Graph-based Neural Weather Prediction for Limited Area Modeling

Joel Oskarsson
Linköping University
Linköping, Sweden
joel.oskarsson@liu.se

Tomas Landelius
SMHI
Norrköping, Sweden
tomas.landelius@smhi.se

Fredrik Lindsten
Linköping University
Linköping, Sweden
fredrik.lindsten@liu.se

Abstract

The rise of accurate machine learning methods for weather forecasting is creating radical new possibilities for modeling the atmosphere. In the time of climate change, having access to high-resolution forecasts from models like these is also becoming increasingly vital. While most existing Neural Weather Prediction (NeurWP) methods focus on global forecasting, an important question is how these techniques can be applied to limited area modeling. In this work we adapt the graph-based NeurWP approach to the limited area setting and propose a multi-scale hierarchical model extension. Our approach is validated by experiments with a local model for the Nordic region.

1 Introduction

Accurately forecasting weather is an immense challenge, but also a problem with broad impact on society. Today, Numerical Weather Prediction (NWP) systems combine vast amounts of physics knowledge with powerful computational resources in order to model the atmosphere. Performing these computations is however time-consuming, limiting the possibility to model finer resolutions and detect rare events [1]. Forecasting these events is vital in adapting society to the effects of climate change [2]. Recently, data-driven machine learning models have shown impressive weather forecasting performance, matching or even outperforming existing NWP systems [3–5]. These Neural Weather Prediction (NeurWP) models produce forecasts in a fraction of the time of traditional NWP, opening up many new possibilities. One successful family of NeurWP models, that we focus on in this work, utilize Graph Neural Networks (GNNs) to produce forecasts [6, 5]. As these models are trained on NWP data, with built-in physics assumptions, they represent a useful blend of existing knowledge and new possibilities introduced by machine learning.

While most existing work in NeurWP has been focused on global weather forecasting, it is of substantial interest how these methods can be utilized also for Limited Area Models (LAMs). Such local models are used by many institutes to create high resolution forecasts for specific regions of interest [7–14]. In this paper we adapt graph-based NeurWP to the limited area setting, and also introduce a hierarchical model extension.[1] The graph-based framework is an attractive choice for LAMs due to the freedom in designing the associated graphs. By choosing these graphs appropriately, a general model formulation can be applied to regions of different shape or alignment with global coordinate systems. We evaluate our graph-based models on a dataset from the MetCoOp Ensemble Prediction System (MEPS) LAM [7].

Climate Impact Due to climate change, access to efficient and accurate weather modeling is increasingly important. Fast NeurWP LAMs contribute to climate change mitigation and adaptation

[1] Our code is available at `https://github.com/joeloskarsson/neural-lam`.

Tackling Climate Change with Machine Learning: workshop at NeurIPS 2023.

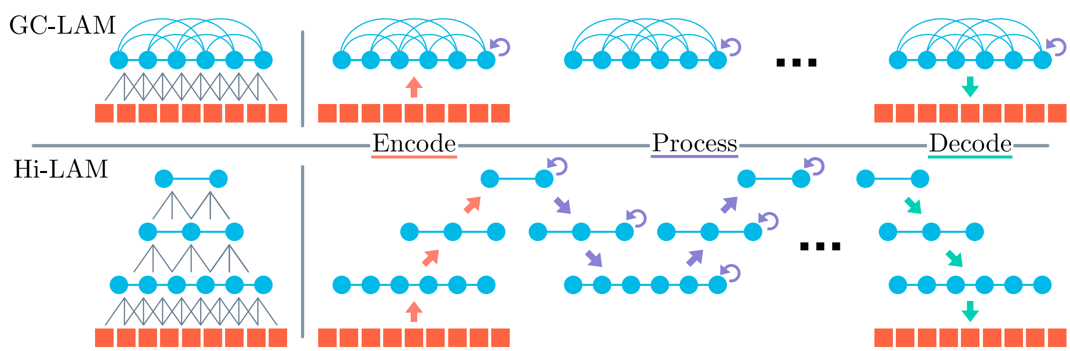


Figure 1: Overview of the prediction process of the GC-LAM and Hi-LAM models. Inputs at grid nodes ■ are encoded to mesh nodes ●, processed and decoded back to produce a one step prediction.

through: 1) Extreme weather prediction, made possible by fast NeurWP models enabling large ensemble forecasts. Detecting extreme events requires modeling the full distribution of weather states, which is typically done by creating an ensemble of forecasts from perturbed versions of initial states [15]. 2) Providing accurate forecasts for renewable energy production, enabling better integration of these volatile sources into existing energy systems. 3) Reducing the energy required to produce fine-grained weather forecasts [16]. In appendix A we discuss paths to climate impact in more detail.

2 Related Work

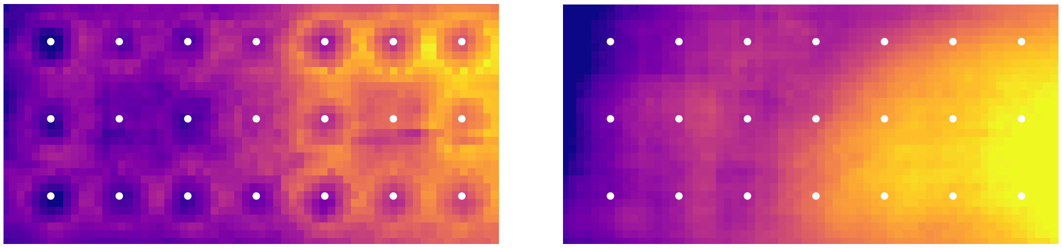
Global NeurWP Our work builds on graph-based NeurWP [6] and in particular the GraphCast model [5]. Apart from this approach, another family of global NeurWP models rely on transformer architectures [17, 18], treating spatial or spatio-temporal patches of the weather state as input tokens [4, 19, 16, 20, 21]. Some of these transformer-based approaches have been framed as *foundation models*, aiming to learn general representations of the atmosphere that can be utilized in diverse downstream tasks [20, 21]. Such models can be fine-tuned for limited area forecasting, but using the same variables as the global models and an area aligned with the global latitude-longitude grid. Other noteworthy models are based on *neural operator* architectures [22, 23], which perform efficient global convolutions by operating in the frequency domain [16, 24].

Hierarchical Graph Neural Networks Our proposed hierarchical GNN structure shares many ideas with the models of Fortunato et al. [25] and Lino et al. [26]. While also motivated by capturing multiple spatial scales, they apply hierarchical GNNs to general Partial Differential Equation (PDE) solving, rather than the weather forecasting task.

3 Graph-based Forecasting for a Limited Area

The vast majority of NeurWP models learn to approximate a function $X^t = f\left(X^{t-1}, \dots, X^{t-p}\right)$ that autoregressively maps from the p last weather states to the state X^t at time step t [4, 6, 16]. Each such weather state X^t is a matrix of shape $N \times S$ containing S different atmospheric variables at N different spatial locations. Each atmospheric variable is typically modeled at a number of vertical levels, but for clarity we here consider all levels for all variables stacked as one vector of length S. We refer to the N spatial locations as *grid cells* or (in the graph context) as *grid nodes*. In global models each such grid node typically corresponds to a cell in a latitude-longitude grid [4, 5]. Using machine learning, a mapping $\hat{f} \approx f$ can be learned from a dataset of weather state trajectories $X^1, X^2, \dots, X^T$. Given p initial states, a full forecast can be produced by unrolling the model multiple time steps, applying $\hat{f}$ iteratively to its own predictions.

Graph-based Forecasting In graph-based NeurWP, $\hat{f}$ is defined as an encode-process-decode sequence using GNNs [6, 5]. See Sanchez-Lengeling et al. [27] for an accessible introduction to GNNs. Input states are *encoded* to the nodes of a *mesh graph*, *processed* through multiple GNN



(a) GC-LAM (b) Hi-LAM

Figure 2: Enlarged prediction of `z_1000`. The GC-LAM model shows circular artifacts centered at mesh nodes with > 8 neighbors (white dots).

layers and then *decoded* back to the grid nodes to produce a prediction $\hat{X}^t$. The process is illustrated in fig. 1. The mesh graph contains fewer nodes than the grid, making it more efficient to work on.

In previous works, Keisler [6] constructed the mesh graph as an icosahedral mesh of triangles around the globe. Lam et al. [5] extended this idea in the GraphCast model, by combining such icosahedral meshes at 7 resolutions. These 7 graphs are merged by taking the union of their edge sets, creating a *multi-scale* mesh graph with edges of varying length. The idea behind the multi-scale graph is that longer edges can propagate information over longer spatial distances, which should be useful for forecasting physical processes taking place at larger spatial scales [5].

We base our first model, titled GC-LAM, on directly adapting the GraphCast model to the limited area setting. This model produces one step predictions using the last two states ($p = 2$) [5]. The previous weather states and known forcing inputs are encoded by Multi-Layer Perceptrons (MLPs) to vector representations in each grid node. Additional static features, associated with edges and mesh nodes, are encoded using separate MLPs. As in GraphCast, all GNNs used in each step of the encode-process-decode sequence are interaction networks [28]. These networks update all node representations $\boldsymbol{v}_{\cdot}$ and edge representations $\boldsymbol{e}_{\cdot\to\cdot}$ according to

$$\tilde{\boldsymbol{e}}_{s\to r} \leftarrow \mathrm{MLP}_E([\boldsymbol{e}_{s\to r}, \boldsymbol{v}_s, \boldsymbol{v}_r]) \tag{1a}$$

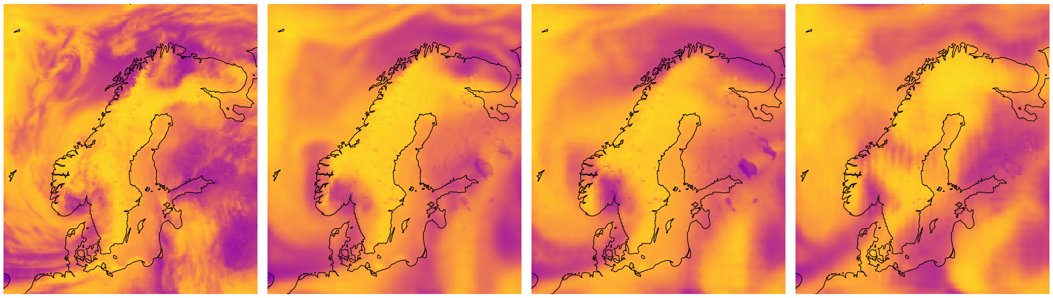
$$\boldsymbol{e}_{s\to r} \leftarrow \boldsymbol{e}_{s\to r} + \tilde{\boldsymbol{e}}_{s\to r}, \qquad \boldsymbol{v}_r \leftarrow \boldsymbol{v}_r + \mathrm{MLP}_V\left(\left[\boldsymbol{v}_r, \textstyle\sum_{s:(s,r)\in\mathcal{E}} \tilde{\boldsymbol{e}}_{s\to r}\right]\right) \tag{1b}$$

where $\mathcal{E}$ is the set of edges used by the the current GNN. In the encoding step these edges go from grid nodes to mesh nodes, in the processing step they connect only mesh nodes and in the decoding step they go from the mesh back to the grid. See appendix B for model details.

Hierarchical Graph Neural Network From initial experiments with the GC-LAM model, we noticed issues with circular artifacts in the predictions. The positions of these artifacts, which can be seen in fig. 2, correspond to mesh nodes with many incoming edges in the multi-scale graph. Motivated by this, we introduce the extended Hi-LAM model with a *hierarchical* mesh graph in L levels. This hierarchy is created by not merging the L meshes of different resolution levels used in the multi-scale graph, but instead introducing additional edges between the levels. With such a hierarchical construction, illustrated in fig. 1, we retain longer spatial edges at higher levels in the hierarchy, but get a more uniform structure in the bottom mesh level that is connected to the grid.

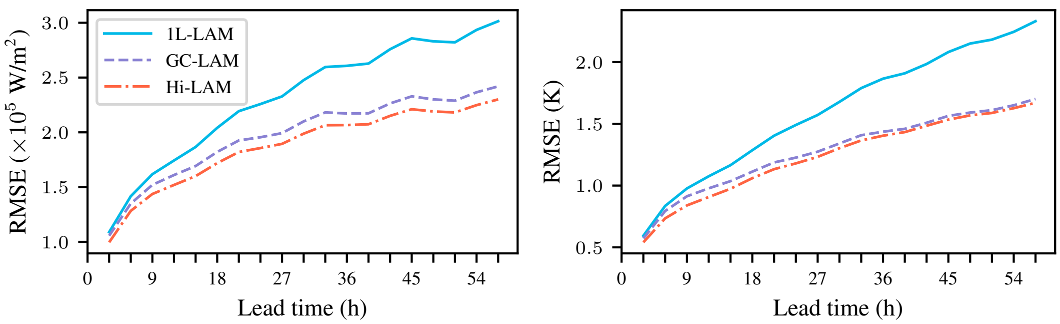
In the hierarchical model we extend the encoding step to not just propagate information from the grid to the bottom mesh level, but also up through the mesh hierarchy. During the processing step we sequentially update the representations at each level in the hierarchy. At each level we run one iterations of intra-level processing and then propagates information on to the next level, in the sequence $L, \ldots, 1, \ldots, L$. It is also possible to run multiple such sweeps down and up through the hierarchy. The decoding step then propagates information down through mesh levels $L, \ldots, 1$ and finally to the grid for prediction. All steps are parametrized using independent GNNs.

Forecasting for a Limited Area A major part of adapting the graph-based models to the limited area setting is to redesign the mesh graph. We will here specifically consider the operational area of the MEPS forecasting system, but the method is general. The MEPS area is defined by drawing a rectangular grid on a Lambert conformal conic map projection [7], which yields approximately



(a) Ground Truth (b) Hi-LAM (c) GC-LAM (d) 1L-LAM

Figure 3: Ground truth and example forecasts of `nlwrs` at lead time 57 h.



(a) `nlwrs` (b) `t_2`

Figure 4: RMSE of net longwave solar radiation flux (`nlwrs`) and temperature 2 m above ground (`t_2`), evaluated at different lead times.

equally sized grid cells. We define our mesh graphs as regular quadrilateral meshes covering the same area, but with far less nodes than the original grid. Regular meshes are created at $L = 4$ different resolutions and either merged for GC-LAM or connected with additional edges in Hi-LAM. More details on the graph construction are given in appendix D.

In limited area NWP, the regional model needs information from the surrounding area. This is provided by updating lateral boundary conditions with data from a (often global) host model [7, 29, 30]. We follow the same methodology for our NeurWP model, including *boundary forcing* at each time step. To incorporate this information, we let the model output an initial prediction $\hat{X}^t$ and then replace each row $\hat{X}^t_v$ with X^t_v if grid node v lies within the boundary area. When unrolling for multiple time steps this boundary forcing will then be fed back into the model through $\hat{X}^t$. Note that we here use the ground truth weather state in the dataset as the boundary forcing, but in operational scenarios this could be replaced by a forecast from an external host model.

4 Experiments with MEPS Data

Dataset and Models We experiment with limited area NeurWP on a dataset of historical forecasts from the MEPS system [7]. Training on such data allows for building a fast surrogate model approximating the system. The dataset contains 6069 forecasts from the years 2021–2023. We use the first 15 months for training and validation and the remaining as a test set. In total we model $S = 17$ atmospheric variables, including temperature, wind and net solar radiation, among others (see details in appendix C). Our MEPS data contains a grid of 238×268 nodes, resulting in a spatial resolution of 10 km. We evaluate the models based on 57 h forecasts with 3 h time steps. This is close to the 66 h forecasts used in the operational system [7]. Apart from the GC-LAM and Hi-LAM models we also train a model (titled 1L-LAM) without any multi-scale edges, using a regular mesh of only a single resolution. All models were trained first for single step prediction and then fine-tuned using 4 time step rollouts.

Results We showcase example forecasts in fig. 3 and Root Mean Squared Error (RMSE) values for net longwave radiation (`nlwrs`) and 2 m temperature (`t_2`) in fig. 4. More results, including errors and example forecasts for all variables can be found in appendix E. While the RMSE metric gives a useful overview of the model performance, equally important are the qualitative comparisons in figs. 2 and 3 and appendix E.2. In general we observe that the Hi-LAM model does not just reduce visual artifacts, but also gives overall lower RMSE. The performance of the 1L-LAM model is substantially worse than the others, verifying the importance of including edges of multiple spatial scales in the model. The predictions of the 1L-LAM model also show severe visual artifacts. Producing a 57 h forecast with Hi-LAM takes only 1.5 seconds on a single A100 GPU, making the method promising for the type of large ensemble forecasting necessary to model extreme weather events.

Future Work A general issue in existing NeurWP models, ours included, is that they produce over-smooth, unrealistic forecasts [31, 5]. This is a consequence of the loss-functions used, making the models output only the mean of the predictive distribution. To combat over-smoothing and increase the usefulness of NeurWP models we believe that future works should move in the direction of more probabilistic and generative modeling. For improving NeurWP LAMs there is also much to explore in how to best make use of learned representations or forecasts from existing global models. In exemplifying a framework of graph-based NeurWP LAMs we hope to inspire the community to further explore also this modeling direction and its unique challenges.

Acknowledgments and Disclosure of Funding

We would like to thank Filip Ekström Kelvinius for helpful discussions about the GNN implementation. This research is financially supported by the Swedish Research Council via the project *Handling Uncertainty in Machine Learning Systems* (contract number: 2020-04122), the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, the Excellence Center at Linköping–Lund in Information Technology (ELLIIT), and European Union's Horizon 2020 research and innovation program under grant agreements 646039, 775970 and 883973. Our computations were enabled by the Berzelius resource at the National Supercomputer Centre, provided by the Knut and Alice Wallenberg Foundation.

References

[1] Marina Astitha, Linus Magnusson, and Efthymios Nikolopoulos. Weather forecasting (subchapter 1.2). In *Extreme Weather Forecasting*, pages 7–14. Elsevier, 2023.

[2] Core writing team, Hoesung Lee, and José Romero (eds.). IPCC, 2023: Climate change 2023: Synthesis report. contribution of working groups I, II and III to the sixth assessment report of the intergovernmental panel on climate change. Technical report, Intergovernmental Panel on Climate Change (IPCC), Geneva, Switzerland, 2023.

[3] Zied Ben-Bouallegue, Mariana C. A. Clare, Linus Magnusson, Estibaliz Gascon, Michael Maier-Gerber, Martin Janousek, Mark Rodwell, Florian Pinault, Jesper S. Dramsch, Simon T. K. Lang, Baudouin Raoult, Florence Rabier, Matthieu Chevallier, Irina Sandu, Peter Dueben, Matthew Chantry, and Florian Pappenberger. The rise of data-driven weather forecasting. *arXiv preprint arXiv:2307.10128*, 2023.

[4] Kaifeng Bi, Lingxi Xie, Hengheng Zhang, Xin Chen, Xiaotao Gu, and Qi Tian. Accurate medium-range global weather forecasting with 3d neural networks. *Nature*, pages 1–6, 2023.

