GRAPHCAST–MAMBA: LEARNING NON-MARKOVIAN CORRECTIONS FOR WEATHER FORECASTING

Anonymous authors
Paper under double-blind review

ABSTRACT

Unresolved atmospheric processes can introduce history dependence beyond the short input windows used by many weather forecasting models. We introduce GraphCast–Mamba, which augments a frozen, pretrained 1° GraphCast backbone with a trainable Mamba correction branch combining graph message passing and Mamba layers. The branch retains a temporal memory state across successive six-hour forecast steps, learning corrections informed by temporal context beyond GraphCast's two-frame input. Training is confined to the correction branch, using ERA5 data from 2015–2021. After only 2,500 optimizer updates, including warm-up, the selected model achieves substantial improvements relative to the uncorrected GraphCast backbone. It reduces the GraphCast loss (normalized weighted mean-squared error; MSE) by 16.3% when aggregated over the full ten-day forecast and by 23.68% at day 10 (240 h after forecast initialization). At this final forecast lead time, root-mean-square error (RMSE) decreases by 10.2–17.1% across all 11 evaluated variable groups, spanning temperature, wind, pressure, geopotential, specific humidity, and precipitation. Separate memory-reset controls support a contribution from persistent temporal context beyond additional trainable capacity. These results demonstrate broad improvements in ten-day weather forecasts through a short correction-training schedule while retaining the pretrained backbone.

1 INTRODUCTION

Learned global weather models produce skillful forecasts from one or a few recent atmospheric states (Bi et al., 2023; Lam et al., 2023; Chen et al., 2023b). GraphCast, for example, uses two states six hours apart to predict the next state. These inputs provide only a partial description of the atmosphere: they contain a limited set of variables at finite spatial resolution, while unresolved processes can still influence their evolution. The history of the resolved state may carry information about these influences that a short input window does not capture.

This is a familiar closure problem in turbulent-flow modeling. Resolving the smallest dynamically active scales is computationally costly (Yang & Griffin, 2021), so practical models must account for their effects on the resolved scales. A common approach uses approximate dissipative closures, such as eddy viscosity or hyperviscosity, to represent transfer to unresolved scales. These closures can reproduce selected aspects of the flow, but their accuracy depends on the quantity and scale of interest. For example, in forced two-dimensional turbulence, viscous, hyperviscous, and Leith closures can approximate enstrophy transfer, whereas the Smagorinsky closure dissipates enstrophy at the wrong scales (Pietarila Graham & Ringler, 2013). Resolution studies provide a related caution: mean dissipation and enstrophy can be nearly converged while higher-order moments of velocity gradients, dissipation, and enstrophy, together with distribution tails, remain substantially underestimated (Donzis et al., 2008). Reproducing bulk statistics therefore does not ensure accurate fluctuations or extreme events.

A complementary approach allows the closure to depend on the history of the resolved state. Unresolved motions have their own dynamics, so their influence need not be determined by the current resolved variables alone; past observations can contain information about those hidden degrees of freedom. The Mori–Zwanzig formalism makes this explicit: eliminating unresolved variables generally yields an instantaneous term, a history-dependent memory term, and a term associated with unresolved initial conditions, even when the full dynamics are Markovian (Lin et al., 2021;

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

De Wit et al., 2026). Retaining the memory contribution leads to a *non-Markovian* closure, in which the resolved evolution depends on past states as well as the present one. This motivates persistent temporal memory as a mechanism for representing unresolved influences that an instantaneous dissipative approximation may miss.

Evidence from learned partial differential equation (PDE) models supports this motivation. Memory Neural Operator (MemNO), which combines a neural operator with a structured state-space sequence (S4) layer, improves predictions from coarse or noisy observations (Buitrago Ruiz et al., 2025). Here, we ask whether retaining temporal context beyond GraphCast's two-frame input improves forecast corrections when the correction architecture and parameter count are held fixed.

To investigate this question, we introduce GraphCast–Mamba, which augments a frozen pretrained 1° GraphCast backbone with a trainable correction branch combining graph message passing and Mamba layers (Lam et al., 2023; Gu & Dao, 2023). Graph message passing exchanges spatial information, while Mamba retains temporal context across successive six-hour forecast steps. The branch adds a history-dependent correction to each backbone forecast, and the corrected forecast supplies the next input. Only the correction branch is trained.

