3.1 MiB (3,283,650)
```

Nodes name	Num. nodes	Attribute dim	Min. latitude	Max. latitude	Min. longitude
data	10,840	4	-3.135	3.140	0.02
hidden	6,200	4	-3.141	3.137	0.01

Source	Destination	Num. edges	Attribute dim	Min. length	Max. length	Mean
data	hidden	13508	1	0.3116	25.79	11
hidden	data	40910	1	0.2397	21.851	12

```
Graph ready, last update 17 seconds ago.
Statistics ready.
```

Console log when describing/inspecting a graph with anemoi-graphs.

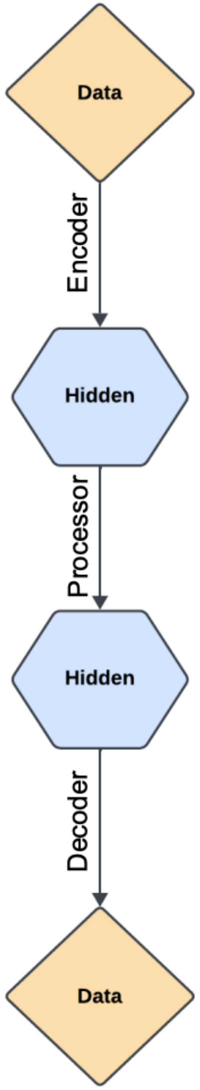
Note: The inspection tools provided are designed for testing different graph configuration but it is not recommendded for high-resolution graphs with a high number of nodes/edges.

anemoi graphs

Graph recipe

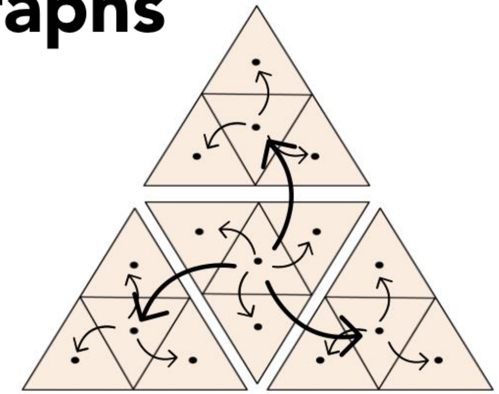
recipe.yaml

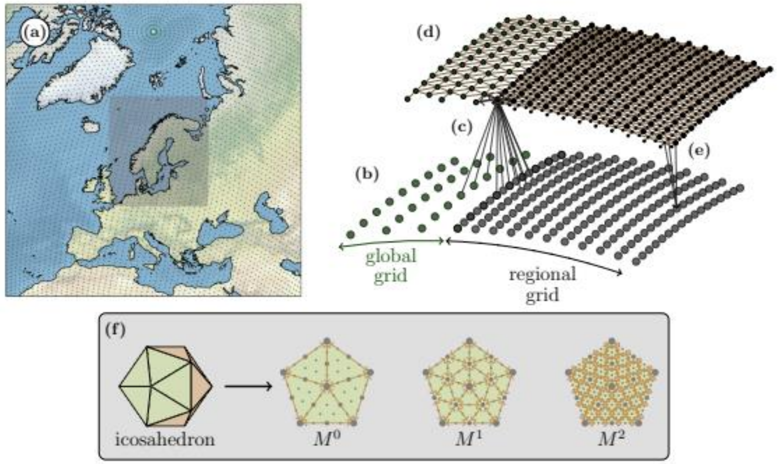
```yaml
nodes:
  data:
    node_builder:
      _target_: anemoi.graphs.nodes.ZarrDatasetNodes