[5] Remi Lam, Alvaro Sanchez-Gonzalez, Matthew Willson, Peter Wirnsberger, Meire Fortunato, Alexander Pritzel, Suman Ravuri, Timo Ewalds, Ferran Alet, Zach Eaton-Rosen, Weihua Hu, Alexander Merose, Stephan Hoyer, George Holland, Jacklynn Stott, Oriol Vinyals, Shakir Mohamed, and Peter Battaglia. GraphCast: Learning skillful medium-range global weather forecasting. *arXiv preprint arXiv:2212.12794*, 2022.

[6] Ryan Keisler. Forecasting global weather with graph neural networks. *arXiv preprint arXiv:2202.07575*, 2022.

[7] Malte Müller, Mariken Homleid, Karl-Ivar Ivarsson, Morten A. Ø Køltzow, Magnus Lindskog, Knut Helge Midtbø, Ulf Andrae, Trygve Aspelien, Lars Berggren, Dag Bjørge, Per Dahlgren, Jørn Kristiansen, Roger Randriamampianina, Martin Ridal, and Ole Vignes. AROME-MetCoOp: A nordic convective-scale operational weather prediction model. *Weather and Forecasting*, 32(2):609–627, 2017.

[8] Clifford Mass. The uncoordinated giant ii: Why u.s. operational numerical weather prediction is still lagging and how to fix it. *Bulletin of the American Meteorological Society*, 104(4), 2023.

[9] Mike Bush, Ian Boutle, John Edwards, Anke Finnenkoetter, Charmaine Franklin, Kirsty Hanley, Aravindakshan Jayakumar, Huw Lewis, Adrian Lock, Marion Mittermaier, Saji Mohandas, Rachel North, Aurore Porson, Belinda Roux, Stuart Webster, and Mark Weeks. The second met office unified model–JULES regional atmosphere and land configuration, RAL2. *Geoscientific Model Development*, 16(6):1713–1734, 2023.

[10] Yann Seity, Pierre Brousseau, Sophie Malardel, G. Hello, P. Bénard, F. Bouttier, C. Lac, and V. Masson. The AROME-france convective-scale operational model. *Monthly Weather Review*, 139(3):976 – 991, 2011.

[11] Trang Van Pham, Christian Steger, Burkhardt Rockel, Klaus Keuler, Ingo Kirchner, Mariano Mertens, Daniel Rieger, Günther Zängl, and Barbara Früh. ICON in climate limited-area mode (ICON release version 2.6.1): a new regional climate model. *Geoscientific Model Development*, 14(2):985–1005, 2021.

[12] Jordan G. Powers, Joseph B. Klemp, William C. Skamarock, Christopher A. Davis, Jimy Dudhia, David O. Gill, Janice L. Coen, David J. Gochis, Ravan Ahmadov, Steven E. Peckham, Georg A. Grell, John Michalakes, Samuel Trahan, Stanley G. Benjamin, Curtis R. Alexander, Geoffrey J. Dimego, Wei Wang, Craig S. Schwartz, Glen S. Romine, Zhiquan Liu, Chris Snyder, Fei Chen, Michael J. Barlage, Wei Yu, and Michael G. Duda. The weather research and forecasting model: Overview, system efforts, and future directions. *Bulletin of the American Meteorological Society*, 98(8):1717 – 1737, 2017.

[13] Xueshun Shen, Jianjie Wang, Zechun Li, Dehui Chen, and Jiandong Gong. Research and operational development of numerical weather prediction in china. *Journal of Meteorological Research*, 34(4):675–698, 2020.

[14] Kosuke Ono, Masaru Kunii, and Yuki Honda. The regional model-based mesoscale ensemble prediction system, MEPS, at the japan meteorological agency. *Quarterly Journal of the Royal Meteorological Society*, 147(734):465–484, 2021.

[15] Jean Coiffier. *Fundamentals of numerical weather prediction*. Cambridge University Press, 2011.

[16] Jaideep Pathak, Shashank Subramanian, Peter Harrington, Sanjeev Raja, Ashesh Chattopadhyay, Morteza Mardani, Thorsten Kurth, David Hall, Zongyi Li, Kamyar Azizzadenesheli, Pedram Hassanzadeh, Karthik Kashinath, and Animashree Anandkumar. Fourcastnet: A global data-driven high-resolution weather model using adaptive fourier neural operators. *arXiv preprint arXiv:2202.11214*, 2022.

[17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, volume 30, 2017.

[18] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10012–10022, 2021.

[19] Kang Chen, Tao Han, Junchao Gong, Lei Bai, Fenghua Ling, Jing-Jia Luo, Xi Chen, Leiming Ma, Tianning Zhang, Rui Su, Yuanzheng Ci, Bin Li, Xiaokang Yang, and Wanli Ouyang. FengWu: Pushing the skillful global medium-range weather forecast beyond 10 days lead. *arXiv preprint arXiv:2304.02948*, 2023.

[20] Tung Nguyen, Johannes Brandstetter, Ashish Kapoor, Jayesh K Gupta, and Aditya Grover. ClimaX: A foundation model for weather and climate. In *Proceedings of the 40th International Conference on Machine Learning*, pages 25904–25938, 2023.

[21] Christian Lessig, Ilaria Luise, Bing Gong, Michael Langguth, Scarlet Stadler, and Martin Schultz. AtmoRep: A stochastic model of atmosphere dynamics using large scale representation learning. *arXiv preprint arXiv:2308.13280*, 2023.

[22] Zongyi Li, Nikola Borislavov Kovachki, Kamyar Azizzadenesheli, Burigede liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations*, 2021.

[23] John Guibas, Morteza Mardani, Zongyi Li, Andrew Tao, Anima Anandkumar, and Bryan Catanzaro. Adaptive fourier neural operators: Efficient token mixers for transformers. In *International Conference on Learning Representations*, 2022.

[24] Boris Bonev, Thorsten Kurth, Christian Hundt, Jaideep Pathak, Maximilian Baust, Karthik Kashinath, and Anima Anandkumar. Spherical Fourier neural operators: Learning stable dynamics on the sphere. In *Proceedings of the 40th International Conference on Machine Learning*, pages 2806–2823, 2023.

[25] Meire Fortunato, Tobias Pfaff, Peter Wirnsberger, Alexander Pritzel, and Peter Battaglia. Multiscale meshgraphnets. In *ICML 2022 2nd AI for Science Workshop*, 2022.

[26] Mario Lino, Chris Cantwell, Anil A. Bharath, and Stathi Fotiadis. Simulating continuum mechanics with multi-scale graph neural networks. *arXiv preprint arXiv:2106.04900*, 2021.

[27] Benjamin Sanchez-Lengeling, Emily Reif, Adam Pearce, and Alexander B. Wiltschko. A gentle introduction to graph neural networks. *Distill*, 2021. URL `https://distill.pub/2021/gnn-intro`.

[28] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, and koray kavukcuoglu. Interaction networks for learning about objects, relations and physics. In *Advances in Neural Information Processing Systems*, volume 29, 2016.

[29] Eugenia Kalnay. *Atmospheric Modeling, Data Assimilation and Predictability*. Cambridge University Press, 2002.

[30] Fedor Mesinger and Katarina Veljovic. Limited area nwp and regional climate modeling: a test of the relaxation vs eta lateral boundary conditions. *Meteorology and Atmospheric Physics*, 119, 2013.

[31] Stephan Rasp, Stephan Hoyer, Alexander Merose, Ian Langmore, Peter Battaglia, Tyler Russel, Alvaro Sanchez-Gonzalez, Vivian Yang, Rob Carver, Shreya Agrawal, Matthew Chantry, Zied Ben Bouallegue, Peter Dueben, Carla Bromberg, Jared Sisk, Luke Barrington, Aaron Bell, and Fei Sha. WeatherBench 2: A benchmark for the next generation of data-driven global weather models. *arXiv preprint arXiv:2308.15560*, 2023.

A Climate Impact

Accurate weather forecasting has always been massively useful, but in the time of climate change its importance is greater than ever. The capabilities of NeurWP systems introduce great new possibilities which can impact our ability to tackle climate change. In many of these scenarios limited area NeurWP can offer functionality that is of particular usefulness.

Extreme Weather Due to climate change the prevalence and severity of extreme weather events is expected to increase substantially, endangering both property and human life [2]. These events are also getting harder to predict and the cost of damages per event has increased nearly 77% over the past five decades [32]. In order to detect extreme events, there is a need to model the full distribution of possible weather states. To achieve this, *ensemble forecasts* are typically used, where multiple weather forecasts are produced from perturbed versions of initial states [15]. While ensemble forecasting historically has been limited by the computational costs of NWP systems, efficient NeurWP models has the potential to vastly improve our ability to model extreme weather. For tracking extreme weather it is desirable to model at a high resolution mainly the area where the weather event is expected to unfold. Such investigations are enabled by limited area NeurWP.

Renewable Energy The production of renewable energy, such as solar and wind power, can be highly volatile [33]. This creates challenges for including these sources in the larger energy system. Accurate weather forecasts, translated into forecasts of energy production, fill an important role as enablers of these energy sources by making their output predictable. Here, fast limited area NeurWP models enable high resolution modeling in the areas where renewable energy is produced. Detailed probability estimates from ensemble forecasting additionally allow for improved cost-loss decision making in these systems.

The Energy Footprint of NWP Traditional NWP systems utilize massive computing clusters [15], resulting in a substantial energy footprint of the forecasting process. In comparison, NeurWP models are highly energy efficient, even when taking into account their initial training process. The total energy required to train the large global NeurWP model FourCastNet is comparable to running a 10 day forecast with 50 ensemble members using a traditional NWP system [16]. Producing a forecast using the same model uses four orders of magnitude less energy than traditional NWP. A similar reduction in energy footprint is to be expected for LAMs.

Interactive Digital Models By creating interactive digital models of the earth scientists and non-experts can perform simulations to aid policy-making and societal adaptation to climate change. An example of such a model is the *Destination Earth* initiative of the European Commission [34]. Fast and efficient NeurWP is an important enabler for much of the monitoring and prediction capabilities of these models. One promise of digital twin models is to be able to zoom in and with high-resolution details study extreme weather events. Limited area NeurWP is a natural and appealing choice to achieve such functionality.

National and Local Community Agency In a future where climate modeling is of increasing importance, both global and local models have important roles to play. Global models, often enabled by international collaborative efforts, are important for understanding large-scale climate processes and provide necessary boundary information to LAMs [15, 7]. Local models complement these in allowing for studying regions of specific interest to single nations or communities. Limited area NeurWP bring the capabilities of machine learning models to local actors. This allows them to use these models to forecast and study what is important to the local community they serve. We hope that the effect of this will be increased agency of non-global organization to steer their own climate change adaptation efforts.

B Model Details

In this appendix we provide more details on the models used.

B.1 GraphCast (GC-LAM and 1L-LAM)

We start by re-iterating the steps of the GraphCast model from Lam et al. [5]. GC-LAM and 1L-LAM follow exactly the same procedure for making predictions, but using graphs adapted to our setting. To compute a one time step prediction, the function $\hat{f}$ in GraphCast operates in three steps:

1. **Encoding**: Encode grid node information to the mesh.
2. **Processing**: Perform multiple steps of processing on the mesh.
3. **Decoding**: Decode from the mesh nodes down to the grid nodes for prediction.

Each of the steps above is associated with one set of edges. Grid2Mesh edges $\mathcal{E}^{\text{G2M}}$ are used during encoding, Mesh2Mesh edges $\mathcal{E}^{\text{M2M}}$ during processing and Mesh2Grid edges $\mathcal{E}^{\text{M2G}}$ during decoding. The edges in $\mathcal{E}^{\text{G2M}}$ and $\mathcal{E}^{\text{M2G}}$ are directed, whereas $\mathcal{E}^{\text{M2M}}$ contains bidirectional edges[2].

Encoding The input to this whole procedure is a set of vector representations for nodes and edges. Previous weather states, forcing inputs and static grid node features are concatenated and encoded using an MLP into a vector $\boldsymbol{v}_s^G$ for each grid node s. Similarly, static features of each mesh node r are encoded into a vector representation $\boldsymbol{v}_r^M$. Also the static features of edges are encoded into representations $\boldsymbol{e}_{s\to r}^{\text{G2M}}$, $\boldsymbol{e}_{s\to r}^{\text{M2M}}$ and $\boldsymbol{e}_{s\to r}^{\text{M2G}}$ using separate MLPs. The notation $s \to r$ describes that a vector is associated with the edge from node s to node r. Each of the nodes s and r can belong either to the grid or the mesh depending on the context. For example, in $\boldsymbol{e}_{s\to r}^{\text{G2M}}$ the node s belongs to the grid and r to the mesh. All MLPs used have one hidden layer with Swish activation function [35] and are followed by a LayerNorm [36] layer.

The GNN layers used in GraphCast are Interaction Networks [28], which update node and edge representations using two MLPs. During encoding, one round of GNN message passing is performed from the grid to the mesh. The complete update being performed by this step is

$$\tilde{\boldsymbol{e}}_{s\to r}^{\text{G2M}} \leftarrow \text{MLP}_E^{\text{G2M}}\left(\left[\boldsymbol{e}_{s\to r}^{\text{G2M}}, \boldsymbol{v}_s^G, \boldsymbol{v}_r^M\right]\right) \tag{2a}$$

$$\boldsymbol{e}_{s\to r}^{\text{G2M}} \leftarrow \boldsymbol{e}_{s\to r}^{\text{G2M}} + \tilde{\boldsymbol{e}}_{s\to r}^{\text{G2M}} \tag{2b}$$

$$\boldsymbol{v}_s^G \leftarrow \boldsymbol{v}_s^G + \text{MLP}_G^{\text{G2M}}\left(\boldsymbol{v}_s^G\right) \tag{2c}$$

$$\boldsymbol{v}_r^M \leftarrow \boldsymbol{v}_r^M + \text{MLP}_V^{\text{G2M}}\left(\left[\boldsymbol{v}_r^M, \sum_{s:(s,r)\in\mathcal{E}^{\text{G2M}}} \tilde{\boldsymbol{e}}_{s\to r}^{\text{G2M}}\right]\right) \tag{2d}$$

where $[\cdot]$ denotes vector concatenation. The updates in each such step are always performed concurrently for all nodes or edges. Note that eqs. 2b–d describe residual connections in the updates of all node and edge representations. The point of updating also the grid node representations here is that they will be used again in the decoder step.

Processing The processing GNNs operate only on the mesh nodes using $\mathcal{E}^{\text{M2M}}$. There can be arbitrary many layers of processing GNNs and these generally do not share parameters. One processing layer k performs the update

$$\tilde{\boldsymbol{e}}_{s\to r}^{\text{M2M}} \leftarrow \text{MLP}_{E,k}^{\text{M2M}}\left(\left[\boldsymbol{e}_{s\to r}^{\text{M2M}}, \boldsymbol{v}_s^M, \boldsymbol{v}_r^M\right]\right) \tag{3a}$$

$$\boldsymbol{e}_{s\to r}^{\text{M2M}} \leftarrow \boldsymbol{e}_{s\to r}^{\text{M2M}} + \tilde{\boldsymbol{e}}_{s\to r}^{\text{M2M}} \tag{3b}$$

$$\boldsymbol{v}_r^M \leftarrow \boldsymbol{v}_r^M + \text{MLP}_{V,k}^{\text{M2M}}\left(\left[\boldsymbol{v}_r^M, \sum_{s:(s,r)\in\mathcal{E}^{\text{M2M}}} \tilde{\boldsymbol{e}}_{s\to r}^{\text{M2M}}\right]\right). \tag{3c}$$

Decoding The decoder utilizes $\mathcal{E}^{\text{M2G}}$ to arrive at the final grid node representations as

$$\tilde{\boldsymbol{e}}_{s\to r}^{\text{M2G}} \leftarrow \text{MLP}_E^{\text{M2G}}\left(\left[\boldsymbol{e}_{s\to r}^{\text{M2G}}, \boldsymbol{v}_s^M, \boldsymbol{v}_r^G\right]\right) \tag{4a}$$

$$\boldsymbol{v}_r^G \leftarrow \boldsymbol{v}_r^G + \text{MLP}_V^{\text{M2G}}\left(\left[\boldsymbol{v}_r^G, \sum_{s:(s,r)\in\mathcal{E}^{\text{M2G}}} \tilde{\boldsymbol{e}}_{s\to r}^{\text{M2G}}\right]\right). \tag{4b}$$

[2] i.e. $\mathcal{E}^{\text{M2M}}$ also contains directed edges, but satisfies $(a, b) \in \mathcal{E}^{\text{M2M}} \Rightarrow (b, a) \in \mathcal{E}^{\text{M2M}}$.

Note that there is no need to update the edge or mesh node representations at this point. The final predicted weather state in each grid node r is then computed as

$$\hat{X}_r^t = \hat{f}\big(X^{t-1}, X^{t-2}\big)_r = X_r^{t-1} + \text{MLP}^{\text{pred}}\big(\boldsymbol{v}_r^G\big). \tag{5}$$

Note that eq. 5 includes a residual connection to the previous weather state, meaning that the output of the model (from MLP^{pred}) is the state difference rather than the new state. No LayerNorm is used in MLP^{pred}.

B.2 Hi-LAM

Motivation GraphCast augments the original graph construction from Keisler [6] by the addition of multi-scale edges, but does not introduce extra nodes [5]. When the meshes of multiple resolutions are merged, the final multi-scale graph will share its node set with the finest resolution mesh. Nodes with long incoming multi-scale edges will aggregate information from longer spatial distances. This information can then spread locally to neighboring nodes through additional layers of processing GNNs. However, since no new nodes are introduced in the multi-scale architecture, some nodes will 1) have many more neighbors than others 2) aggregate both highly local and long-range information. All nodes need to contain relevant local information, which should be decoded to the connected grid nodes for prediction. Nodes with long multi-scale edges will thus have a dual-responsibility: retain local information and spread long-range information. We hypothesize that these nodes with more neighbors will see a distributional shift in their representation vectors, which would explain the artifacts observed in fig. 2. As the GNN parameters in each layer are shared for all mesh nodes these issues could make it hard for the model to learn useful processing layers.

We propose to not just add multi-scale edges, but create a hierarchical structure of nodes and edges that alleviates the problems mentioned above. To achieve this, we extend the encoding and decoding steps of GraphCast and fully redefine the processing step.

