Graph-Based Weight Cascading Methods for Multisite Time Series Forecasting

Journées de Statistique 2025
Marseille

Eloi Campagne[1,2] Yvenn Amara-Ouali[2] Yannig Goude[2] Mathilde Mougeot[1] Argyris Kalogeratos[1]
eloi.campagne@ens-paris-saclay.fr
05/06/2025

Motivations

Industrial context

Anticipation of the consumption of **electricity** and **renewable energy production** is a major challenge for EDF, especially for electricity market operations:

- ▷ Maintaining a **balance between electricity supply and demand** is important for grid stability;
- ▷ Optimizing the **production fleet and demand response**;
- ▷ **Buying** and **selling** on electricity markets

New **geolocated data** can be exploited by spatial models — such as **GNNs** — and improve forecasts. (Obst, Vilmarest, and Goude 2020; Vilmarest and Goude 2021)

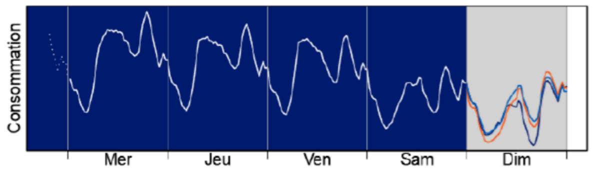


Figure 1 – Electricity consumption.

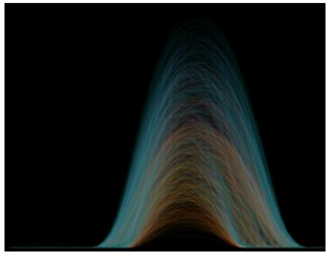


Figure 2 – Renewable energy production.

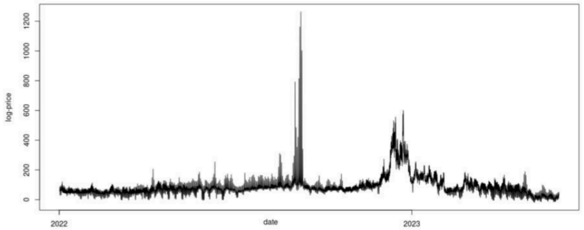


Figure 3 – Electricity prices.

Motivations

Academic context

- ▷ You have a **large dataset** of N time series with a **limited number of observations** T;
- ▷ You want to have N accurate forecasts but you **do not have a huge budget**.

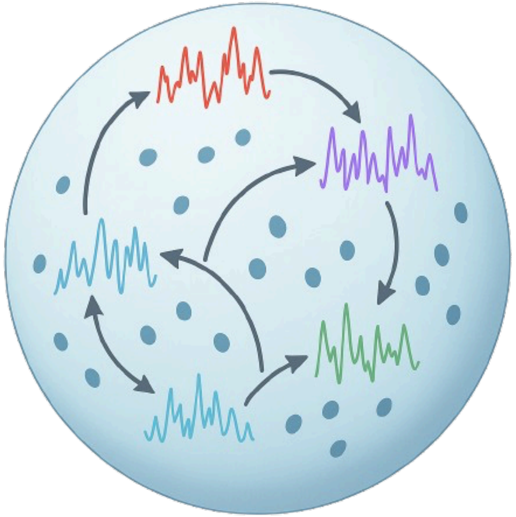


Figure 4 – Timeseries may be hard to order.

Motivations

Approaches

We have tested three different approaches:

Approach 1 – Individual: Train N individual models.

Approach 2 – Cascade: Transfer models weights through a tree structure.

Approach 3 – GNNs: Train a single Graph Neural Network.

About the dataset

- The dataset[1] consists of aggregated **half-hourly residential smart meter electricity consumption data** collected by four UK Distribution Network Operators (DNOs);
- $120,000$ **low voltage feeders**;
  ↳ very **heterogeneous** data;
- Dataset spans **January 2024**;
- Focus on **Oxford**'s urban area.