      dataset: my_zarr_dataset
  hidden:
    node_builder:
      _target_: anemoi.graphs.nodes.TriNodes
      resolution: 5 # num of refinements

edges:
  # Encoder configuration
  - source_name: data
    target_name: hidden
    edge_builders:
     - _target_: anemoi.graphs.edges.CutOffEdges
       cutoff_factor: 0.6
  # Processor configuration
  - source_name: hidden
    target_name: hidden
    edge_builders:
     - _target_: anemoi.graphs.edges.MultiScaleEdges
       x_hops: 1
  # Decoder configuration
  - source_name: hidden
    target_name: data
    edge_builders:
     - _target_: anemoi.graphs.edges.KNNEdges
       num_nearest_neighbours: 3
```

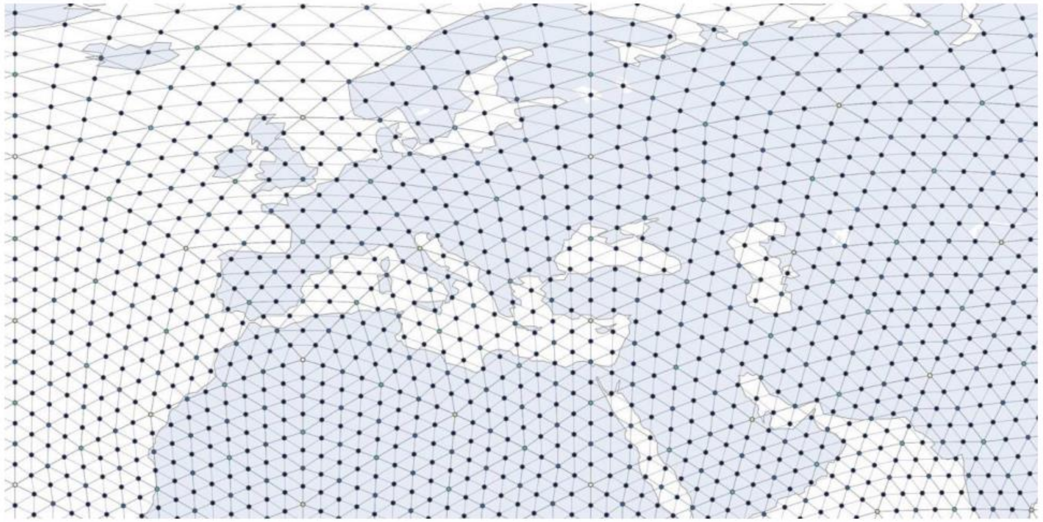


https://events.ecmwf.int/event/446/contributions/4905/attachments/2814/4766/3_Graphs.pdf

anemoi
graphs







Multi-scale graph edges

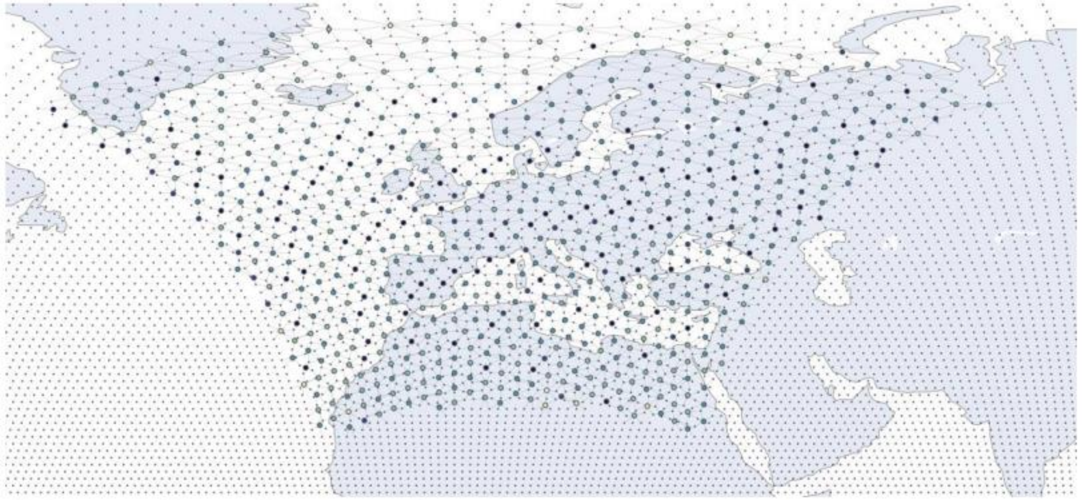


Diagram of encoder connections.