Hierarchical Structure Our hierarchical multi-scale mesh graph consists of L levels of nodes, with level 1 at the bottom (connected to the grid) and level L at the top. Higher levels in the hierarchy contain less nodes with longer spatial distances between them. While we think of and visualize these levels at different height there is no vertical component to the node positions and all graphs lay flat on the surface of the globe. Let $\boldsymbol{v}_r^{M[l]}$ denote the vector representation of mesh node r in level l. We retain the edge sets $\mathcal{E}^{\text{G2M}}$ and $\mathcal{E}^{\text{M2G}}$ from earlier, but connect grid nodes only to mesh nodes at the lowest level $l = 1$. Instead of a single set of mesh edges, the hierarchical setup will require multiple edge sets connecting nodes both inside and between levels. For each mesh level l we define $\mathcal{E}^{[l \leftrightarrow l]}$ as the (bidirectional) edges connecting nodes internally in this level. To propagate information up in the hierarchy, we define sets $\mathcal{E}^{[l \nearrow l+1]}$ for $l \in \{1, \dots, L-1\}$ with directed edges from nodes in level l to nodes in level $l+1$. Similarly, we define downward edges $\mathcal{E}^{[l \searrow l-1]}$ for $l \in \{2, \dots, L\}$. Static features for all these edges are initially encoded into representations $\boldsymbol{e}_{s\to r}^{[l \leftrightarrow l]}$, $\boldsymbol{e}_{s\to r}^{[l \nearrow l+1]}$, $\boldsymbol{e}_{s\to r}^{[l \searrow l-1]}$.

Encoding The encoding in eq. 2 will only spread information from the grid nodes to the bottom level of the mesh hierarchy. If we would start the processing from there, the upper levels of the hierarchy would initially only be processing information from static features. As these static features are generally not key to making the prediction, this would be a waste of computation and memory.

In order to avoid this we augment the encoding step to propagate information up from the grid to the top of the hierarchy. The new encoding step starts by encoding from the grid to the bottom mesh level as

$$\tilde{\boldsymbol{e}}_{s\to r}^{\text{G2M}} \leftarrow \text{MLP}_E^{\text{G2M}}\Big(\Big[\boldsymbol{e}_{s\to r}^{\text{G2M}}, \boldsymbol{v}_s^G, \boldsymbol{v}_r^{M[1]}\Big]\Big) \tag{6a}$$

$$\boldsymbol{e}_{s\to r}^{\text{G2M}} \leftarrow \boldsymbol{e}_{s\to r}^{\text{G2M}} + \tilde{\boldsymbol{e}}_{s\to r}^{\text{G2M}} \tag{6b}$$

$$\boldsymbol{v}_s^G \leftarrow \boldsymbol{v}_s^G + \text{MLP}_G^{\text{G2M}}\big(\boldsymbol{v}_s^G\big) \tag{6c}$$

$$\boldsymbol{v}_r^{M[1]} \leftarrow \boldsymbol{v}_r^{M[1]} + \text{MLP}_{\text{Enc.}}^{M[1]}\left(\left[\boldsymbol{v}_r^{M[1]}, \sum_{s:(s,r)\in\mathcal{E}^{\text{G2M}}} \tilde{\boldsymbol{e}}_{s\to r}^{\text{G2M}}\right]\right) \tag{6d}$$

and then continues up through the hierarchy

$$\tilde{\boldsymbol{e}}^{[l-1 \nearrow l]}_{s \to r} \leftarrow \text{MLP}^{[l-1 \nearrow l]}_{\text{Enc.}}\left(\left[\boldsymbol{e}^{[l-1 \nearrow l]}_{s \to r}, \boldsymbol{v}^{M[l-1]}_{s}, \boldsymbol{v}^{M[l]}_{r}\right]\right) \tag{7a}$$

$$\boldsymbol{e}^{[l-1 \nearrow l]}_{s \to r} \leftarrow \boldsymbol{e}^{[l-1 \nearrow l]}_{s \to r} + \tilde{\boldsymbol{e}}^{[l-1 \nearrow l]}_{s \to r} \tag{7b}$$

$$\boldsymbol{v}^{M[l]}_{r} \leftarrow \boldsymbol{v}^{M[l]}_{r} + \text{MLP}^{M[l]}_{\text{Enc.}}\left(\left[\boldsymbol{v}^{M[l]}_{r}, \sum_{s:(s,r) \in \mathcal{E}^{[l-1 \nearrow l]}} \tilde{\boldsymbol{e}}^{[l-1 \nearrow l]}_{s \to r}\right]\right) \tag{7c}$$

which is ran sequentially for level $l = 2 \ldots, L$. Equation 7 propagates information up from the previous level through $\boldsymbol{v}^{M[l-1]}_{s}$ and combines this with the static feature embeddings.

Processing We redefine the processing step to sequentially pass through the different levels of the hierarchy, using independent MLPs at each level. One processing layer k performs a sweep from the top level L, down to the bottom mesh level 1 and the all the way up to level L again. The steps of this process alternate between intra-level processing at each level and message passing to the next level in the sweep. In more detail, the downward part of the process is described by

$$\tilde{\boldsymbol{e}}^{[l \leftrightarrow l]}_{s \to r} \leftarrow \text{MLP}^{[l \leftrightarrow l]}_{k\downarrow}\left(\left[\boldsymbol{e}^{[l \leftrightarrow l]}_{s \to r}, \boldsymbol{v}^{M[l]}_{s}, \boldsymbol{v}^{M[l]}_{r}\right]\right) \tag{8a}$$

$$\boldsymbol{e}^{[l \leftrightarrow l]}_{s \to r} \leftarrow \boldsymbol{e}^{[l \leftrightarrow l]}_{s \to r} + \tilde{\boldsymbol{e}}^{[l \leftrightarrow l]}_{s \to r} \tag{8b}$$

$$\boldsymbol{v}^{M[l]}_{r} \leftarrow \boldsymbol{v}^{M[l]}_{r} + \text{MLP}^{M[l]}_{k\downarrow}\left(\left[\boldsymbol{v}^{M[l]}_{r}, \sum_{s:(s,r) \in \mathcal{E}^{[l \leftrightarrow l]}} \tilde{\boldsymbol{e}}^{[l \leftrightarrow l]}_{s \to r}\right]\right) \tag{8c}$$

$$\tilde{\boldsymbol{e}}^{[l \searrow l-1]}_{s \to r} \leftarrow \text{MLP}^{[l \searrow l-1]}_{k}\left(\left[\boldsymbol{e}^{[l \searrow l-1]}_{s \to r}, \boldsymbol{v}^{M[l]}_{s}, \boldsymbol{v}^{M[l-1]}_{r}\right]\right) \tag{8d}$$

$$\boldsymbol{e}^{[l \searrow l-1]}_{s \to r} \leftarrow \boldsymbol{e}^{[l \searrow l-1]}_{s \to r} + \tilde{\boldsymbol{e}}^{[l \searrow l-1]}_{s \to r} \tag{8e}$$

$$\boldsymbol{v}^{M[l-1]}_{r} \leftarrow \boldsymbol{v}^{M[l-1]}_{r} + \text{MLP}^{M[l-1]}_{k\searrow}\left(\left[\boldsymbol{v}^{M[l-1]}_{r}, \sum_{s:(s,r) \in \mathcal{E}^{[l \searrow l-1]}} \tilde{\boldsymbol{e}}^{[l \searrow l-1]}_{s \to r}\right]\right) \tag{8f}$$

ran sequentially for levels $l = L, \ldots, 1$ (skipping eqs. 8d–f on level $l = 1$). Then, the process continues back up as

$$\tilde{\boldsymbol{e}}^{[l \leftrightarrow l]}_{s \to r} \leftarrow \text{MLP}^{[l \leftrightarrow l]}_{k\uparrow}\left(\left[\boldsymbol{e}^{[l \leftrightarrow l]}_{s \to r}, \boldsymbol{v}^{M[l]}_{s}, \boldsymbol{v}^{M[l]}_{r}\right]\right) \tag{9a}$$

$$\boldsymbol{e}^{[l \leftrightarrow l]}_{s \to r} \leftarrow \boldsymbol{e}^{[l \leftrightarrow l]}_{s \to r} + \tilde{\boldsymbol{e}}^{[l \leftrightarrow l]}_{s \to r} \tag{9b}$$

$$\boldsymbol{v}^{M[l]}_{r} \leftarrow \boldsymbol{v}^{M[l]}_{r} + \text{MLP}^{M[l]}_{k\uparrow}\left(\left[\boldsymbol{v}^{M[l]}_{r}, \sum_{s:(s,r) \in \mathcal{E}^{[l \leftrightarrow l]}} \tilde{\boldsymbol{e}}^{[l \leftrightarrow l]}_{s \to r}\right]\right) \tag{9c}$$

$$\tilde{\boldsymbol{e}}^{[l \nearrow l+1]}_{s \to r} \leftarrow \text{MLP}^{[l \nearrow l+1]}_{k}\left(\left[\boldsymbol{e}^{[l \nearrow l+1]}_{s \to r}, \boldsymbol{v}^{M[l]}_{s}, \boldsymbol{v}^{M[l+1]}_{r}\right]\right) \tag{9d}$$

$$\boldsymbol{e}^{[l \nearrow l+1]}_{s \to r} \leftarrow \boldsymbol{e}^{[l \nearrow l+1]}_{s \to r} + \tilde{\boldsymbol{e}}^{[l \nearrow l+1]}_{s \to r} \tag{9e}$$

$$\boldsymbol{v}^{M[l+1]}_{r} \leftarrow \boldsymbol{v}^{M[l+1]}_{r} + \text{MLP}^{M[l+1]}_{k\nearrow}\left(\left[\boldsymbol{v}^{M[l+1]}_{r}, \sum_{s:(s,r) \in \mathcal{E}^{[l \nearrow l+1]}} \tilde{\boldsymbol{e}}^{[l \nearrow l+1]}_{s \to r}\right]\right) \tag{9f}$$

ran sequentially for levels $l = 1, \ldots, L$ (skipping eqs. 9d–f on level $l = L$). It is possible to use multiple processing layers, meaning that multiple such sweeps throughout the hierarchy will be performed. As each edge set and mesh level has its own MLPs, we arrive at a more flexible parametrization than the GraphCast mesh processing. In particular, we learn separate MLPs for each spatial scale, corresponding to the different levels in our hierarchy.

Decoding Similarly to the encoding, we change the decoding step to sequentially aggregate information from the top of the mesh hierarchy down to the lowest level. If we consider each level of the hierarchical mesh graph to model physical processes at a specific spatial scale, this type of decoding

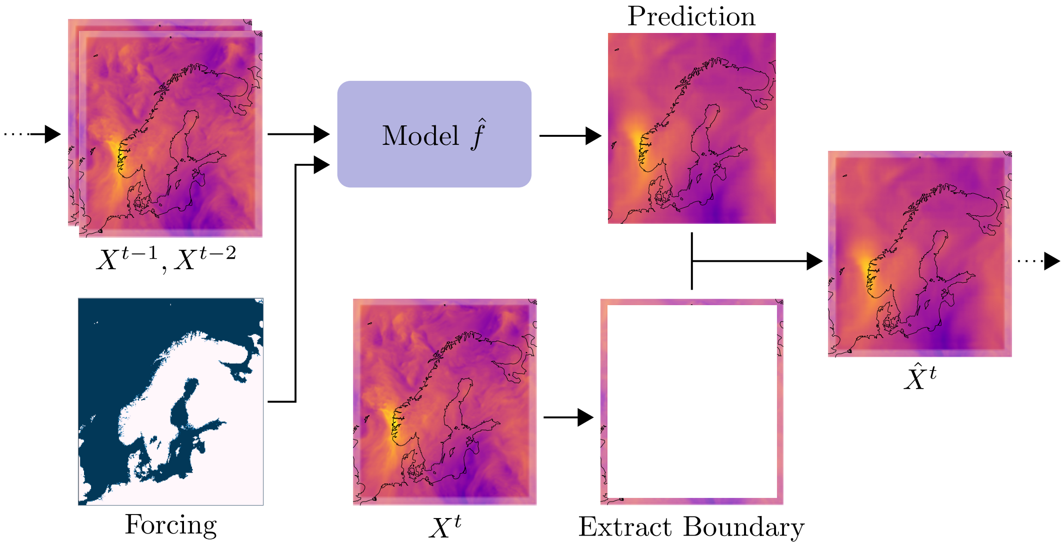


Figure 5: Schematic showing how boundary forcing is combined with the prediction during each autoregressive step.

has a natural interpretation as aggregating all of this information for the final prediction. We update the node representations according to

$$\tilde{\boldsymbol{e}}_{s\to r}^{[l+1\searrow l]} \leftarrow \mathrm{MLP}_{\mathrm{Dec.}}^{[l+1\searrow l]}\left(\left[\boldsymbol{e}_{s\to r}^{[l+1\searrow l]}, \boldsymbol{v}_s^{M[l+1]}, \boldsymbol{v}_r^{M[l]}\right]\right) \tag{10a}$$

$$\boldsymbol{v}_r^{M[l]} \leftarrow \boldsymbol{v}_r^{M[l]} + \mathrm{MLP}_{\mathrm{Dec.}}^{M[l]}\left(\left[\boldsymbol{v}_r^{M[l]}, \sum_{s:(s,r)\in\mathcal{E}^{[l+1\searrow l]}} \tilde{\boldsymbol{e}}_{s\to r}^{[l+1\searrow l]}\right]\right) \tag{10b}$$

sequentially for level $l = L-1, \dots, 1$. This is then followed by the final decoding to the grid

$$\tilde{\boldsymbol{e}}_{s\to r}^{\mathrm{M2G}} \leftarrow \mathrm{MLP}_E^{\mathrm{M2G}}\left(\left[\boldsymbol{e}_{s\to r}^{\mathrm{M2G}}, \boldsymbol{v}_s^{M[1]}, \boldsymbol{v}_r^{G}\right]\right) \tag{11a}$$

$$\boldsymbol{v}_r^{G} \leftarrow \boldsymbol{v}_r^{G} + \mathrm{MLP}_V^{\mathrm{M2G}}\left(\left[\boldsymbol{v}_r^{G}, \sum_{s:(s,r)\in\mathcal{E}^{\mathrm{M2G}}} \tilde{\boldsymbol{e}}_{s\to r}^{\mathrm{M2G}}\right]\right). \tag{11b}$$

Note that there is no need to update the edge representations during decoding, as they will not be used in any further processing. The final prediction step is the same as in GraphCast (eq. 5).

B.3 Boundary Forcing

In all models we use boundary forcing in order to include information about the surrounding area. The boundary forcing is included at each time step by replacing predictions inside of the boundary area $\mathbb{B}$ with the ground truth forecast X^t. After making the prediction as in eq. 5, we update the row for each node v as

$$\hat{X}_v^t \leftarrow \left(1 - \mathbb{I}_{\{v\in\mathbb{B}\}}\right)\hat{X}_v^t + \mathbb{I}_{\{v\in\mathbb{B}\}} X_v^t \tag{12}$$

where $\mathbb{I}_{\{\cdot\}}$ is the indicator function. This process is also described in fig. 5. Predictions for grid nodes within the boundary area are not used in the training loss or evaluation.

We define the boundary as the 10 closest grid nodes to the edge of the limited area. Note that we use a boundary area that lays *inside* the MEPS area, allowing us to use the ground truth forecasts as boundary forcing. In the operational system the boundary area is defined along the edge *outside* the MEPS area. There is no major conceptual difference between these and one could easily re-define the different areas to match the operational MEPS system.

B.4 Loss Function

All models are trained with a weighted Mean Squared Error (MSE) loss similar to GraphCast. We use batched training with the loss for each sample computed as

$$\mathcal{L} = \frac{1}{N_{\text{Rollout}}} \sum_{t=t_{\text{Init}}+1}^{t_{\text{Init}}+N_{\text{Rollout}}} \frac{1}{|\mathbb{G}|} \sum_{v \in \mathbb{G}} \sum_{i=1}^{S} \lambda_i \omega_i \left(\hat{X}_{v,i}^t - X_{v,i}^t \right)^2 \tag{13}$$

where

- t_{Init} is the initial time step the forecast starts from,
- N_{Rollout} is the number of rollout steps used during training,
- $\mathbb{G} = \{1, \ldots, N\} \setminus \mathbb{B}$ is the set of grid nodes not on the boundary,
- λ_i is the inverse variance of time differences for variable i,
- ω_i is a weight associated with the vertical level of variable i.

The sums in the loss are (in order) over 1) forecast time steps, 2) grid nodes and 3) weather state variables. We refer to Lam et al. [5] for details about the variable weighting scheme. Note that as the MEPS area has grid cells of approximately equal size we do not need to weight the grid nodes.

B.5 Training Details

Our trained models use latent representations of dimensionality 64 everywhere. The GC-LAM and 1L-LAM models use 4 processing layers, and for Hi-LAM we use only 2. With the sweep down and up through the hierarchy each processing layer of Hi-LAM updates node and edge representations twice. To keep a fair comparison we thus use half as many processing layers. With these architectural choices we are able to train models with a batch size of 8 using a single NVIDIA A100 GPU with 80 GB VRAM. All models were trained for single-step prediction for 500 epochs and then fine-tuned using 4 time-step rollouts for 200 epochs. The AdamW optimizer [37] was used with a learning rate of 0.001. Training each model takes 3–4 days in total. The Hi-LAM model, despite being larger and more sequential in nature, only takes 12% longer to train than GC-LAM. Preliminary experiments showed only minor improvements when fine-tuning on longer rollouts, even when considering the error at the highest lead times.

C Data Details

In this appendix we give more details on the exact features and dataset used in our experiments. While some of the forcing and static features follow the exact format of GraphCast [5], we here describe everything for the sake of completeness.

C.1 Atmospheric Variables

At each grid cell we model 17 weather variables, including a broad range of different quantities and different vertical levels in the atmosphere. All variables are described in table 1. The particular choice of variables was motivated by a combination of meteorological relevance, data availability and striving for a diverse set of variables to evaluate the model on. For solar radiation (`nlwrs` and `nswrs`) we consider the net flux at ground level, aggregated over the past 3 hours (since the last time step). Apart from the solar radiation all other quantities are instantaneous. For training we rescale the values of each variable to zero mean and unit variance.

C.2 Forcing

As forcing at each time point we include at each grid node:

- The solar radiation flux at the top of the atmosphere.
- $(\sin(2\pi\tau/\mathcal{T}) + 1)/2$ and $(\cos(2\pi\tau/\mathcal{T}) + 1)/2$, where τ is the time of day and $\mathcal{T}$ the length of one day.

Table 1: The 17 weather variables in our dataset. [1]In the MEPS system 65 vertical levels are defined from the ground to the top of the atmosphere [7]. The lowest MEPS level sits at approximately 12.5 m above ground.