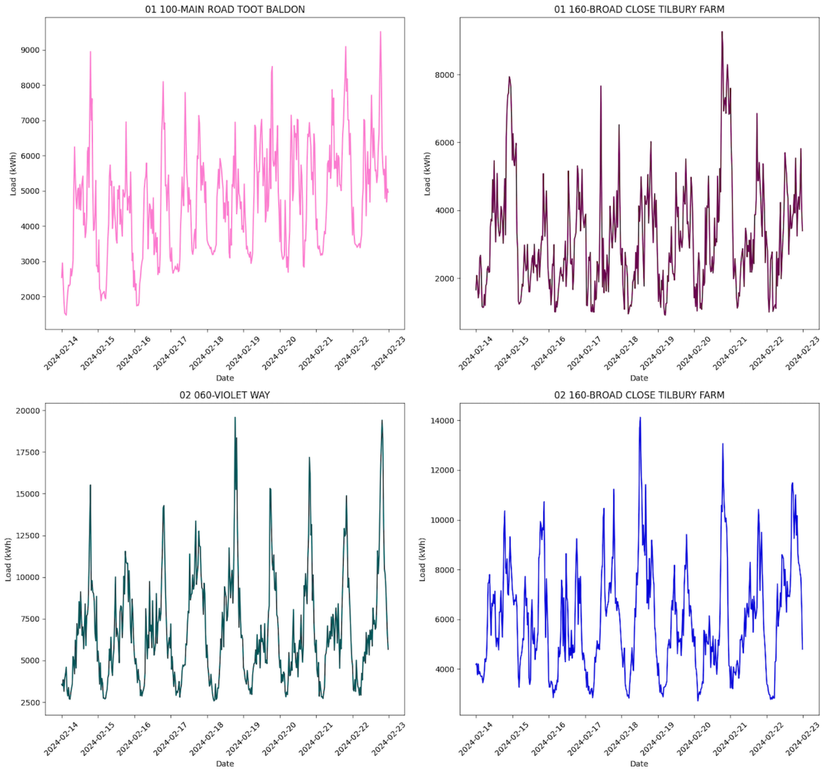


Figure 5 – Subset of 4 nodes of Oxford's urban area.

[1] https://weave.energy/

About the models

Feedforward Neural Networks & Graph Neural Networks

Feedforward Neural Networks

- The general update rule of a hidden vector is given by:

$$\boldsymbol{h}^{(\ell+1)} = \sigma\big(\boldsymbol{W}^{(\ell+1)}\boldsymbol{h}^{(\ell)} + \boldsymbol{b}^{(\ell+1)}\big)$$

where:
↳ $\boldsymbol{W}^{(\ell+1)} \in \mathbb{R}^{d_{\ell+1} \times d_\ell}$ is a learned **weight** matrix;
↳ $\boldsymbol{b}^{(\ell+1)} \in \mathbb{R}^{d_{\ell+1}}$ is a learned **bias** vector;
↳ σ is a non-linear **activation function** (e.g. ReLU, tanh).

Graph Neural Networks (Gori and Monfardini 2005)

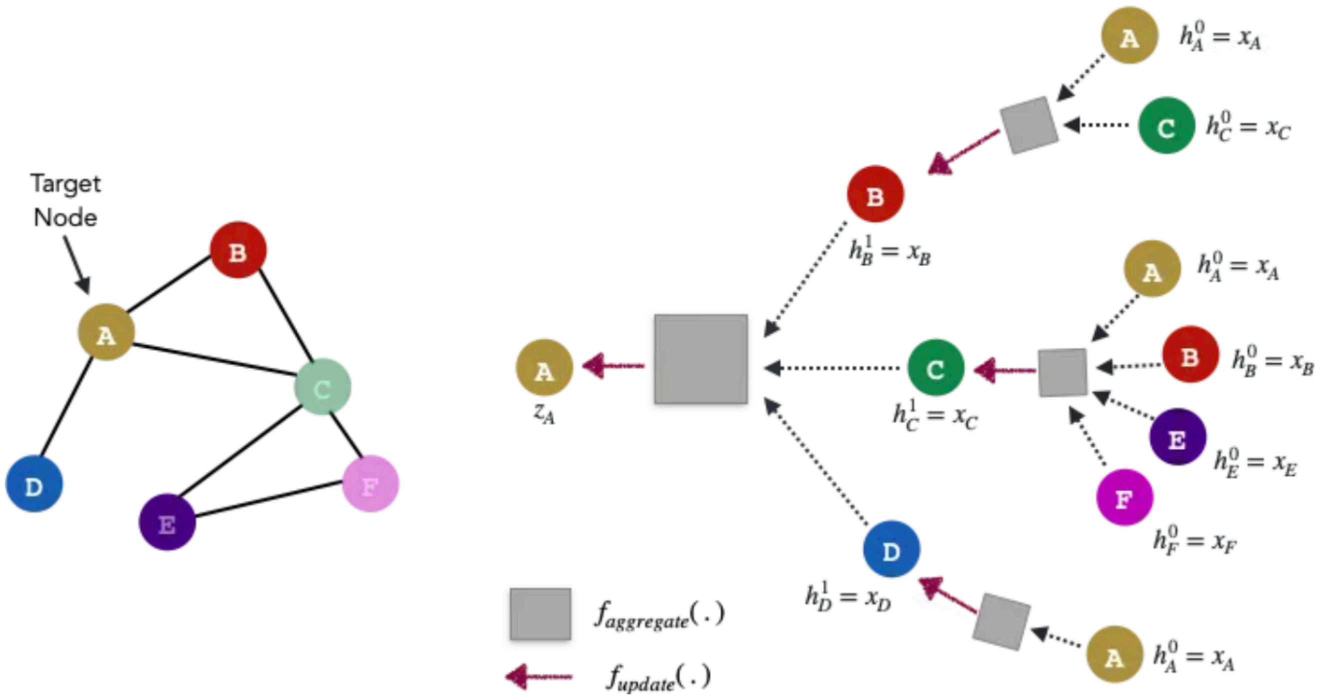
- The general update rule for a node u is given by:

$$\boldsymbol{h}_u^{(\ell+1)} = \phi\Big(\boldsymbol{h}_u^{(\ell)}, \bigoplus_{v \in \mathcal{N}_u} \psi\Big(\boldsymbol{h}_u^{(\ell)}, \boldsymbol{h}_v^{(\ell)}, \boldsymbol{e}_{uv}\Big)\Big)$$

where:
↳ $\mathcal{N}_u$ is the set of neighbors of u;
↳ ϕ and ψ are respectively **update** and **message** functions;
↳ $\bigoplus$ is the **aggregation** operator;
↳ $\boldsymbol{e}_{uv}$ is the edge representation between u and v.

About the models

Visual understanding of a GNN



Two examples of graph convolutions

Graph Convolutional Networks & Graph Attention Networks

Graph Convolutional Networks

(Kipf and Welling 2016)

- ▷ The update rule for a node u is given by:

$$\boldsymbol{h}_u^{(\ell+1)} = \sigma\left(\sum_{v\in\mathcal{N}_u\cup\{u\}} c_{vu} \boldsymbol{\Theta}^{(\ell)} \boldsymbol{h}_v^{(\ell)}\right)$$

where

$$c_{vu} = \frac{e_{vu}}{\sqrt{d_v d_u}}.$$

Graph Attention Networks

(Veličković et al. 2017; Brody, Alon, and Yahav 2022)

- ▷ The update rule for a node u is given by:

$$\boldsymbol{h}_u^{(\ell+1)} = \sigma\left(\sum_{v\in\mathcal{N}_u\cup\{u\}} \alpha_{uv} \boldsymbol{\Theta}_t^{(\ell)} \boldsymbol{h}_v^{(\ell)}\right)$$

where

$$\alpha_{uv} = \frac{\exp\left(\boldsymbol{a}^\top \sigma\left(\boldsymbol{\Theta}_s^{(\ell)} \boldsymbol{h}_u^{(\ell)} + \boldsymbol{\Theta}_t^{(\ell)} \boldsymbol{h}_v^{(\ell)} + \boldsymbol{\Theta}_e^{(\ell)} e_{uv}\right)\right)}{\sum_{k\in\mathcal{N}_u\cup\{u\}} \exp\left(\boldsymbol{a}^\top \sigma\left(\boldsymbol{\Theta}_s^{(\ell)} \boldsymbol{h}_u^{(\ell)} + \boldsymbol{\Theta}_t^{(\ell)} \boldsymbol{h}_k^{(\ell)} + \boldsymbol{\Theta}_e^{(\ell)} e_{uk}\right)\right)}.$$

About tree algorithms

Minimum Spanning Trees

- Apply to **undirected** graphs;
- Select a subset of edges **connecting all nodes** with:
  - ↳ Minimum total edge weight;
  - ↳ No cycles;
- Efficiently computed using **Kruskal**'s or Prim's algorithms;
- Commonly used in network design, clustering, and optimization problems. (Cong and Zhao 2015)

Minimum Cost Arborescences

- Apply to **directed** graphs;
- Build a rooted spanning tree with:
  - ↳ Minimum total cost of directed edges;
  - ↳ Reachability from the root to all nodes;
- Solved using **Chu-Liu Edmonds**' algorithm (Chu and Liu 1965; Edmonds 1967);
- Useful for hierarchical structures, flow networks, and decision trees.

Weight Cascading Process

Building a diffusion tree

- ▷ We compute a "proximity" matrix between all nodes;
  - ↳ **distance**-based (e.g. euclidean distance);
  - ↳ **spectral**-based:

$$\boldsymbol{L} = \boldsymbol{I} - \frac{1}{2}\left(\boldsymbol{\Phi}^{\frac{1}{2}}\boldsymbol{P}\boldsymbol{\Phi}^{-\frac{1}{2}} + \boldsymbol{\Phi}^{-\frac{1}{2}}\boldsymbol{P}^{\top}\boldsymbol{\Phi}^{\frac{1}{2}}\right)$$

  where $\boldsymbol{P}$ is a transition matrix and $\boldsymbol{\Phi}$ a matrix with the Perron vector of $\boldsymbol{P}$ in the diagonal and zeros elsewhere (Chung 2005);
- ▷ We apply a MST/MCA algorithm to the proximity matrix.