Regional or limited-area modeling

https://arxiv.org/abs/2409.02891
https://arxiv.org/abs/2507.18378

Further references

(Veličković, 2023) https://arxiv.org/pdf/2301.08210.pdf

(Keisler, 2022) https://arxiv.org/abs/2202.07575

(Lam et al., 2023) https://arxiv.org/abs/2212.12794

(Sanchez-Lengeling et al., 2021) https://distill.pub/2021/gnn-intro/

(Veličković, 2023) https://geometricdeeplearning.com/lectures/

(Battaglia et al., 2018) https://arxiv.org/abs/1806.01261

(Sanchez-Gonzalez et al., 2020) https://arxiv.org/abs/2002.09405

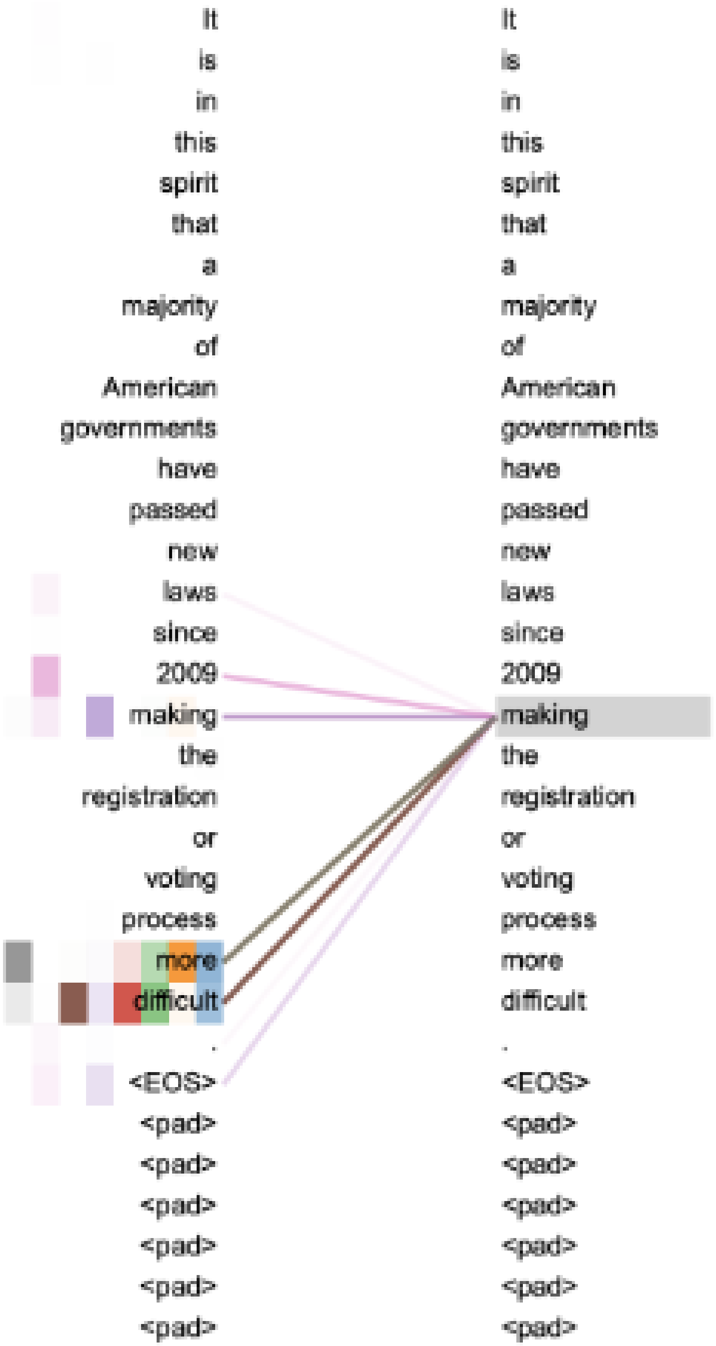
Transformers are fully connected attentional GNNs (+ a positional embedding)

$$A = \mathbb{1}\mathbb{1}^T$$

$$\mathcal{N}_u = \mathcal{V}$$

$$\mathbf{h}_u = \phi\left(\mathbf{x}_u, \bigoplus_{v \in \mathcal{V}} \alpha(\mathbf{x}_u, \mathbf{x}_v)\psi(\mathbf{x}_v)\right)$$

Attention transformers learn a “soft adjacency”



Chaitanya Joshi. Transformers are graph neural networks. The Gradient, 2020.