Abbreviation	Quantity	Vertical Level	Unit	Description
`pres_0g`	Pressure	Ground level	Pa	Atmospheric pressure at ground level
`pres_0s`	Pressure	Sea level	Pa	Atmospheric pressure at sea level
`nlwrs`	Solar radiation	Ground level	W/m^2	Net longwave radiation flux
`nswrs`	Solar radiation	Ground level	W/m^2	Net shortwave radiation flux
`r_2`	Humidity	2 m above ground	-	Relative humidity, in $[0, 1]$
`r_65`	Humidity	Lowest MEPS level[1]	-	Relative humidity, in $[0, 1]$
`t_2`	Temperature	2 m above ground	K	Instantaneous temperature
`t_65`	Temperature	Lowest MEPS level[1]	K	Instantaneous temperature
`t_500`	Temperature	500 hPa pressure	K	Instantaneous temperature
`t_850`	Temperature	850 hPa pressure	K	Instantaneous temperature
`u_65`	Wind	Lowest MEPS level[1]	m/s	u-component of wind
`v_65`	Wind	Lowest MEPS level[1]	m/s	v-component of wind
`u_850`	Wind	850 hPa pressure	m/s	u-component of wind
`v_850`	Wind	850 hPa pressure	m/s	v-component of wind
`wvint`	Water vapor	All	kg/m^2	Integrated column of water vapor above grid cell
`z_500`	Geopotential	500 hPa pressure	m^2/s^2	Instantaneous geopotential
`z_1000`	Geopotential	1000 hPa pressure	m^2/s^2	Instantaneous geopotential

- The same sine and cosine features as above, but with τ as the time in the current year and $\mathcal{T}$ as the length of the year.
- The fraction of open water in the grid cell. We assume this to be constant over the forecast period and take the value from the time of the initial state. In GraphCast the open water mask is assumed to be fully static. Treating this as forcing could be useful for taking into account seasonal fluctuations of the ice cover in the Nordic region.

As in GraphCast we include the forcing for time steps $t-1$, t and $t+1$ as input at time step t, for making the prediction $\hat{X}^{t+1}$. The boundary forcing applied at each time step (as described in appendix B.3) consists of the same 17 weather variables as listed in table 1.

C.3 Static Features

Grid Nodes The static features used at each grid node are:

- The 2-dimensional coordinate of the node in the MEPS Lambert projection, normalized by the maximum coordinate value.
- The surface geopotential (topography).
- A binary indicator describing if the node is in the boundary area or not.

Mesh Nodes The only static features associated with mesh nodes are their coordinates, following the same format as for the grid nodes.

Edges All edges use the same type of static features. This includes

- The length of the edge.
- The vector difference between the source and target nodes, using the MEPS Lambert projection coordinates.

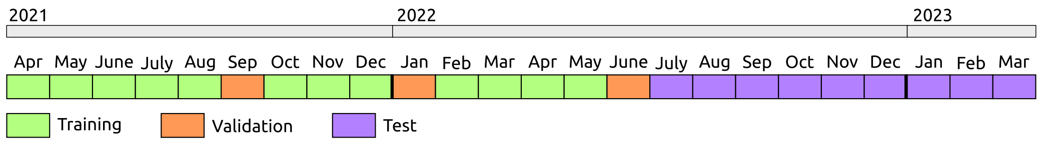


Figure 6: Overview of the period covered in the dataset and the training/validation/test split.

Both of these features are normalized by the length of the longest edge.

C.4 Dataset Details

Our dataset consists of archived forecasts from the operational MEPS system during the period April 2021 – March 2023. This period was chosen due to the system configuration being reasonably stable, preventing distributional shifts within the dataset. From the chosen period we extract the forecasts started at 00 and 12 UTC each day. At each starting time there are 5 ensemble forecasts, resulting in 10 forecasts per day (if no ensemble members are missing due to operational issues). When retrieving the data we additionally downsample the spatial resolution from the original 2.5 km to 10 km. This results in a dataset of 6069 forecasts of length 66 h with 1 h time steps. We split these into training, validation and test sets according to fig. 6. It should be noted that there is high correlation between many of these 6069 samples, in particular at early time steps in the forecasts. The actual information content in the data, in terms of the effective sample size, is substantially lower.

The original data uses 1 h time steps, but our NeurWP models predict in 3 h steps. Because of this we can extract 3 training samples from each forecast in the dataset (i.e. original time steps $\{1, 4, 7, \ldots\}$, $\{2, 5, 8, \ldots\}$ and $\{3, 6, 9, \ldots\}$). As we train on only ≤ 4 rollout steps, we also only need a subset of each such time series for each training iteration. To make maximum use of the data during training we randomly sample which time step to use as initial state for unrolling the model.

The combination of 1) the 3 h time steps, 2) using the last two weather states as model inputs, 3) including forcing from multiple times as input, means that we reduce the effective length of our ground truth forecasts in pre-processing. This explains why we predict for 57 h rather than the original 66 h. Note however that nothing prevents us from unrolling the models to create forecasts for 66 h or beyond. Although this is possible, we do not have ground truth data to compare against past 57 h and therefore view such experiments to be of limited interest. There is still no reason to expect the model performance to become drastically worse specifically past 57 h, as all models are anyhow fine-tuned only on 4 time step rollouts (= 12 h $\ll$ 57 h).

D Graph Construction

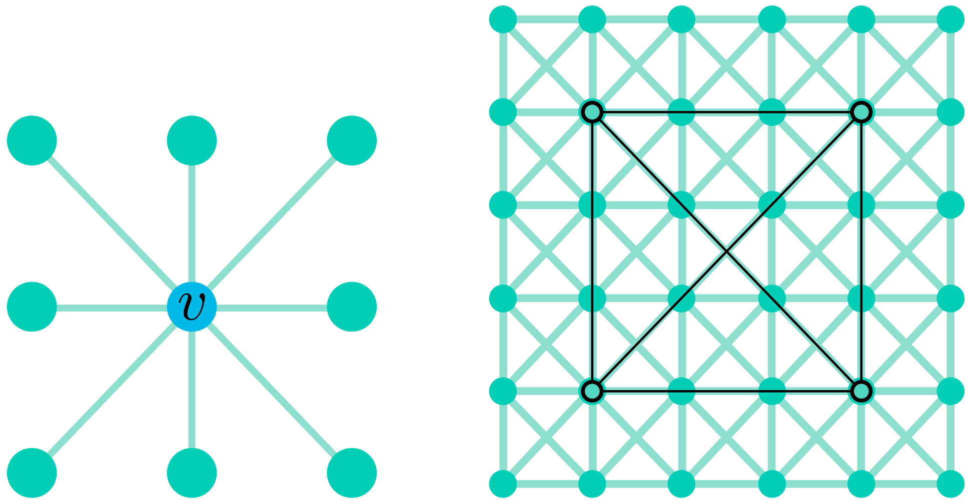
We here give additional details on how the different graphs were constructed for the MEPS area.

D.1 Mesh Graph

Our mesh graphs contain regular quadrilateral meshes covering the MEPS area. To create these we lay out mesh nodes in regular rows and columns over the area. Each node is then connected with bidirectional edges to its neighbors horizontally, vertically and diagonally (see fig. 7a). This results in all nodes (except those at the edge of the area) having 8 neighbors. The procedure is repeated at 4 different resolutions (fig. 8a), tripling the distance between nodes at each resolution. This means that a node at resolution level l is positioned at the center of a group of 3×3 nodes at resolution level $l - 1$, sharing its exact position with the center node of the group (illustrated in fig. 7b).

To create the multi-scale mesh graph for GC-LAM we merge the graphs at different resolutions, combining any nodes that sit at the same coordinates into one node. Note that this is possible due to how nodes align across the resolution levels. In the 1L-LAM model only the one graph of finest resolution is used, and the 3 remaining levels disregarded.

For the Hi-LAM model the graphs of different resolution are not merged, but used as the different levels of the hierarchy. Additional inter-level edge sets $\mathcal{E}^{[l \nearrow l+1]}$ and $\mathcal{E}^{[l+1 \searrow l]}$ are then created for all adjacent levels. Each set $\mathcal{E}^{[l \nearrow l+1]}$ of upwards edges is created by connecting each node on level l



(a) Each mesh node v is connected to its neighbors horizontally, vertically and diagonally.

(b) Alignment of mesh nodes **O** at resolution level l with mesh nodes ● at resolution level $l-1$.

Figure 7: Illustration of the mesh graph construction process.

Table 2: Number of nodes and edges in the different graphs. We count bidirectional edges twice, as these result in two messages being passed in GNN layers (one in each direction). Grid2Mesh and Mesh2Grid describe only the edge sets, since these connect the existing grid and mesh nodes.

	Nodes	**Edges**
Hi-LAM mesh graph	7380	72 358
GC-LAM mesh graph	6561	57 616
1L-LAM mesh graph	6561	51 520
Grid2Mesh	–	100 656
Mesh2Grid	–	255 136

with the closest node on level $l+1$. This means that each node at levels $l > 1$ will have 9 incoming edges from the level below. The downward edges $\mathcal{E}^{[l+1 \searrow l]}$ are a copy of $\mathcal{E}^{[l \nearrow l+1]}$ with the direction of each edge flipped. The mesh graphs used in the different models are visualized in fig. 8 and the number of nodes and edges in each graph listed in table 2.

D.2 Connecting the Grid and Mesh

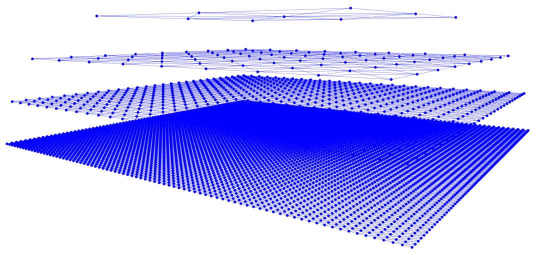
The grid nodes are connected to the mesh (only the bottom level in Hi-LAM) through the edge sets $\mathcal{E}^{\text{G2M}}$ and $\mathcal{E}^{\text{M2G}}$. To form $\mathcal{E}^{\text{G2M}}$, each grid node is connected to all mesh nodes closer than $0.67 d_m$, where d_m is the distance between mesh nodes at the finest resolution. All distances are 2-dimensional euclidean, computed in the MEPS Lambert projection coordinates. The set $\mathcal{E}^{\text{M2G}}$ is constructed by iterating over the grid nodes, and at each creating edges from the 4 closest mesh nodes to the node in the grid.

E Additional Results

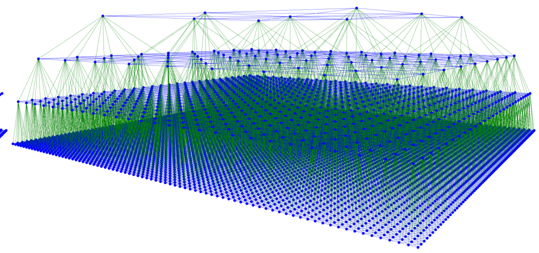
In this appendix we showcase additional results from our experiments with the MEPS data.

E.1 Per-Variable Error

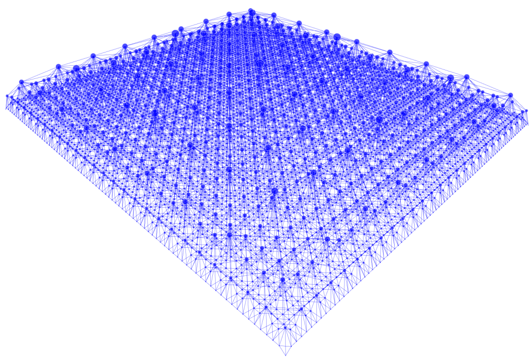
In the subplots of fig. 9 we show the forecasting error for all 17 variables at different lead times. Note that some variables show clear oscillations in the error, which can be explained by the day-night



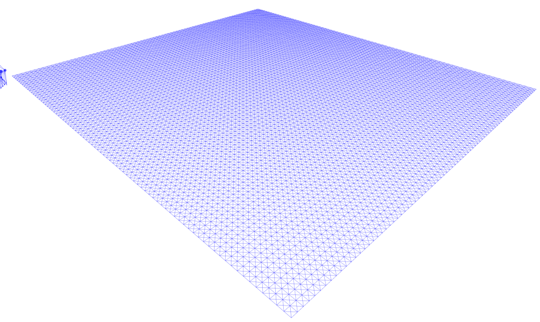
(a) Initial mesh graphs at 4 resolutions



(b) Hi-LAM mesh graph



(c) GC-LAM mesh graph



(d) 1L-LAM mesh graph

Figure 8: Mesh graphs used in the different models. Note that the vertical positioning of nodes is purely for visualization purposes. In the GC-LAM mesh graph the node size and vertical position is determined by its number of neighbors, to highlight the non-uniformity of the multi-scale structure.

cycle. As all forecasts in the dataset were started at either 00 or 12 UTC these cycles do not average out completely in our plots.

E.2 Example Forecasts

Example forecasts for all variables and models are shown in figs. 10 to 26, where we plot predictions at lead times 15, 33 and 57 h. Also the ground truth forecast from the MEPS system is included for comparison. The boundary area, where forcing is applied at each time step, is shown as a faded border in each plot.

E.3 Spatial Distribution of Error

Figure 27 shows the spatial distribution of the error, as measured by the mean test loss computed for each grid node. The errors in the figure come from the Hi-LAM model, but corresponding plots for the other models show similar patterns. As expected, the error increases towards the center of the area and away from the boundary. Information from the boundary forcing is highly useful for grid nodes close to the boundary. To utilize this information in the center of the area, the model has to simulate physical processes over multiple time steps, which is naturally a tougher problem. We also note that the topography has a large impact. Errors are generally higher in areas with large gradients, such as in the Scandinavian Mountains. As atmospheric processes are impacted by topological features this difficulty is inherent to the problem and not a unique issue of our NeurWP method [7, 15].

Appendix References

[32] Jessica Whitt and Scott Gordon. This is the economic cost of extreme weather. In *World Economic Forum Annual Meeting*, 2023. URL `https://www.weforum.org/agenda/2023/01/extreme-weather-economic-cost-wef23/`.

[33] Ralph Sims, Pedro Mercado, Wolfram Krewitt, Gouri Bhuyan, Damian Flynn, Hannele Holttinen, Gilberto Jannuzzi, Smail Khennas, Yongqian Liu, Lars J Nilsson, Joan Ogden, Kazuhiko

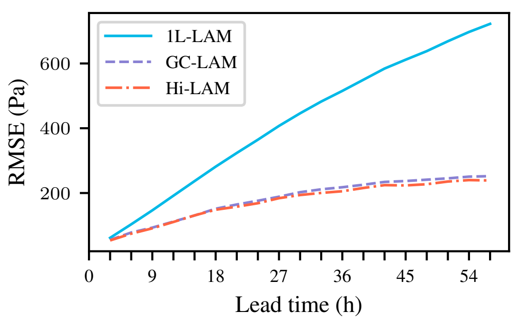
Ogimoto, Mark O'Malley, Hugh Outhred, Øystein Ulleberg, and Frans van Hulle. Integration of renewable energy into present and future energy systems. In *IPCC Special Report on Renewable Energy Sources and Climate Change Mitigation*. Cambridge University Press, 2012.

[34] European Commision. Destination earth, 2023. URL `https://digital-strategy.ec.europa.eu/en/policies/destination-earth`.

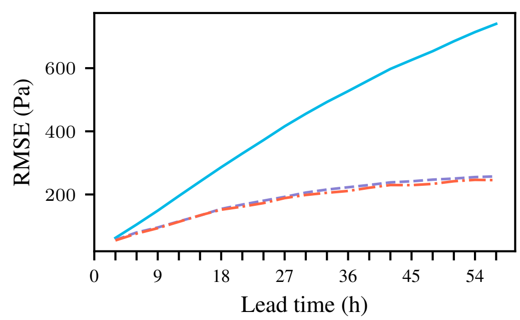
[35] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

[36] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

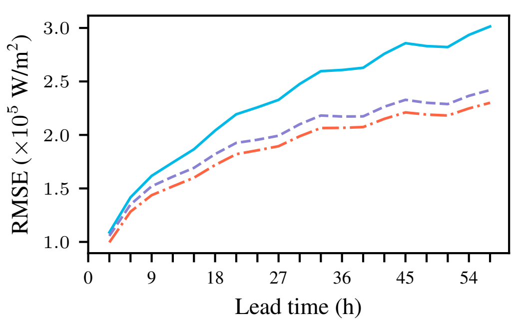
[37] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.