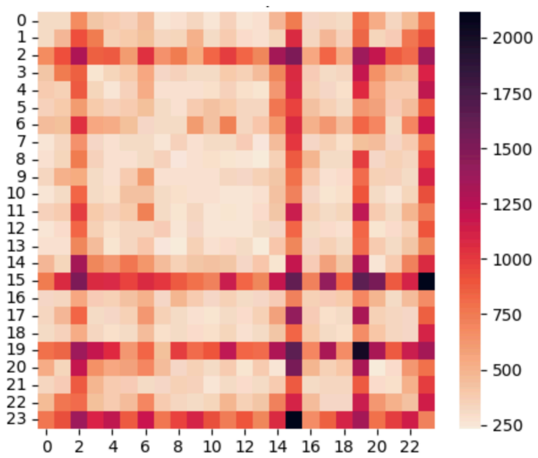


Figure 6 – Distance matrix used to build the diffusion tree.

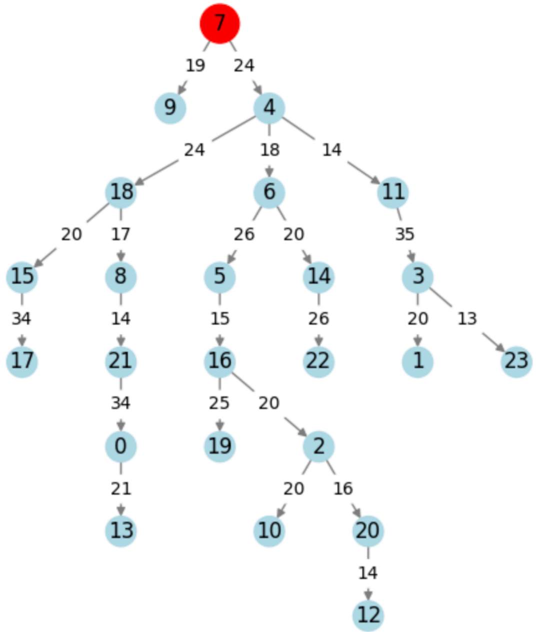


Figure 7 – Diffusion tree built from the distance matrix.

Weight Cascading Process

On the prototype selection

- The **prototype** p is the root of the cascade and acts as a source of weight diffusion;
- Three strategies:
  - ↳ **Centroid**: mean of all points; efficient but sensitive to outliers;
  - ↳ **Medoid**: most central real point (minimum of pairwise distances); robust to outliers;
  - ↳ **Betweenness centrality**: node with highest betweenness centrality in graph; captures topological importance.