Our principal experiment trains the correction on ERA5 data from 2015–2021 and evaluates ten-day forecasts on 32 matched forecast initializations from a 2022 development set at 1° resolution. This experiment tests the benefit of correction for one pretrained backbone at a fixed resolution. Its dependence on resolution and transfer to other forecasters remain open questions.

Our contributions are threefold.

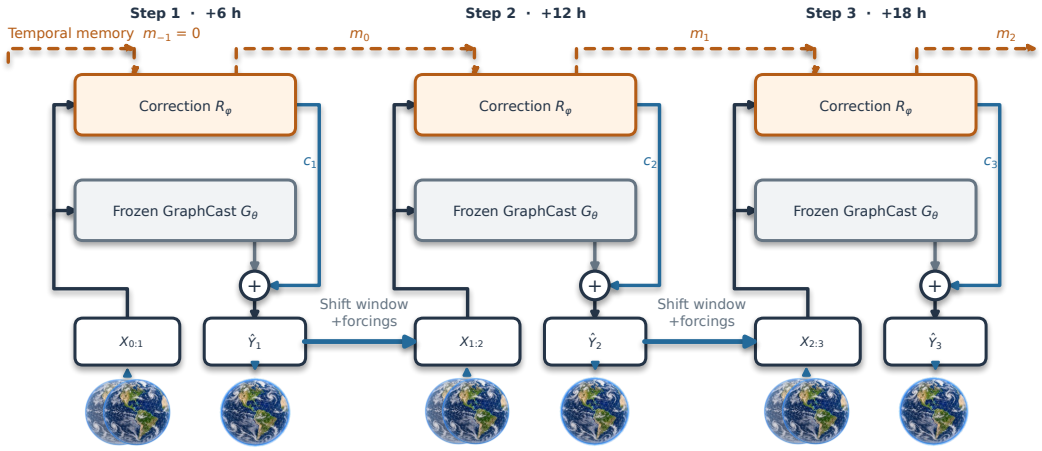
- **A temporal-memory correction model.** GraphCast–Mamba combines spatial message passing with temporal Mamba layers in a separate Mamba correction branch, extending the predictor's temporal context while keeping the pretrained GraphCast backbone frozen.
- **Evidence for a contribution from temporal memory.** Separate temporal-memory ablation experiments include a control trained without persistent temporal memory using the same architecture and parameter count. Comparisons using GraphCast loss support a benefit from retained temporal context beyond additional correction capacity.
- **Broad improvements in ten-day forecasts.** After 2,500 optimizer updates, including warm-up, our model reduces the GraphCast loss by 16.34% aggregated over the ten-day rollout and by 23.68% at the final 240-hour lead. At that final lead, RMSE decreases by 10.21–17.06% across all 11 evaluated variable groups relative to the paired frozen backbone.

2 Related Work

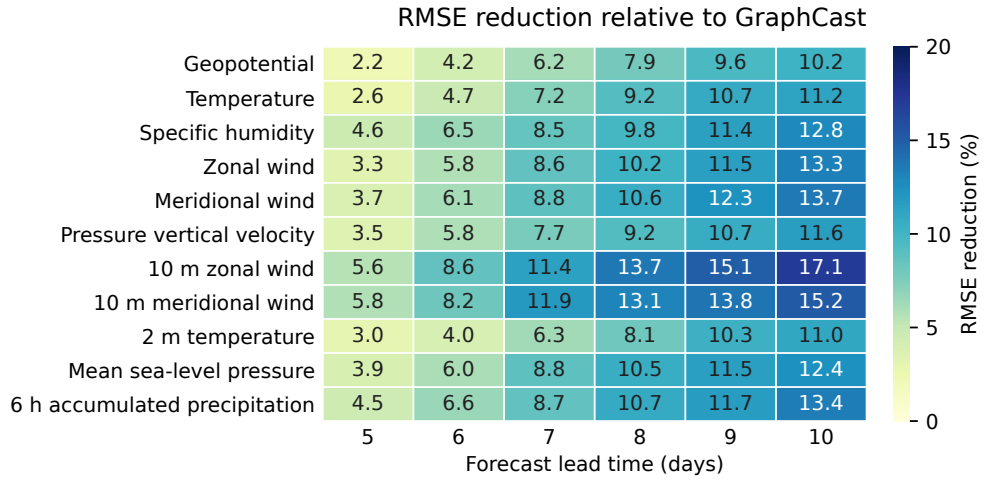
Our work connects three lines of research: memory-based modeling of partially observed dynamics, temporal memory in weather forecasting, and learned corrections to existing predictors.