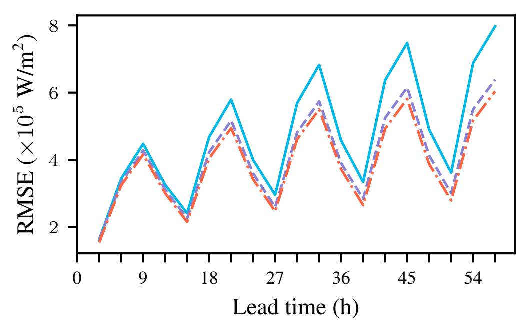
(a) `pres_0g`



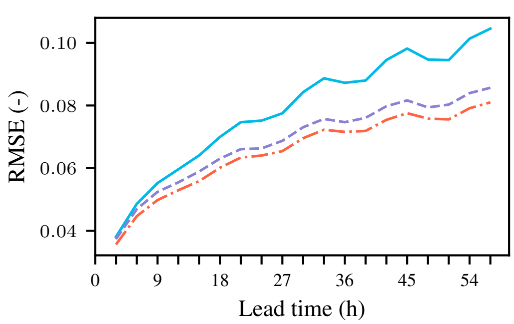
(b) `pres_0s`



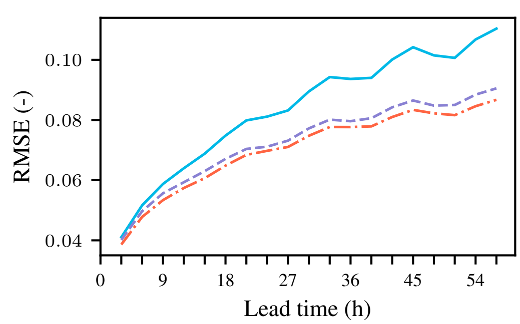
(c) `nlwrs`



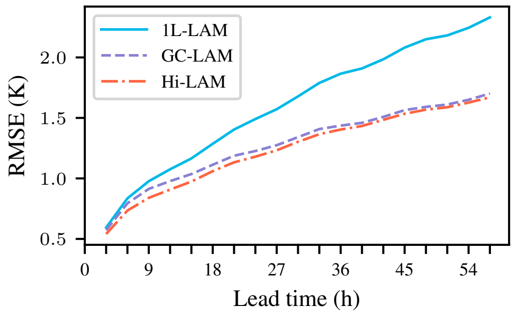
(d) `nswrs`



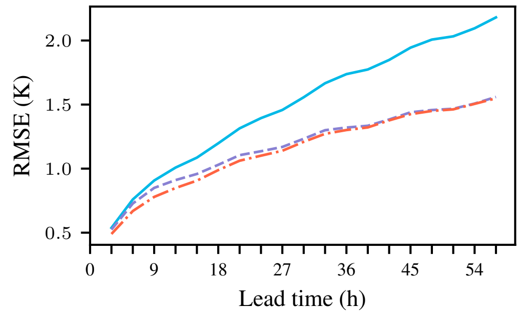
(e) `r_2`



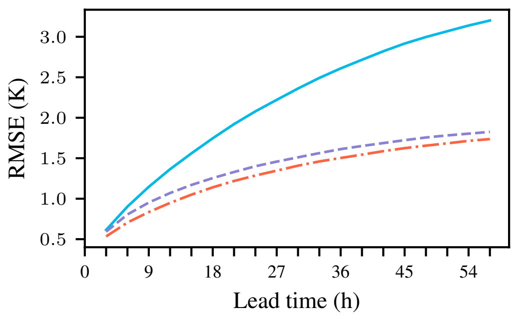
(f) `r_65`



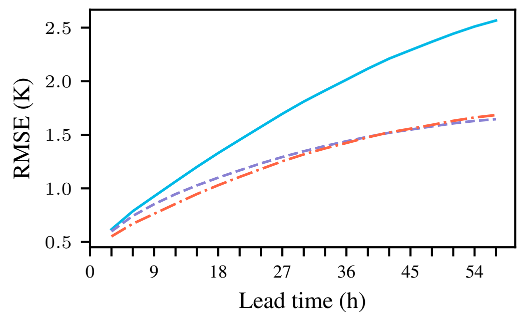
(g) `t_2`



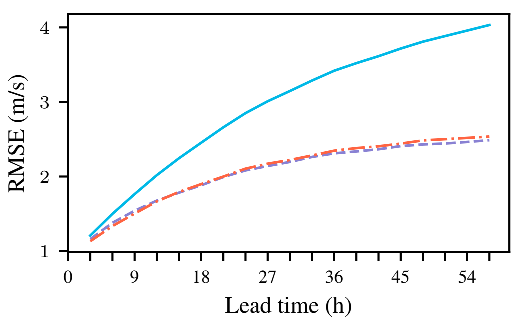
(h) `t_65`



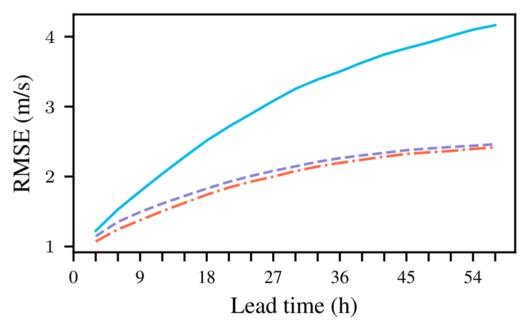
(i) `t_500`



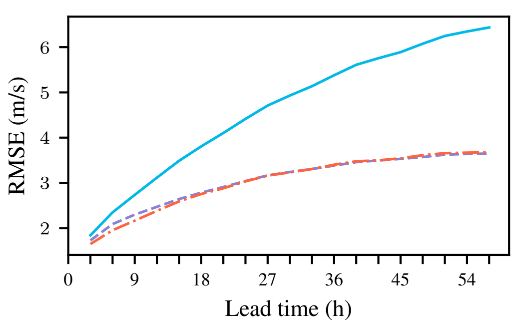
(j) `t_850`



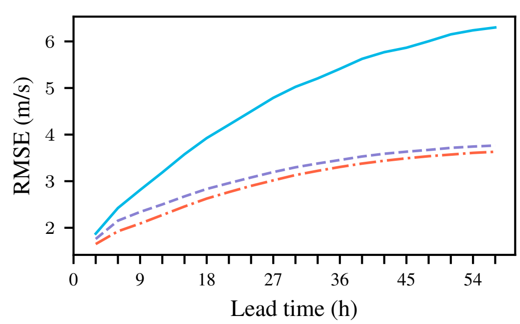
(k) `u_65`



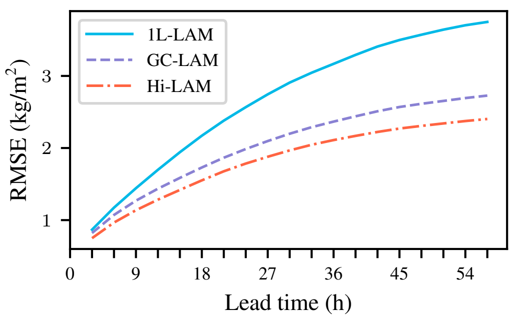
(l) `v_65`



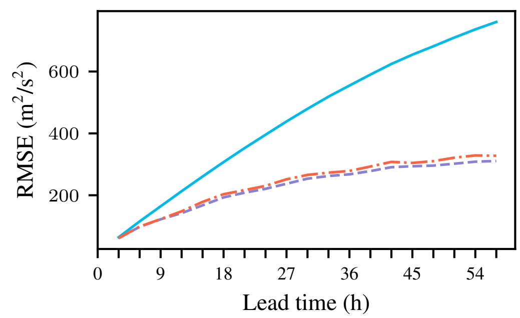
(m) `u_850`



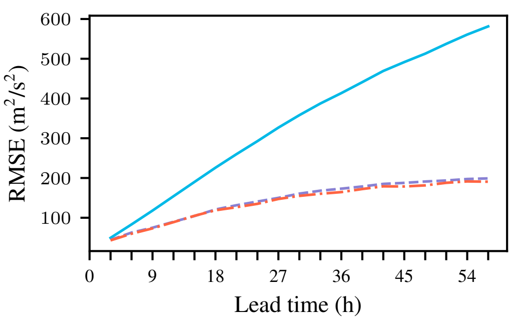
(n) `v_850`



(o) `wvint`



(p) `z_500`



(q) `z_1000`

Figure 9: RMSE for all models and variables, evaluated at each lead time up to 57 h.

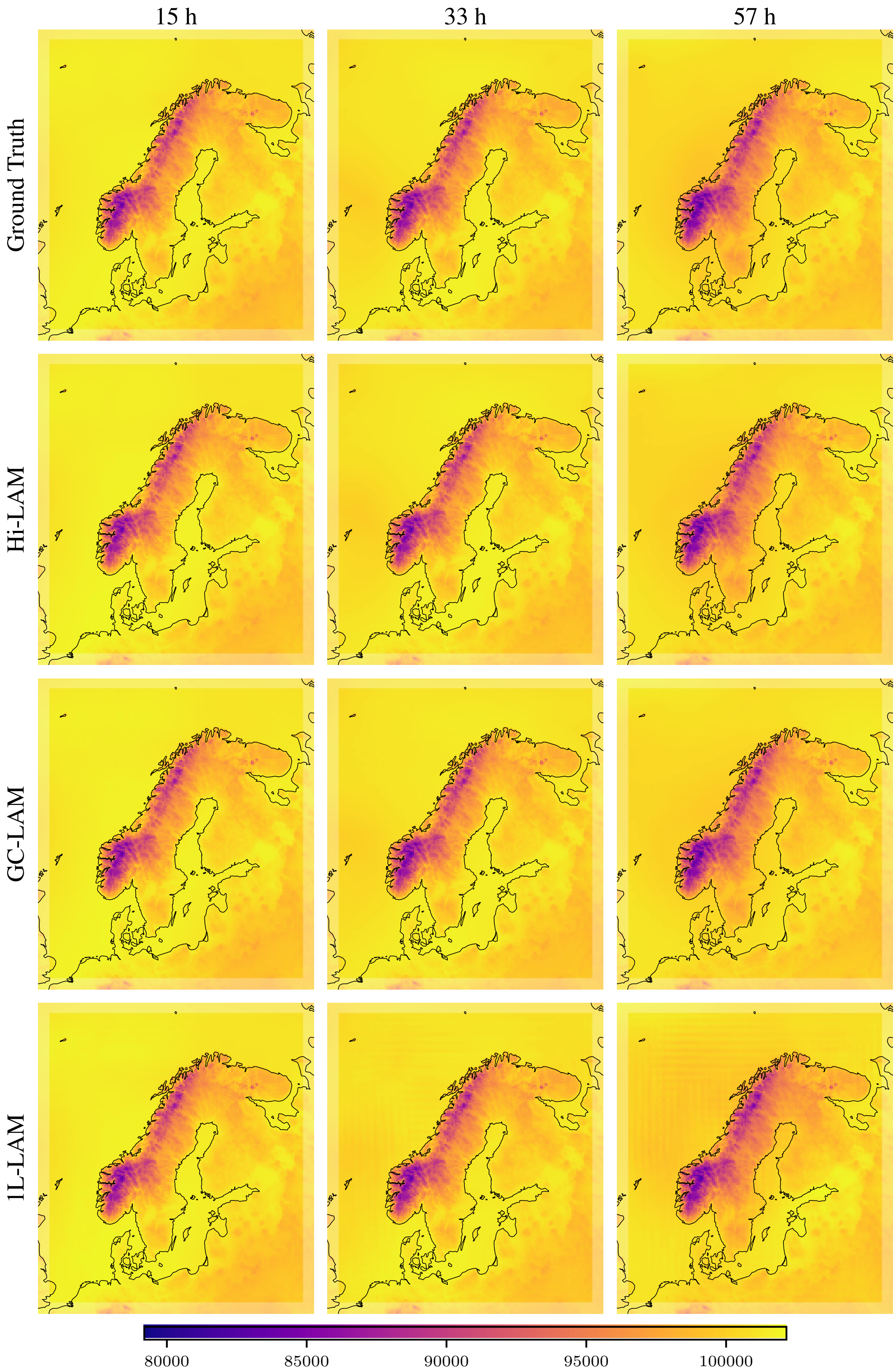


Figure 10: Example model forecasts for `pres_0g` at lead times 15, 33 and 57 h.

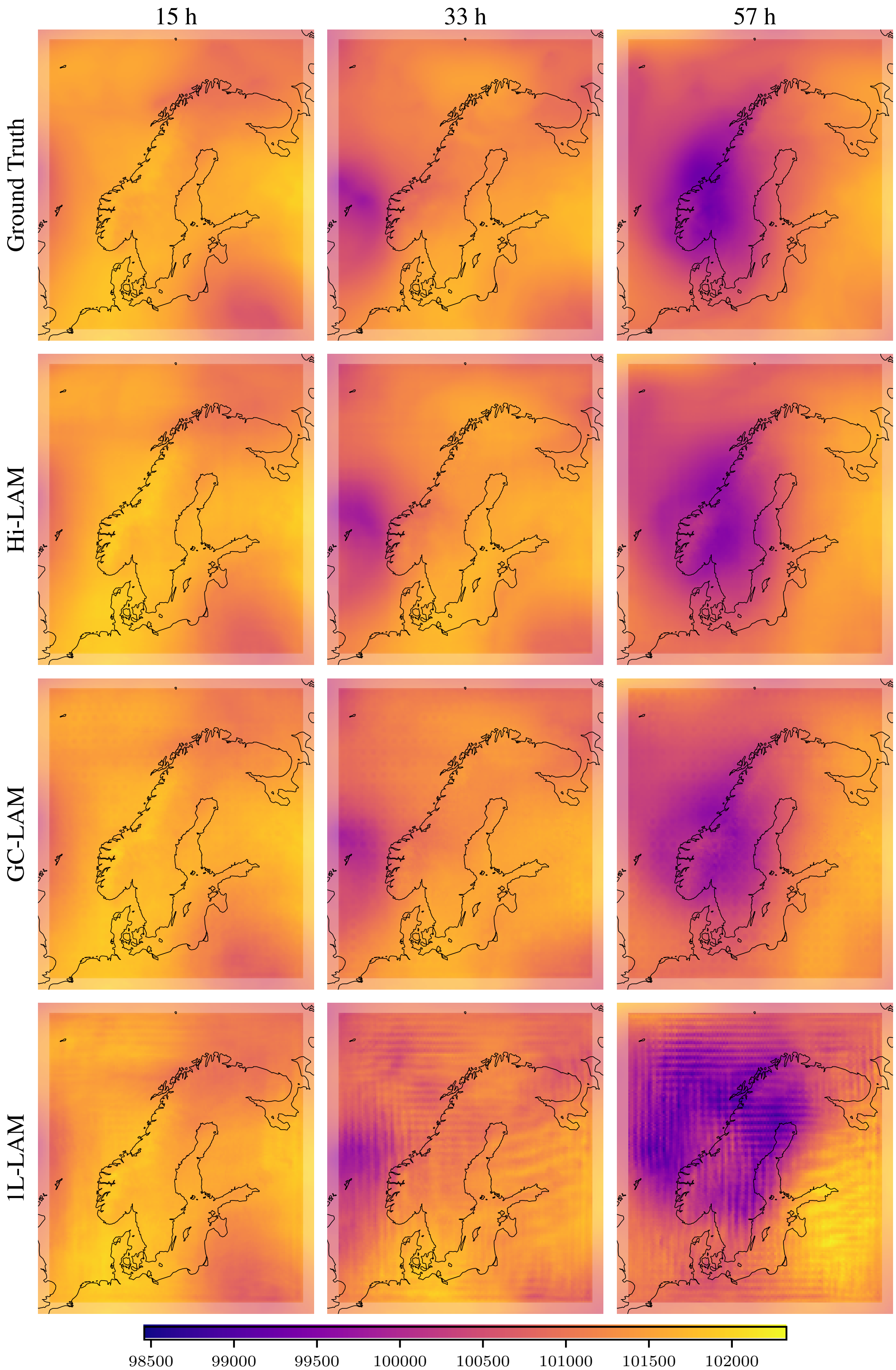


Figure 11: Example model forecasts for pres_0s at lead times 15, 33 and 57 h.

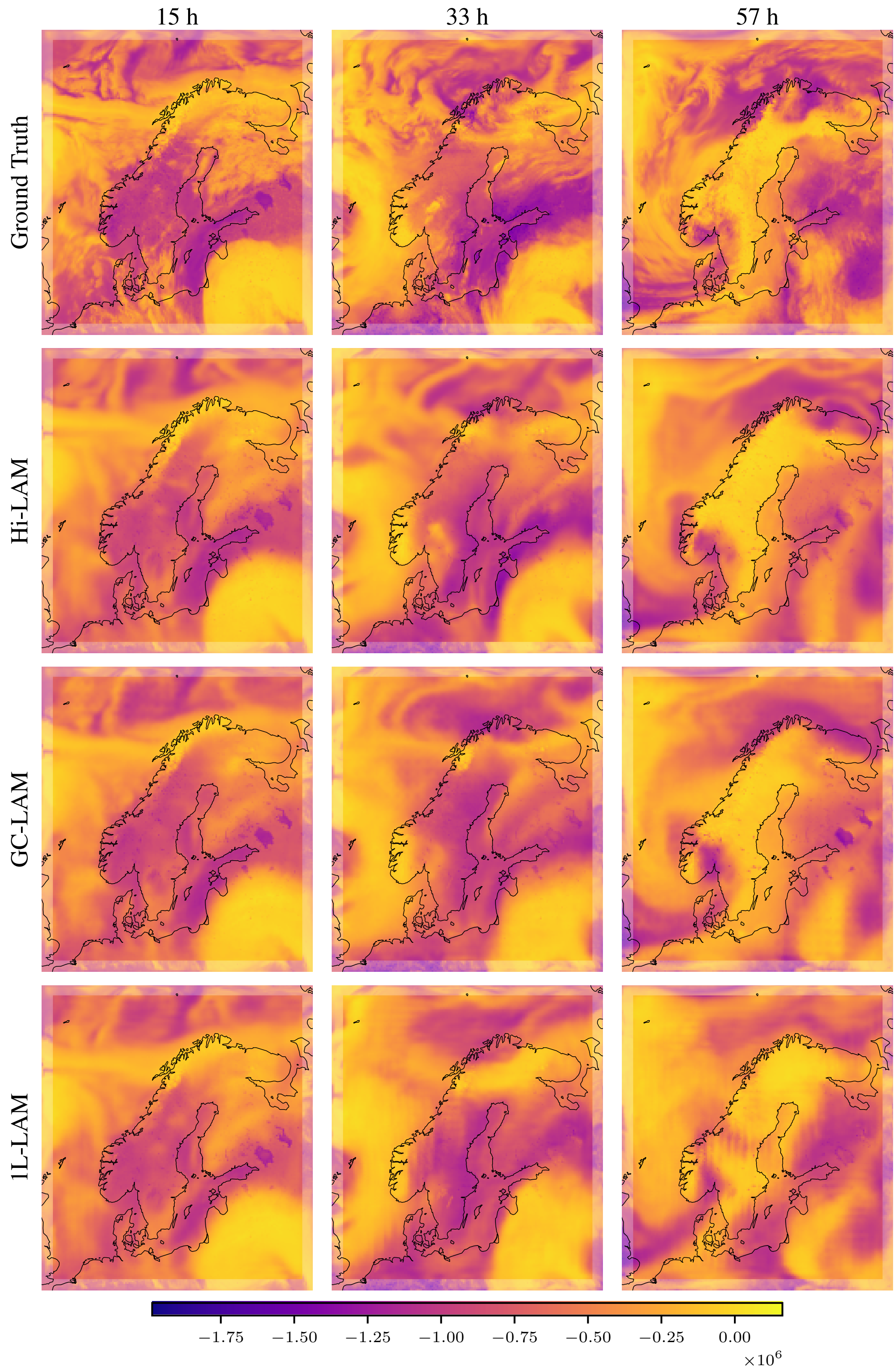


Figure 12: Example model forecasts for `nlwrs` at lead times 15, 33 and 57 h.