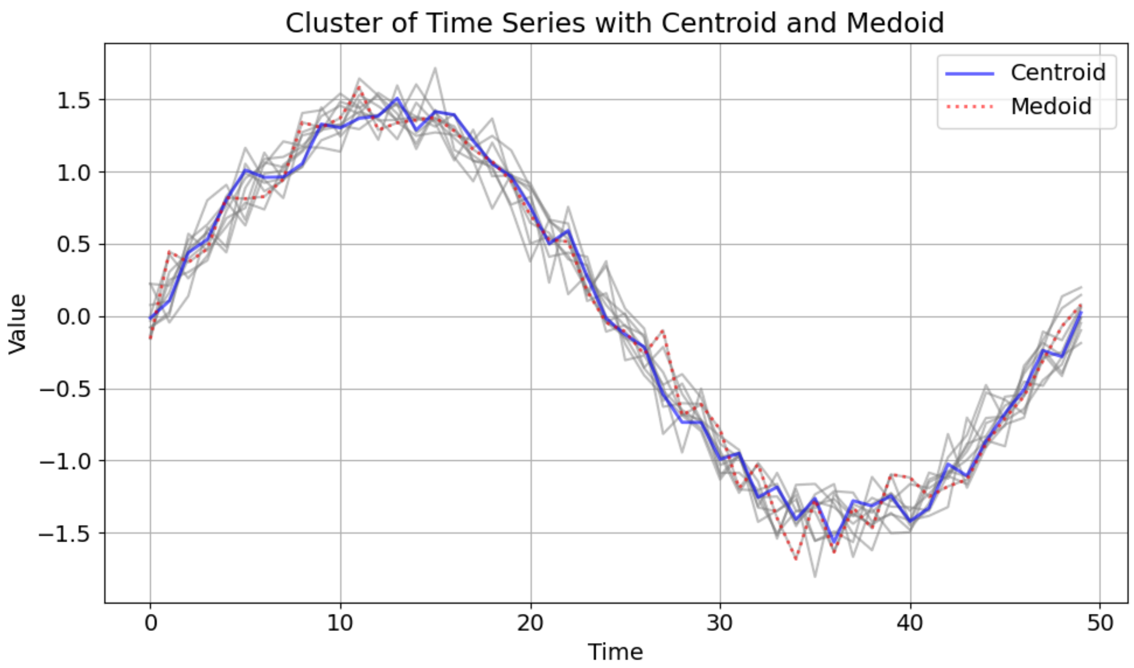


Figure 8 – Centroid and medoid strategies.

Weight Cascading Process

On the prototype selection

- The **prototype** p is the root of the cascade and acts as a source of weight diffusion;
- Three strategies:
  - ↳ **Centroid**: mean of all points; efficient but sensitive to outliers;
  - ↳ **Medoid**: most central real point (minimum of pairwise distances); robust to outliers;
  - ↳ **Betweenness centrality**: node with highest betweenness centrality in graph; captures topological importance.

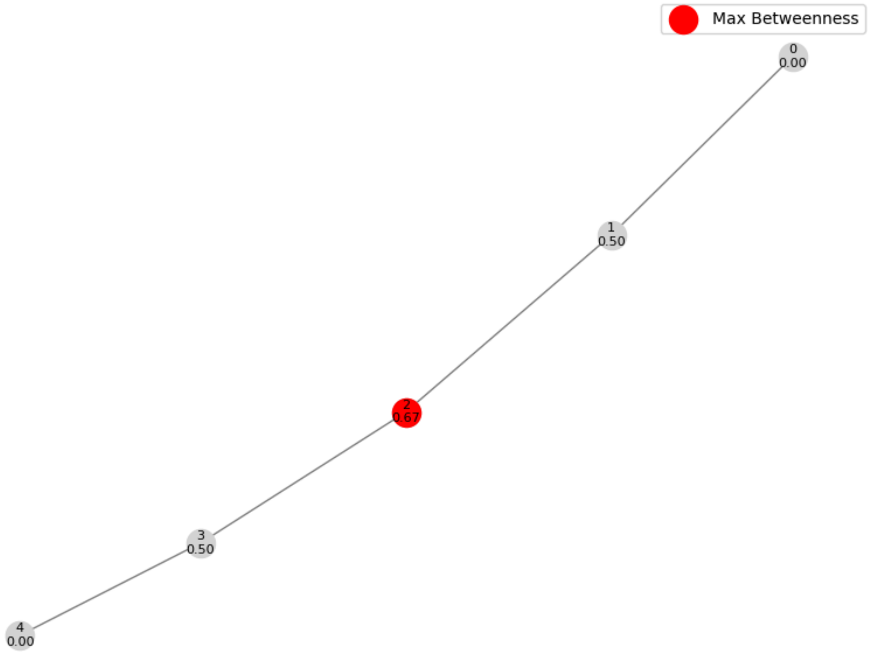


Figure 9 – Betweenness centrality strategy.

Weight Cascading Process

On the algorithm

Cascading algorithm

- Consists of **2 stages**:
  - ↳ $\mathcal{A}_0$: train prototype model on $\boldsymbol{p}$;
  - ↳ $\mathcal{A}_1$: refine each model using parent weights;
- **Single-step**: prototype weights broadcast to all cluster members;
  - ↳ Can use **uniform** or **distance-based budgets**;
- **Multi-step**: weights flow through a tree $\mathcal{T}$ (MST/MCA) from parent to child;
  - ↳ Enables gradual diffusion.

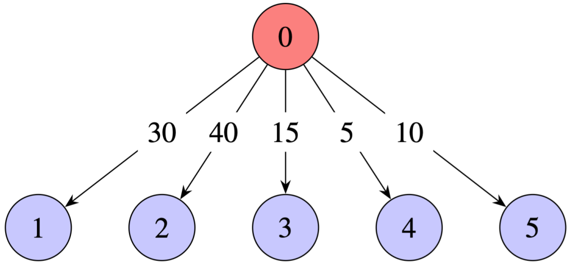


Figure 10 – Single-step cascade for a total budget of 100.