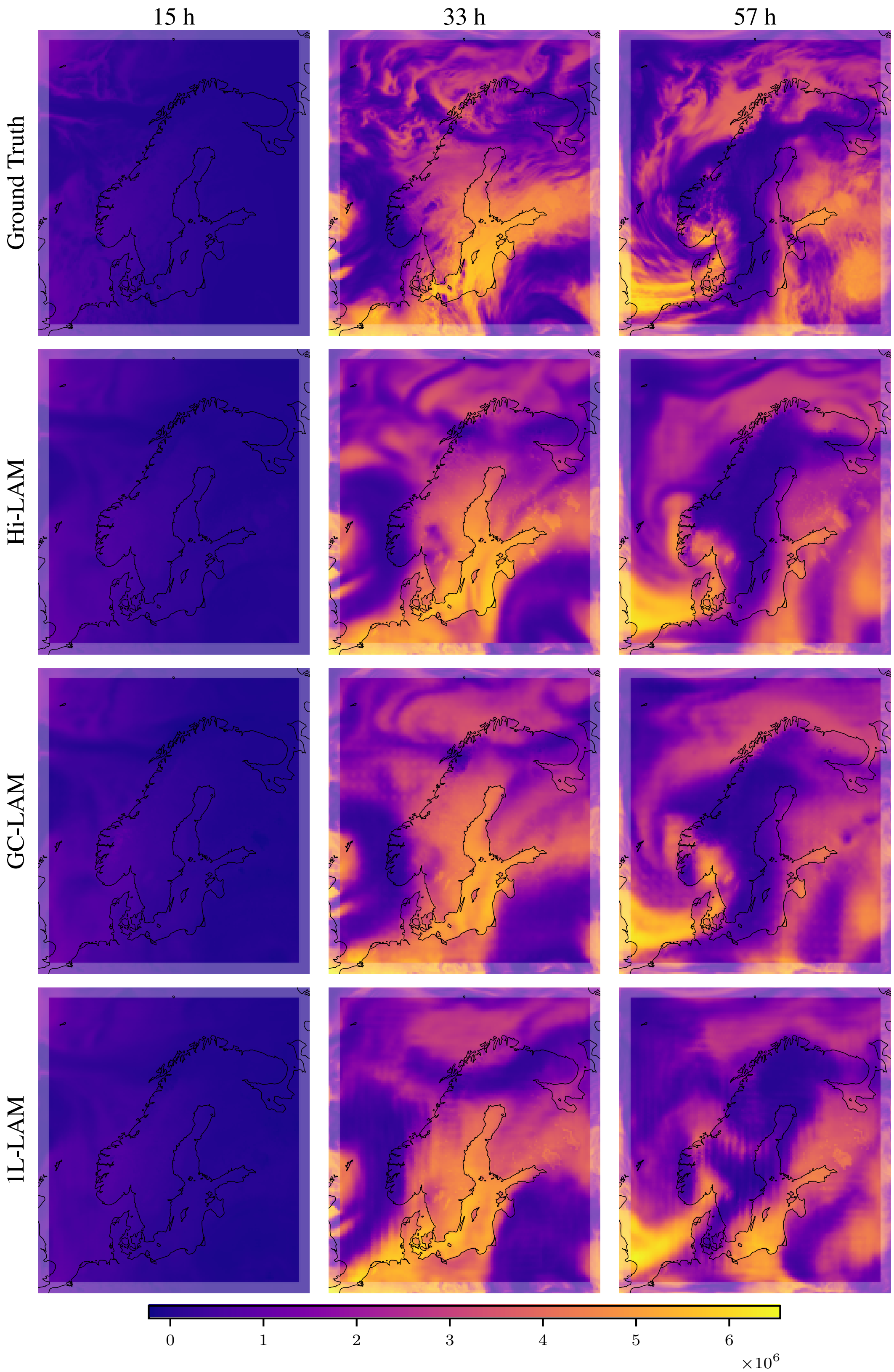


Figure 13: Example model forecasts for `nswrs` at lead times 15, 33 and 57 h.

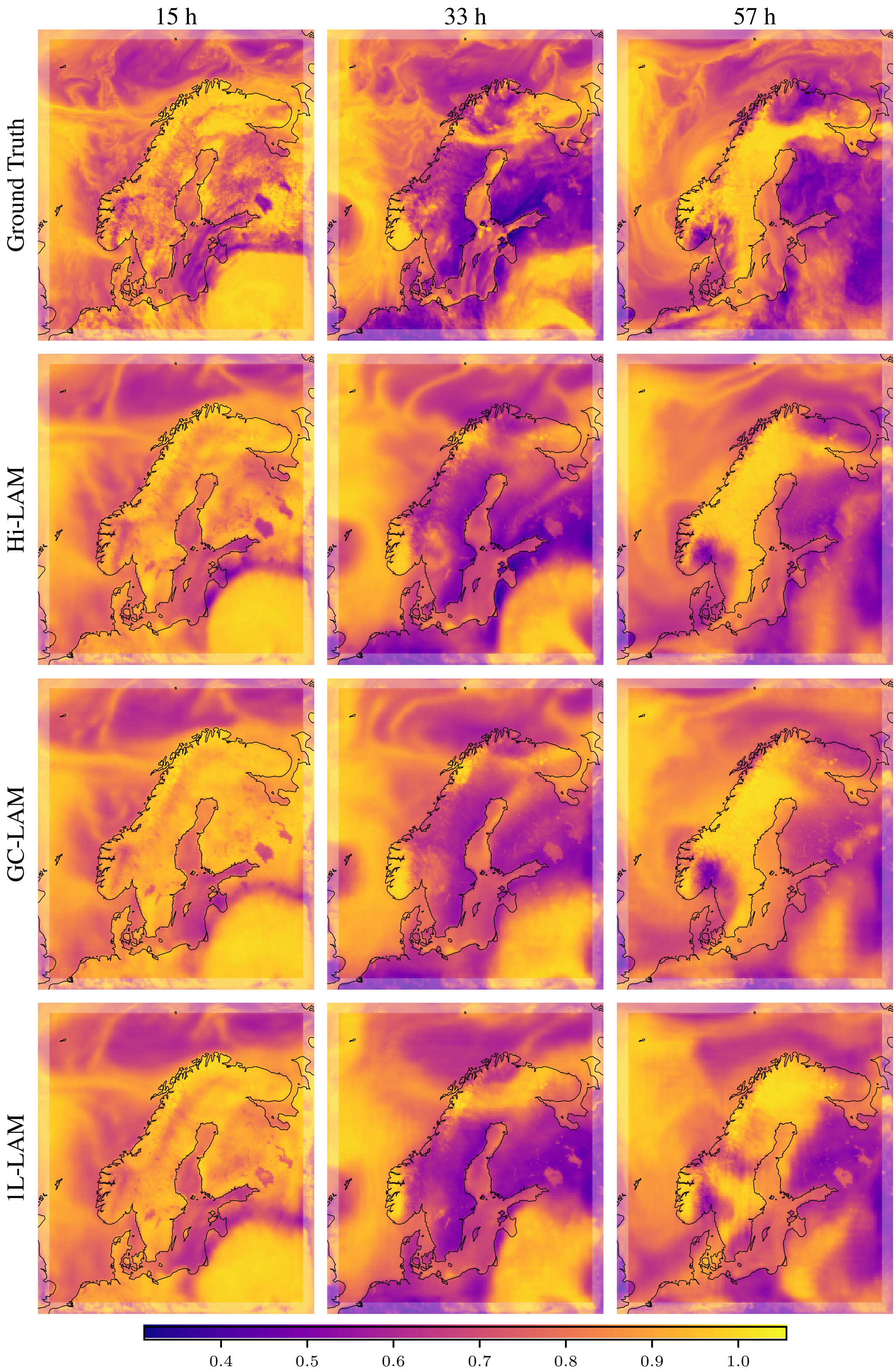


Figure 14: Example model forecasts for `r_2` at lead times 15, 33 and 57 h.

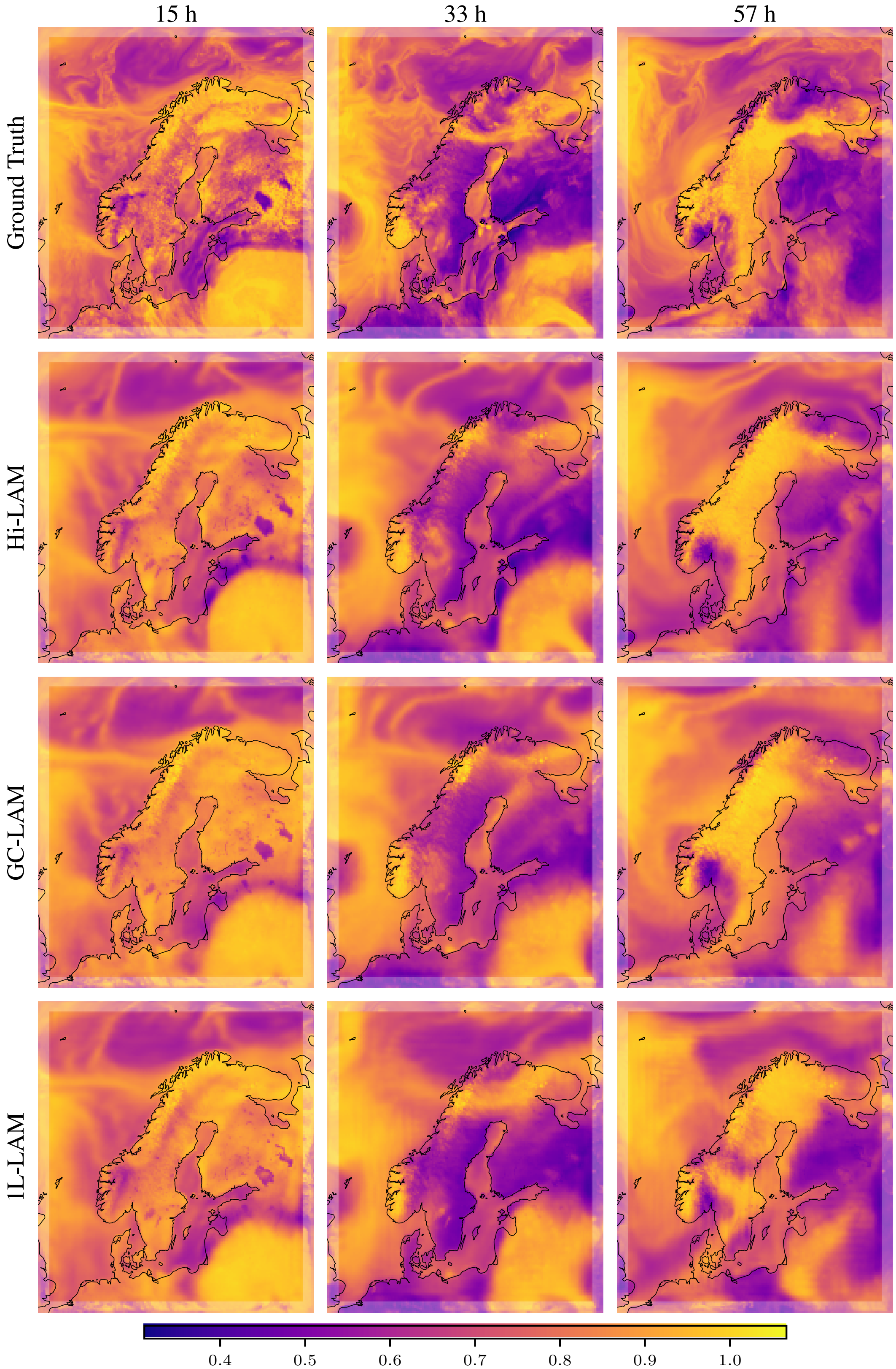


Figure 15: Example model forecasts for `r_65` at lead times 15, 33 and 57 h.

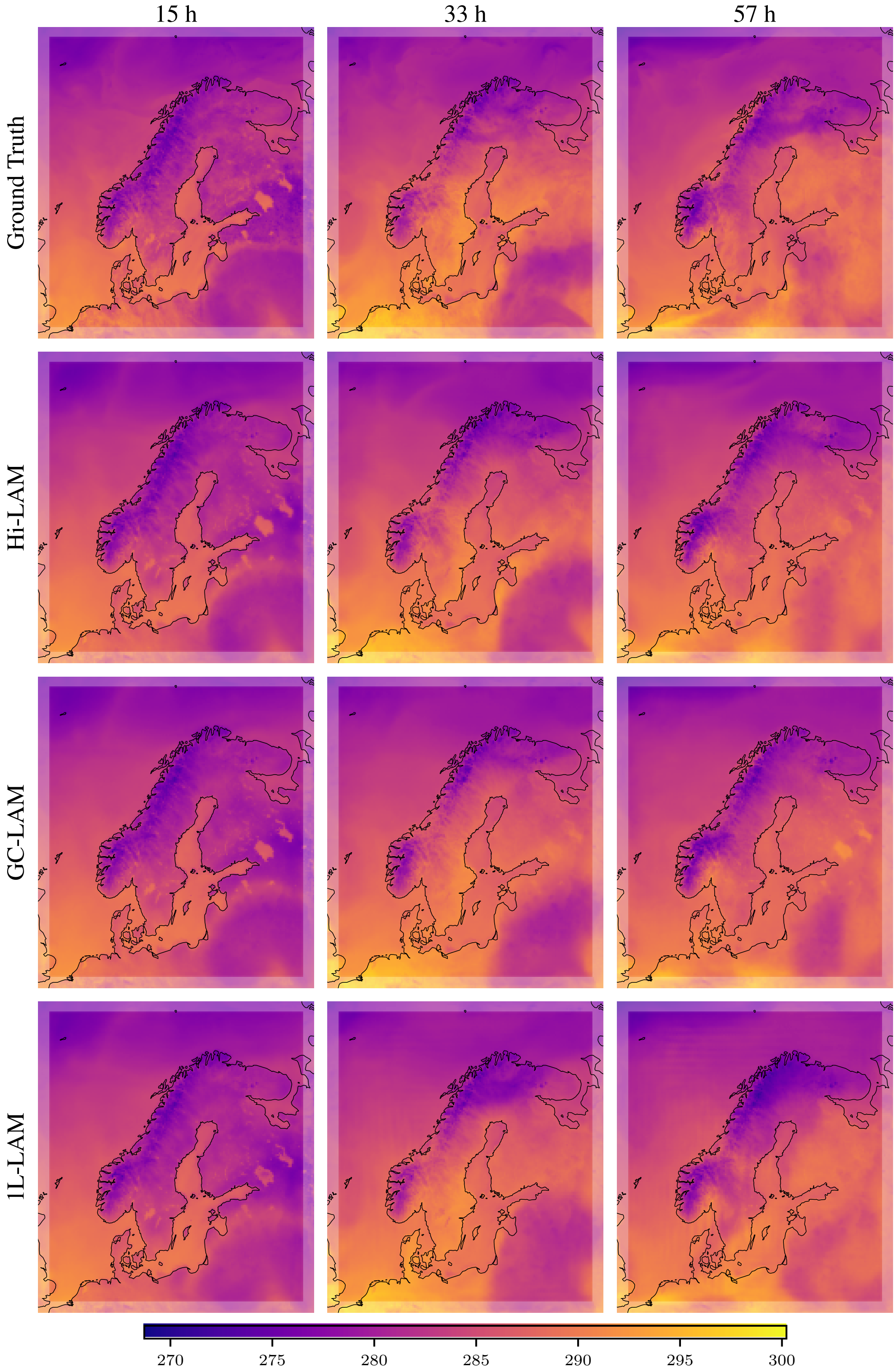


Figure 16: Example model forecasts for `t_2` at lead times 15, 33 and 57 h.

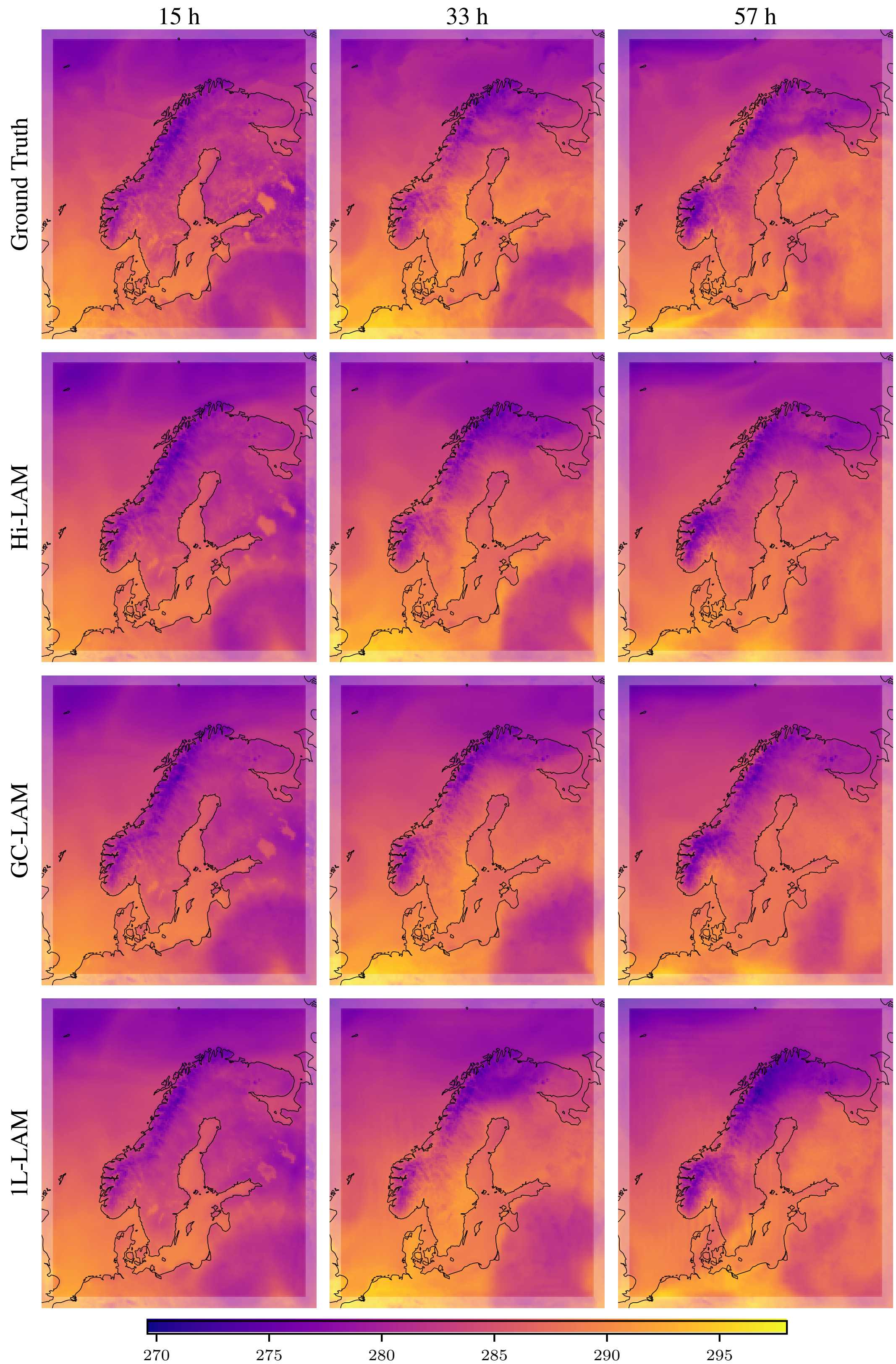


Figure 17: Example model forecasts for `t_65` at lead times 15, 33 and 57 h.

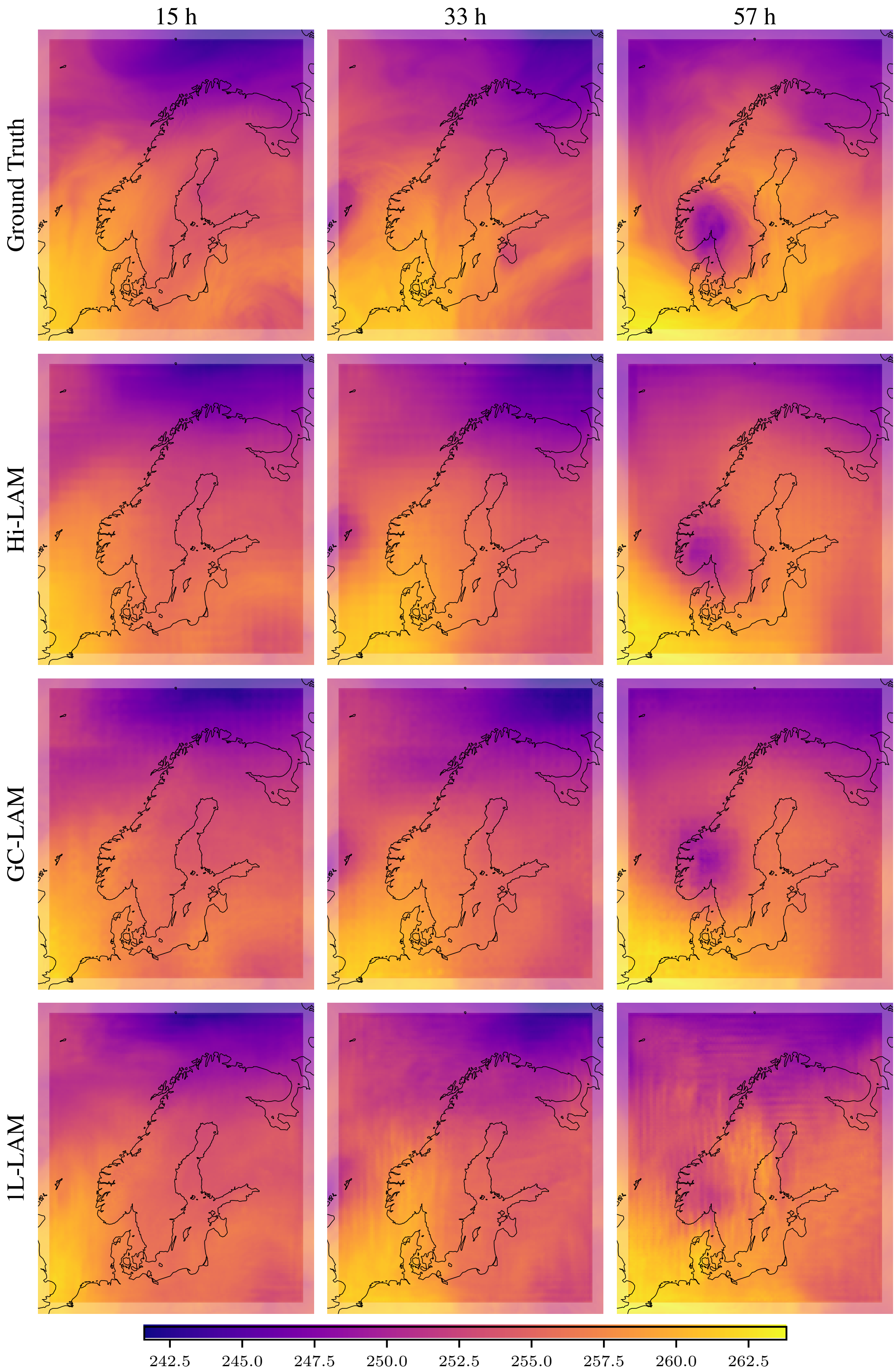


Figure 18: Example model forecasts for t_500 at lead times 15, 33 and 57 h.

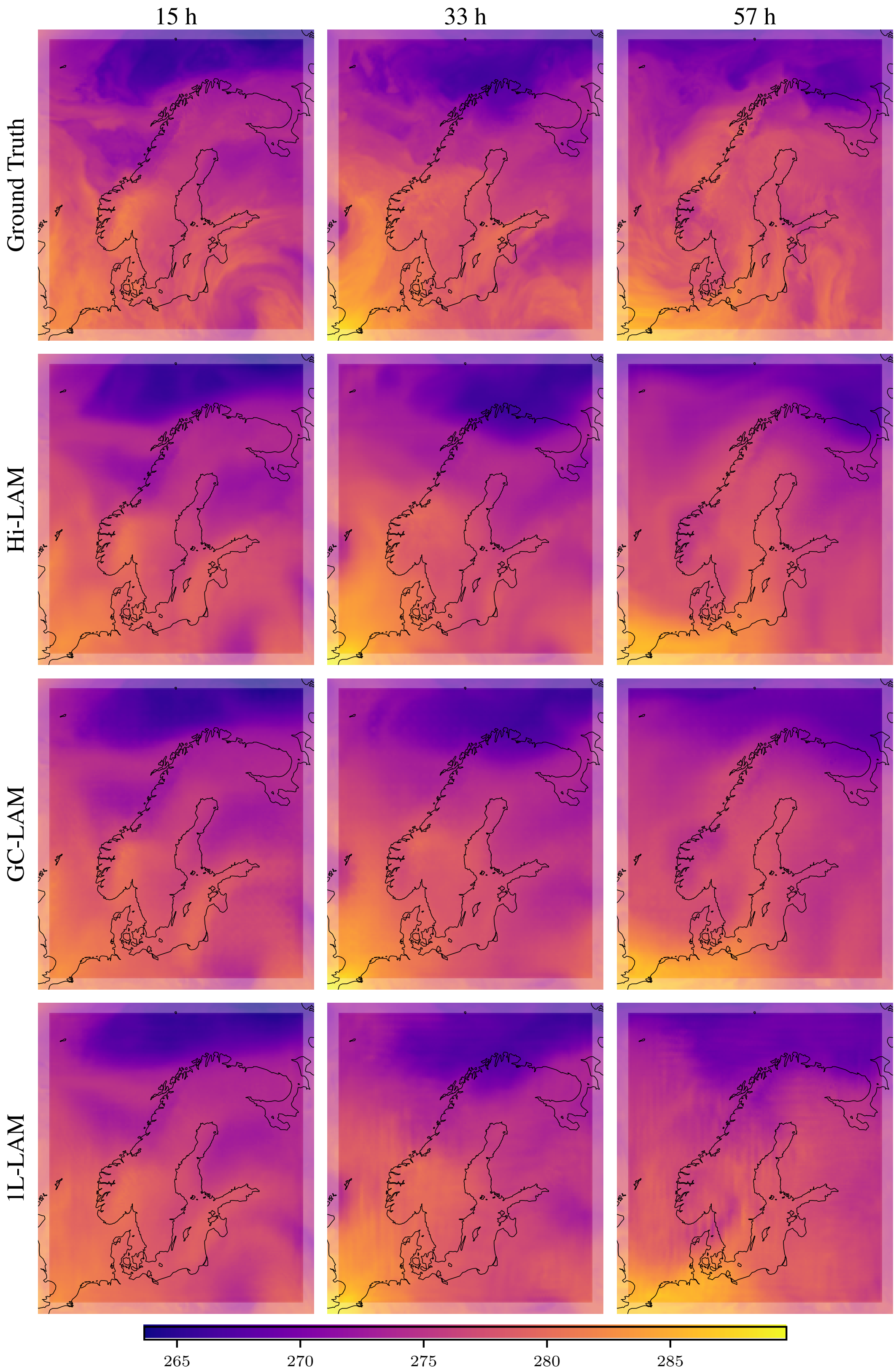


Figure 19: Example model forecasts for `t_850` at lead times 15, 33 and 57 h.

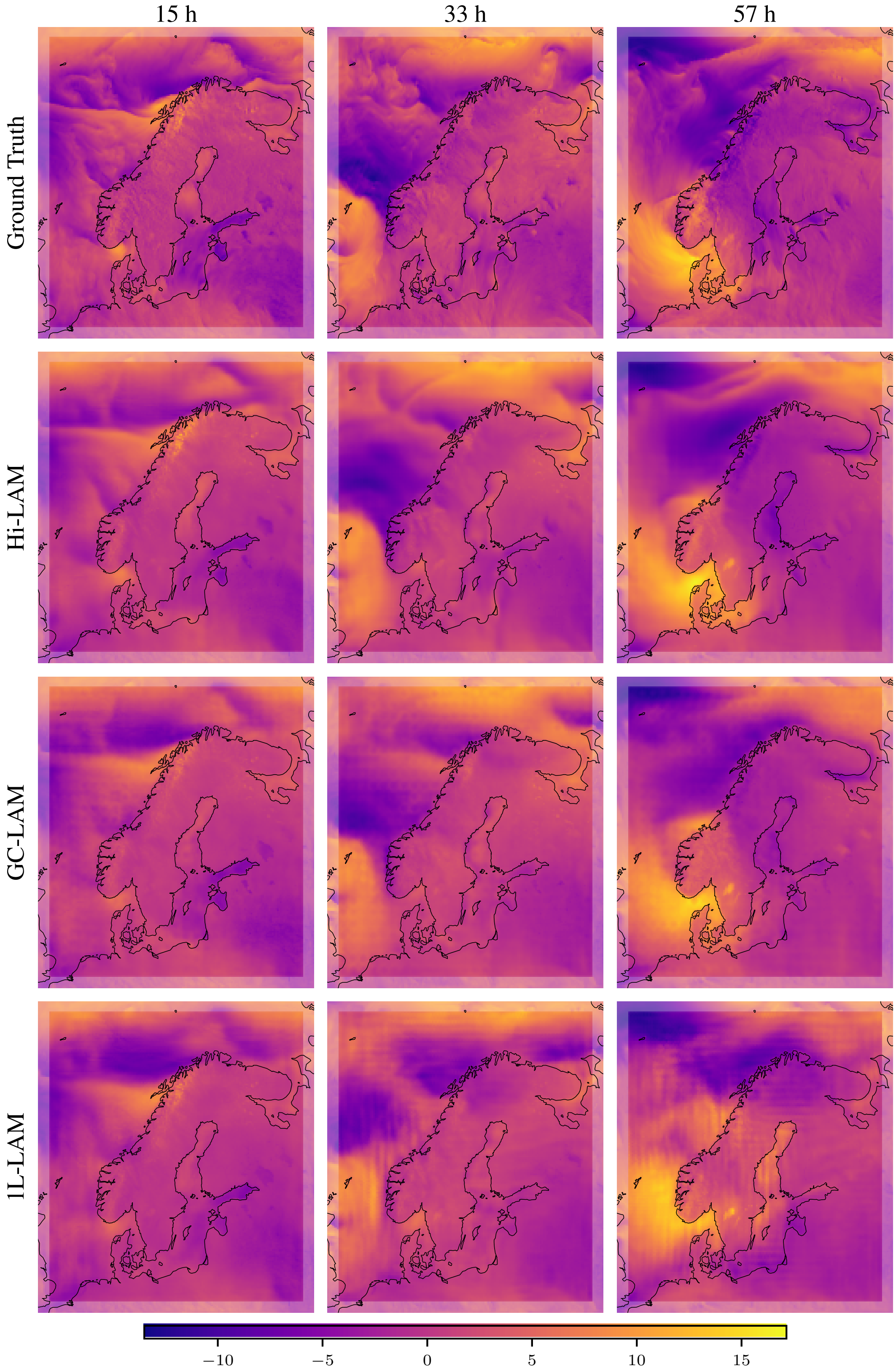


Figure 20: Example model forecasts for u_65 at lead times 15, 33 and 57 h.

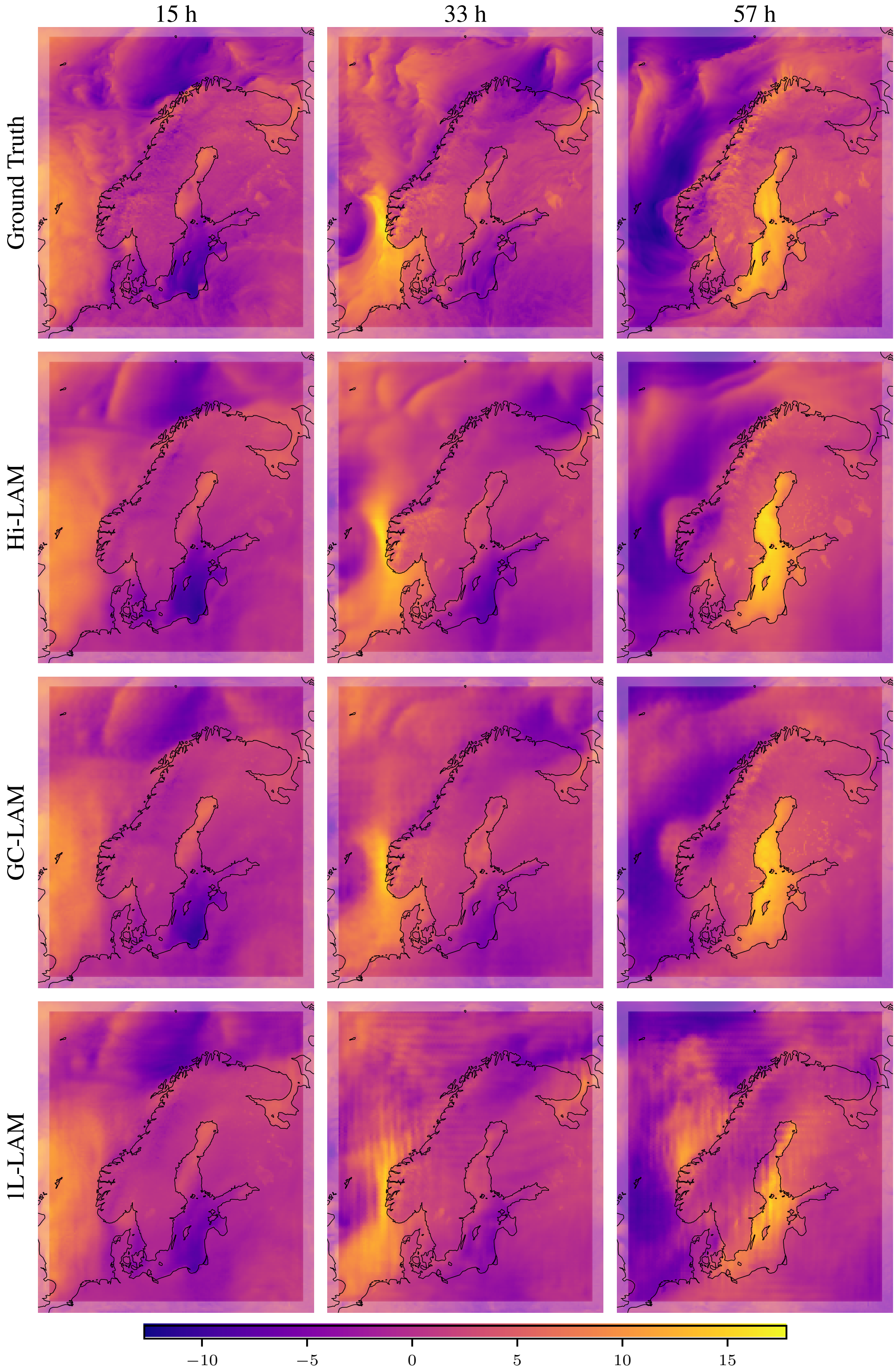


Figure 21: Example model forecasts for v_65 at lead times 15, 33 and 57 h.

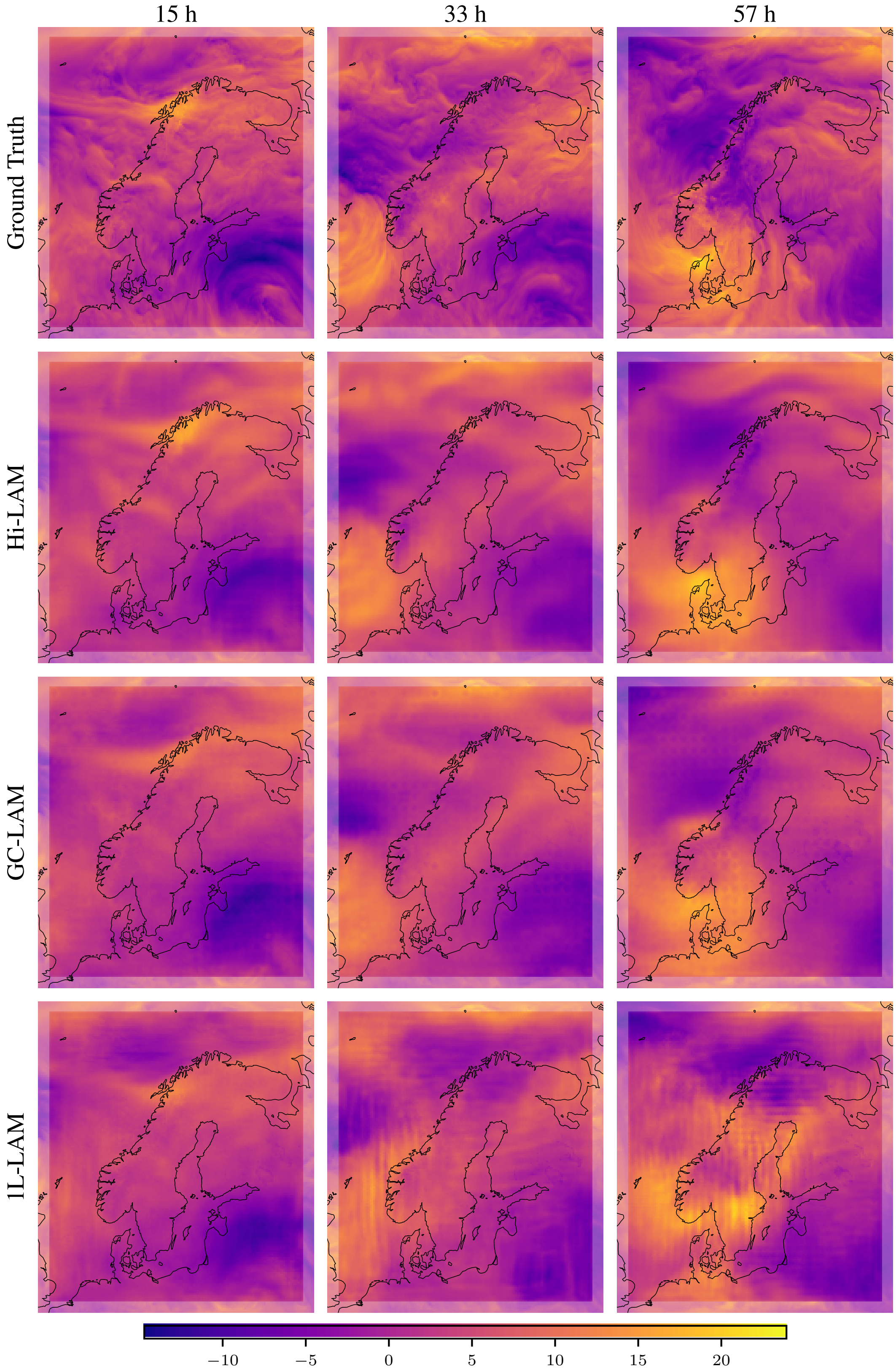


Figure 22: Example model forecasts for u_850 at lead times 15, 33 and 57 h.