Weight Cascading Process

On the algorithm

Cascading algorithm

- Consists of **2 stages**:
  - ↳ $\mathcal{A}_0$: train prototype model on $\boldsymbol{p}$;
  - ↳ $\mathcal{A}_1$: refine each model using prototype weights;
- **Single-step**: prototype weights broadcast to all cluster members;
  - ↳ Can use **uniform** or **distance-based budgets**;
- **Multi-step**: weights flow through a tree $\mathcal{T}$ (MST/MCA) from parent to child;
  - ↳ Enables gradual diffusion.

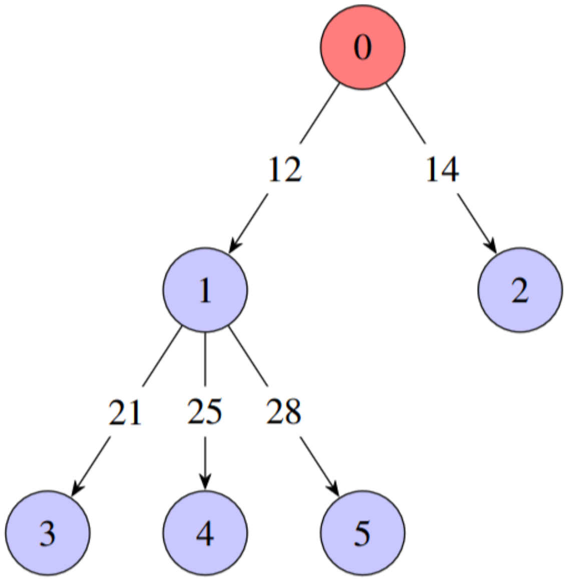


Figure 11 – Multiple-step cascade for a total budget of 100.

About the budget

What is “fair” ?

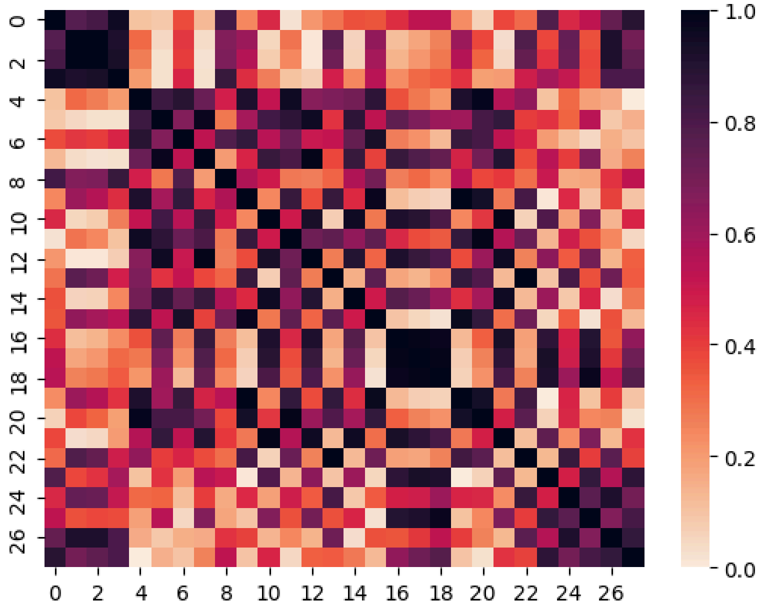
- The learning models deployed across these nodes share a **global computational budget** B, and operate using a **fixed batch size**:
  - ↳ **Individual budgets**: $\forall u \in \mathcal{V}, B_u = \frac{B}{N}$ and models weights are **randomly** initialized;
  - ↳ **Cascade budgets**
    - **Uniform**: $\forall u \in \mathcal{V}, B_u = \frac{B}{N}$, prototype’s model weights are randomly initialized and each child inherits the parent’s weights;
    - **Flexible**: $\forall u, v \in \mathcal{V}, u \to v, \sum_{u \to v} \widetilde{d_{uv}} = 1, B_v = \lceil \widetilde{d_{uv}} B \rceil$; $\left(\sum_{v \in \mathcal{V}} B_v \simeq B\right)$
  - ↳ **GNN budget**: B.

Cascade diffusion experiments on synthetic data

- **Both single-step and multi-step cascades outperform individual models** especially when the budget is **small**;
- Multi-step cascades outperform single-step cascades in some **specific cases** but it is clearly not always the case;
  - ↳ Investigating under **which conditions** multi-step cascades outperform single-step!
- “Well”–chosen diffusion trees **significantly perform better than random** trees.

When Graph Neural Networks Come Into Play

- ▷ **Designing a diffusion tree $\mathcal{T}$**: train a GAT(v2) model and extract **attention weights** to then build a diffusion tree;



- ▷ **A global model**: GNNs can also be used as a global model by relying on the spatial links between nodes to efficiently compute representations.

GNN experiments on real data

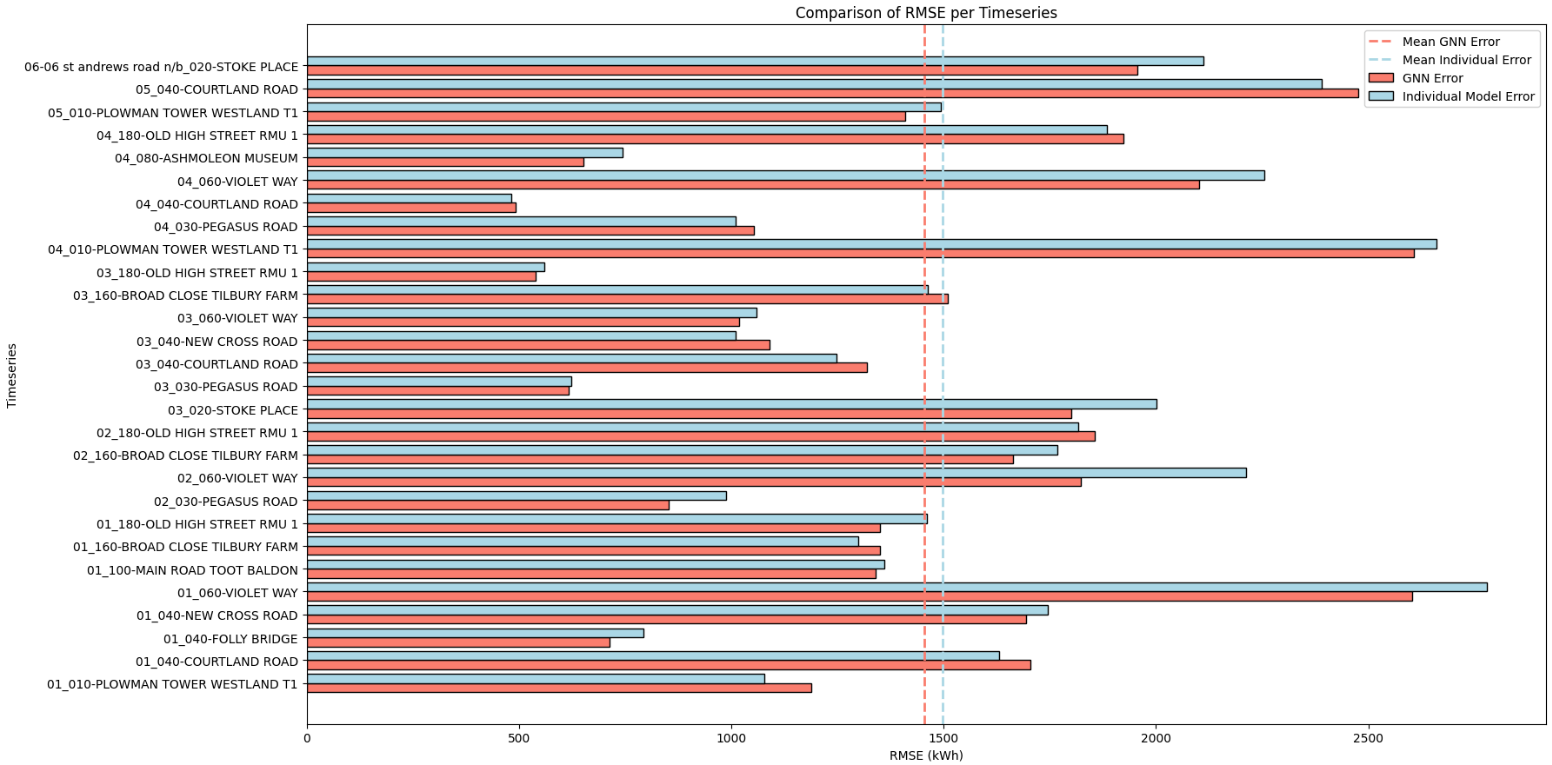


Figure 12 – Comparison of RMSE per timeseries for a **GATv2** and a FFNN with $B_u = 10$.

Conclusion

- ▷ Cascading through MSTs or MCAs enables **low-cost, scalable model refinement**;
- ▷ GNNs can serve as:
  - ↳ a **tree generation method** to diffuse weights across sites;
  - ↳ a **single global model** that captures structural information across sites.

Thanks for your attention!

Feel free to reach out to me at eloi.campagne@ens-paris-saclay.fr.

Bibliography

[1] D. Obst, J. de Vilmarest, and Y. Goude, “Adaptive Methods for Short-Term Electricity Load Forecasting During COVID-19 Lockdown in France.” 2020.