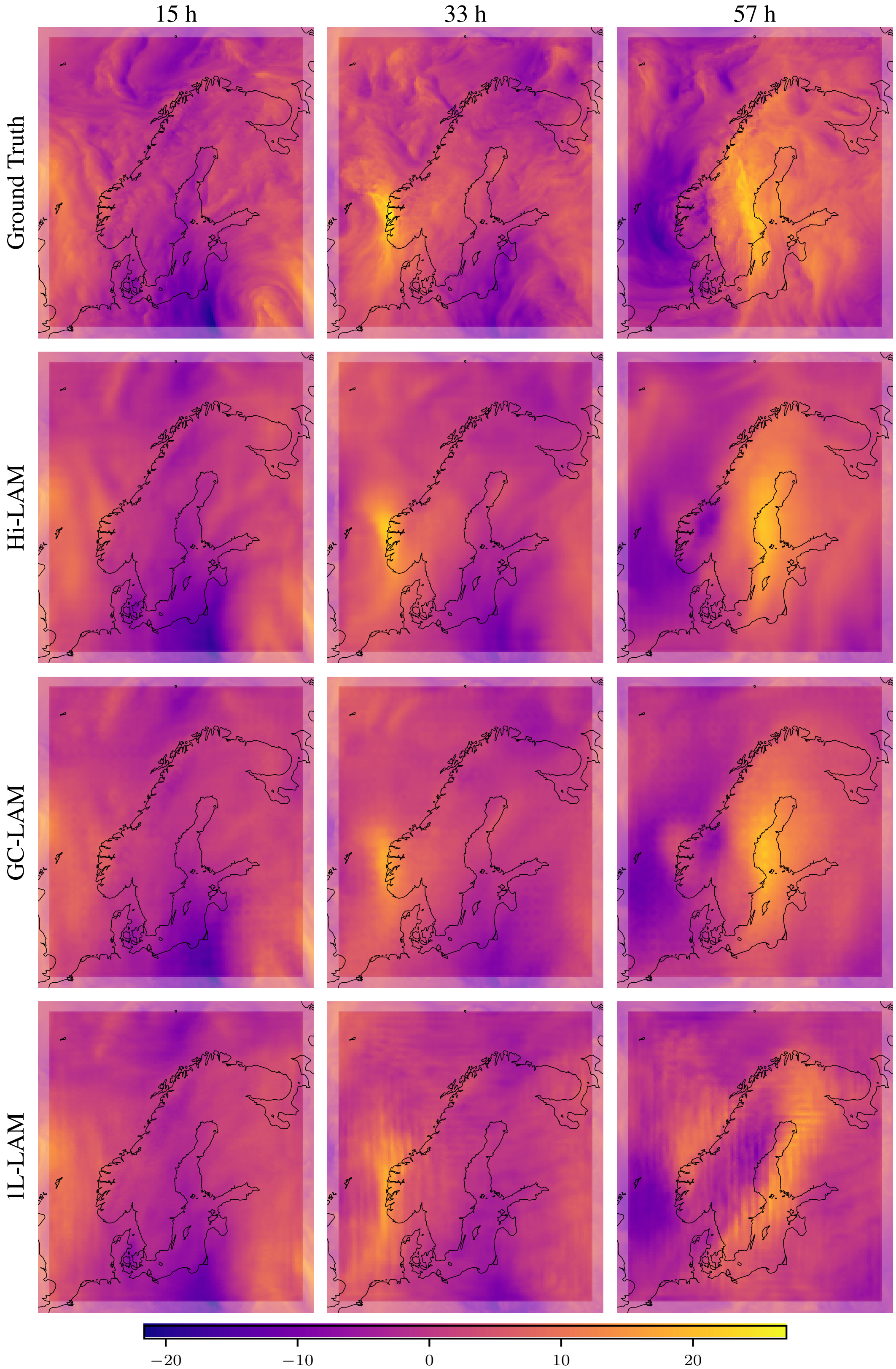


Figure 23: Example model forecasts for v_850 at lead times 15, 33 and 57 h.

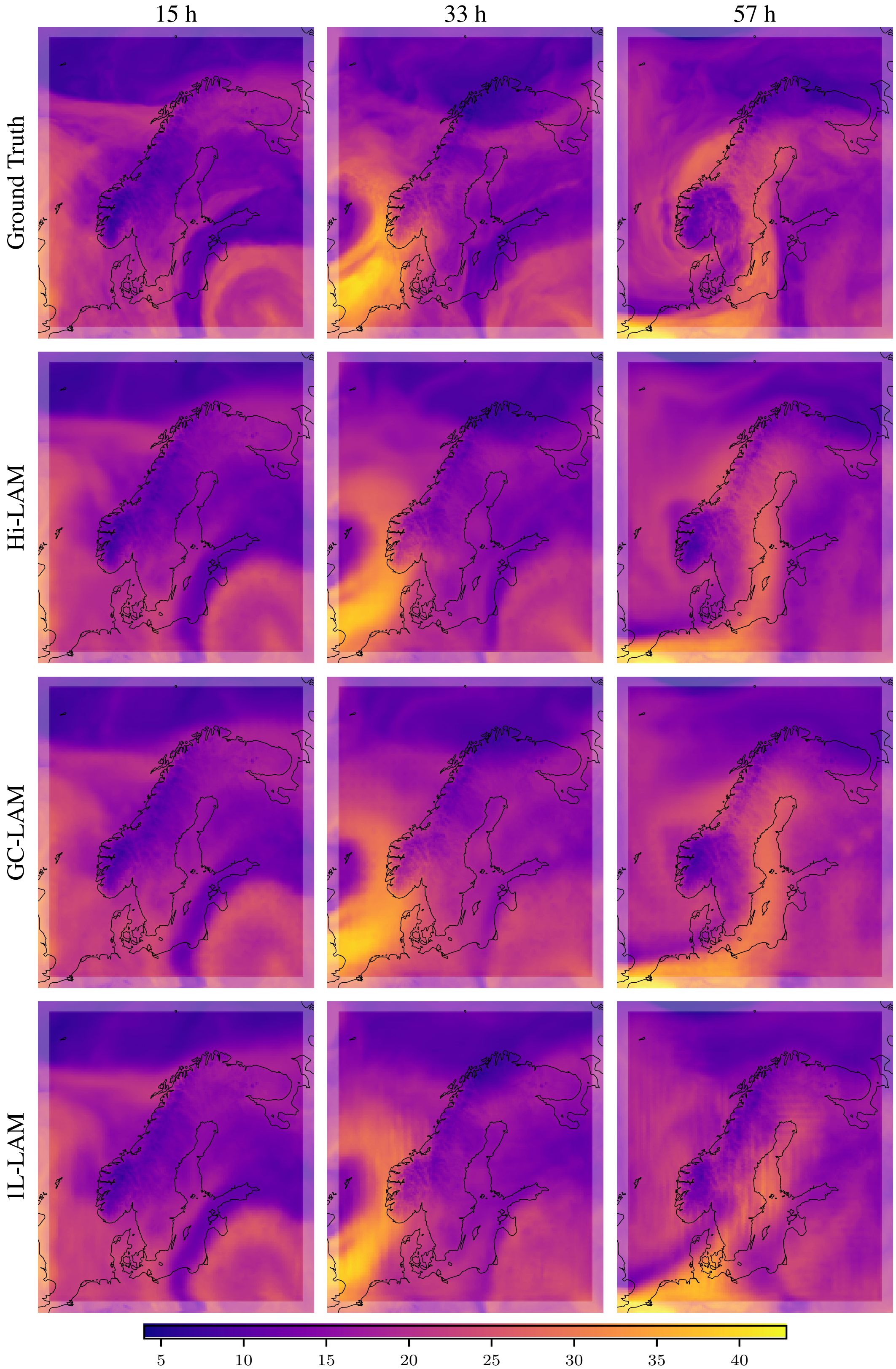


Figure 24: Example model forecasts for `wvint` at lead times 15, 33 and 57 h.

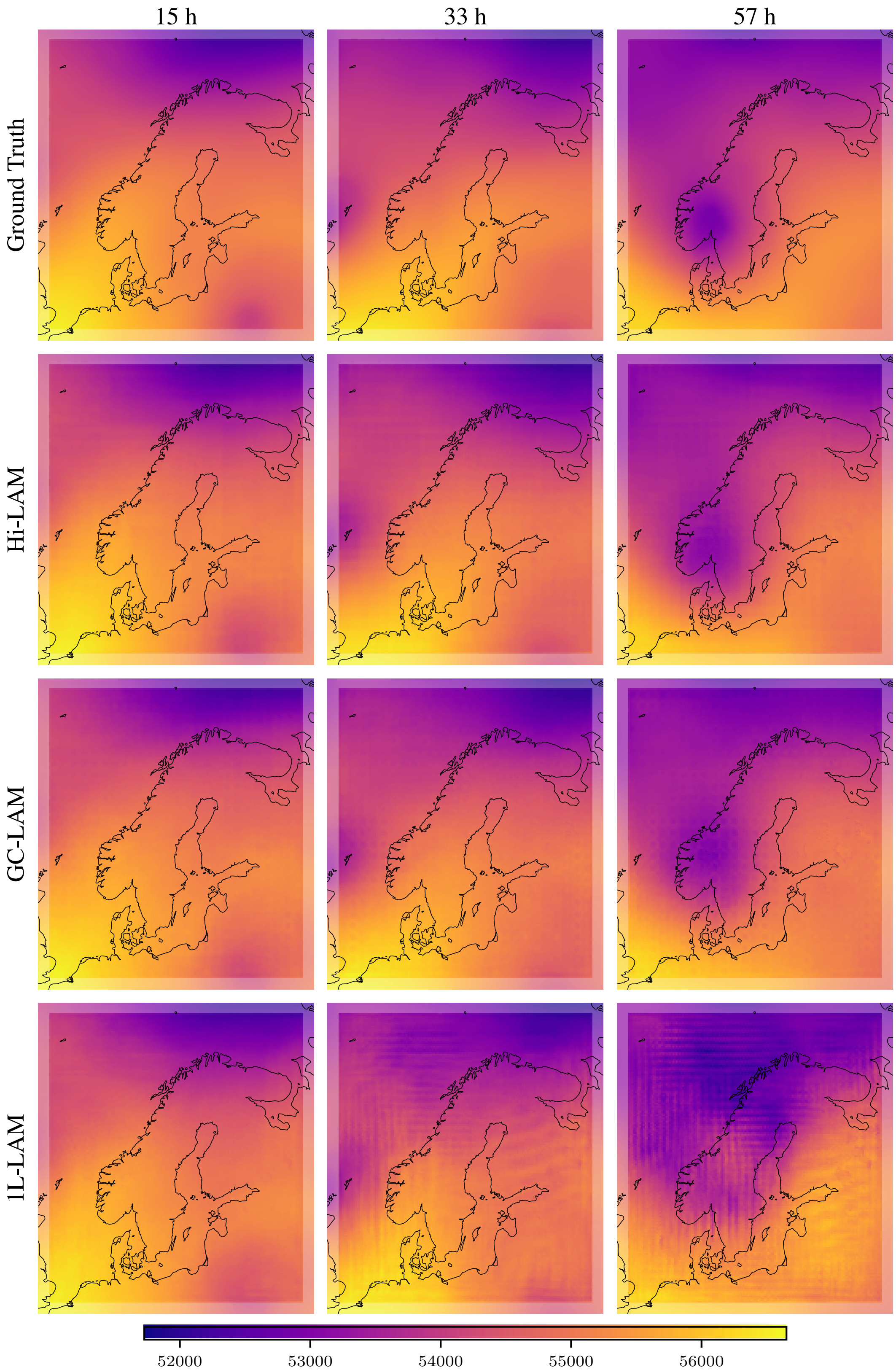


Figure 25: Example model forecasts for z_500 at lead times 15, 33 and 57 h.

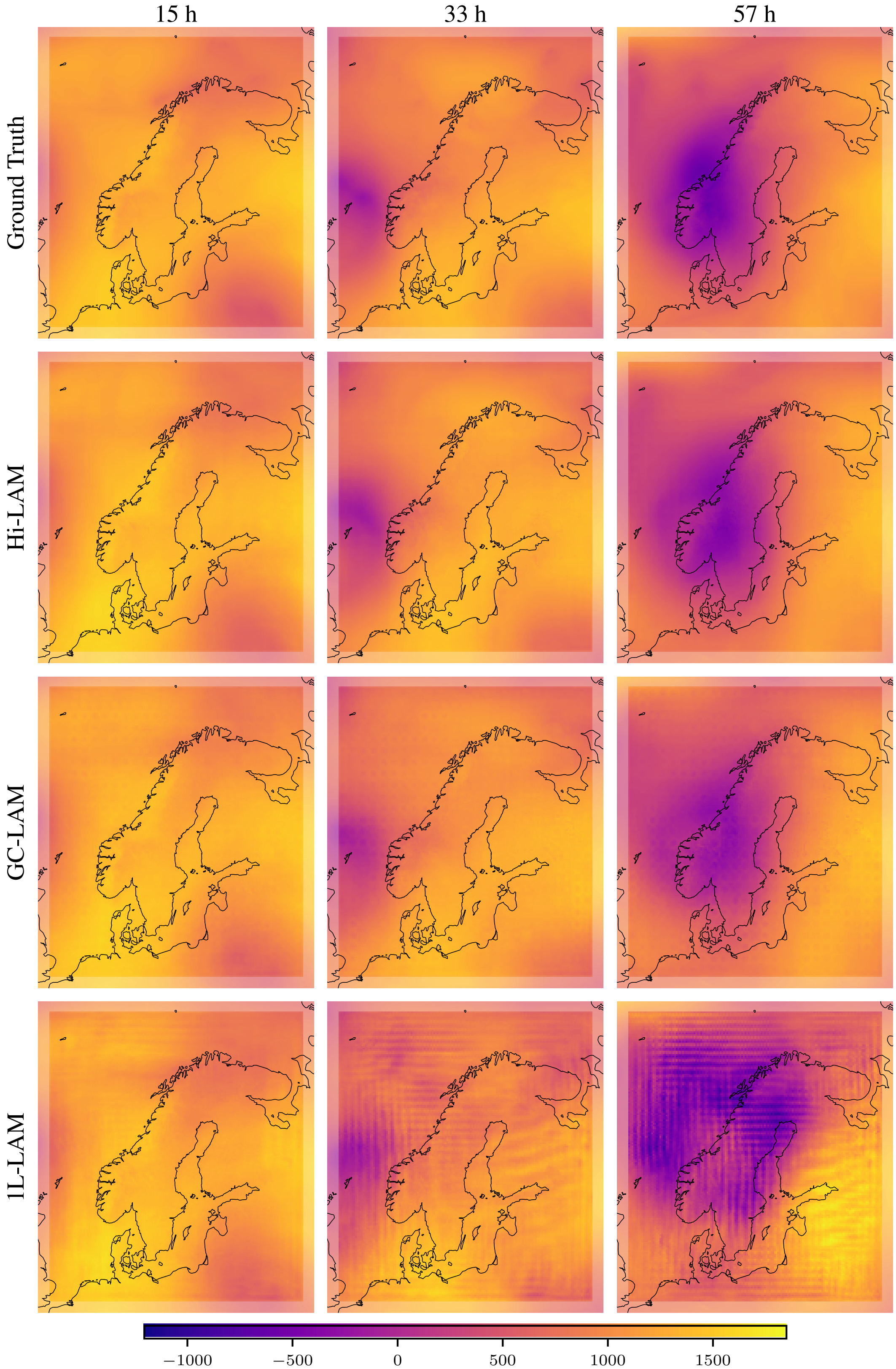
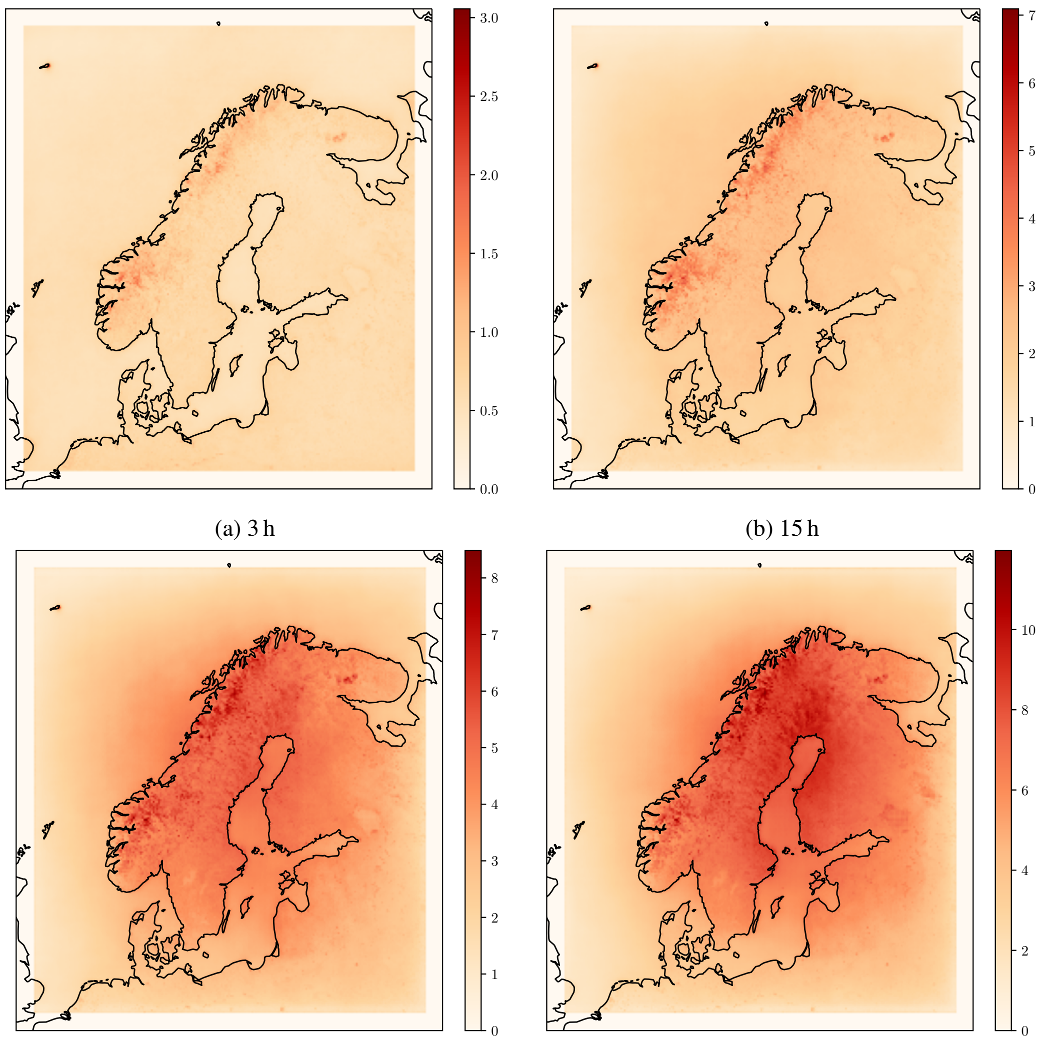


Figure 26: Example model forecasts for `z_1000` at lead times 15, 33 and 57 h.



(a) 3 h

(b) 15 h

(c) 30 h

(d) 57 h

Figure 27: Mean test loss of the Hi-LAM model, evaluated separately at all grid points and at different lead times. Note that the color map is not shared.