[2] J. de Vilmarest and Y. Goude, “State-Space Models Win the IEEE DataPort Competition on Post-covid Day-ahead Electricity Load Forecasting.” 2021.

[3] M. Gori and F. Monfardini Gabriele Scarselli, “A new model for learning in graph domains,” in *Proceedings. 2005 IEEE International Joint Conference on Neural Networks.*, 2005.

[4] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” 2016.

[5] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, “Graph attention networks,” 2017.

[6] S. Brody, U. Alon, and E. Yahav, “How Attentive are Graph Attention Networks?.” 2022.

[7] F. Cong and Y. Zhao, “The application of minimum spanning tree algorithm in the water supply network,” in *2015 International Industrial Informatics and Computer Engineering Conference*, 2015.

[8] Y.-J. Chu and T.-H. Liu, “On the shortest arborescence of a directed graph,” *Scientia Sinica*, 1965.

[9] J. Edmonds, “Optimum branchings,” *Journal of Research of the national Bureau of Standards B*, 1967.

[10] F. Chung, “Laplacians and the Cheeger inequality for directed graphs,” *Annals of Combinatorics*, 2005.

Appendix

Cascading algorithm

Algorithm 3 MST Cascade.

Input: Data $\{\mathbf{x}_i\}_{i=1}^N$, distance function between tasks $\mathtt{dist}()$, number of clusters K to split the tasks, clustering method $\mathtt{findClusters}()$, method that finds a prototype for a given cluster $\mathtt{findPrototype}()$ budget for prototype training b, total budget per cluster B, training procedures $\mathcal{A}_0$ and $\mathcal{A}_1$.

Output: Refined models $\{f_{\tilde{\boldsymbol{\theta}}_i}\}_{i=1}^N$.

■ (Optional) Partition the problem in a number of non-intersecting clusters.
$\{C_k\}_{k=1}^K \leftarrow \mathtt{findClusters}(\{\mathbf{x}_i\}_{i=1}^N, K)$
$\{\mathbf{D}_k\}_{k=1}^K \leftarrow \mathtt{computeDistanceMatrices}(\{C_k\}_{k=1}^K, \mathtt{dist}())$

■ Process each cluster independently.
for each cluster C **do**
  $\mathcal{T} \leftarrow \mathtt{computeMST}(\mathbf{D})$ ▷ Extract the MST with Kruskal's algorithm
  $\mathbf{p} \leftarrow \mathtt{findPrototype}(C, \mathcal{T})$ ▷ Compute a cluster prototype within $\mathcal{T}$
  $\mathbf{d} \leftarrow \mathtt{extractTreeWeights}(\mathcal{T})$ ▷ Extract the distance vector from $\mathcal{T}$
  $\mathbf{d}' \leftarrow \mathtt{softmax}(\mathbf{d})$ ▷ Normalize the distance vector using softmax
  $f_{\widehat{\boldsymbol{\theta}}} \leftarrow \mathcal{A}_0(f_{\boldsymbol{\theta}}, \mathbf{p}, b)$ ▷ Train the prototype model on cluster data with budget b
  for $j = 1$ **to** $|\mathbf{d}'|$ **do**
    $(\mathbf{b})_j \leftarrow \lceil (\mathbf{d}')_j \cdot B \rceil$ ▷ Compute refinement budget
  end for

  ■ Refine individual models using allocated budgets.
  $\mathcal{Q} \leftarrow \emptyset, \quad \mathcal{Q} \leftarrow \mathtt{enqueue}(\mathcal{Q}, \mathbf{p})$ ▷ Initialize a queue with the prototype node for refinement
  while $\neg$ $\mathtt{isEmpty}(\mathcal{Q})$ **do**
    $\mathbf{x}_{parent} \leftarrow \mathtt{dequeue}(\mathcal{Q})$
    for $\mathbf{x}_{child}$ **in** $\mathtt{childrenOf}(\mathbf{x}_{parent})$ **do**
      $\mathcal{Q} \leftarrow \mathtt{enqueue}(\mathcal{Q}, \mathbf{x}_{child})$ ▷ Add the children of the processed node in the queue
      $f_{\tilde{\boldsymbol{\theta}}_{child}} \leftarrow \mathcal{A}_1(f_{\widehat{\boldsymbol{\theta}}_{parent}}, \mathbf{x}_{child}, \mathbf{x}_{parent}, (\mathbf{b})_{child})$ ▷ Refine child model using parent model, child data and budget
    end for
  end while
end for
return $\{f_{\tilde{\boldsymbol{\theta}}_i}\}_{i=1}^N$.