Memory and learned closures in reduced dynamics. When only part of a dynamical system is resolved, the evolution of the retained variables can depend on their history, as described by the Mori–Zwanzig formalism (Lin et al., 2021). Motivated by this connection, MemNO combines Fourier neural operators with S4 memory for coarse or noisy PDE observations (Buitrago Ruiz et al., 2025); SS-NO combines spatial state-space operators with temporal memory (Koren & Lanthaler, 2025). Memory also enters learned corrections to imperfect dynamics: Gupta & Lermusiaux (2021) augment reduced models with neural delay differential equations, while Levine & Stuart (2022) develop a framework for learning both memoryless and memory-dependent model error. These studies motivate using temporal context to represent effects missing from a predictor's resolved state. Our experiments assess the predictive value of retained context without identifying a physical memory kernel.

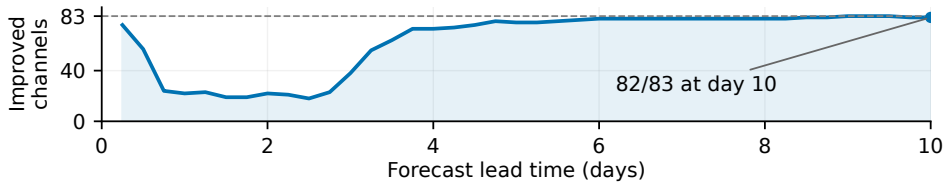
Temporal memory in weather forecasting. Recurrent weather models already retain information across forecast steps. SwinRNN carries hidden features through global forecasts, while SwinVRNN adds learned perturbations for ensemble prediction (Hu et al., 2023); SwinRDM couples recurrent forecasts with diffusion super-resolution (Chen et al., 2023a). For radar nowcasting, GauMamba combines Mamba processing with gated recurrent unit (GRU) memory to predict evolving Gaussian representations (Wang et al., 2025). These models establish temporal memory as a forecasting



(a) Autoregressive rollout over three six-hour steps. Each GraphCast–Mamba forecast advances the two-frame input window; dashed orange arrows carry the temporal memory state, initialized here with $m_{-1} = 0$.



(b) Variable-group RMSE reductions at days 5 through 10.



(c) Channel scores: channels with lower RMSE than GraphCast at each six-hour lead.

Figure 1: GraphCast–Mamba: temporal-memory correction and forecast-skill overview. Panel (a) shows forecast feedback and memory carry. In (b), positive percentages indicate lower variable-group RMSE than the paired GraphCast baseline; (c) counts channels with lower RMSE, out of 83. Counts describe aggregate differences without channel-wise significance tests. Section 4.1 defines the evaluation protocol and RMSE aggregation.

mechanism. Our architecture places memory in a separate correction branch: Mamba processes time at each mesh node, graph message passing exchanges spatial information, and the pretrained GraphCast backbone remains frozen.

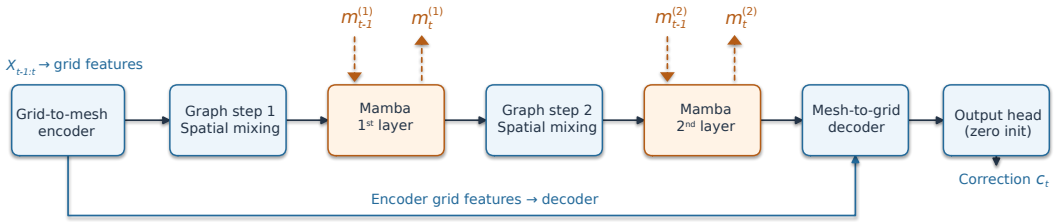


Figure 2: Trainable Mamba correction branch. Two graph message-passing steps each feed a two-layer Mamba module, followed by decoding and an output head initialized to zero. The frozen GraphCast backbone forecast $G_\theta(X_t)$ is combined with the learned forecast correction c_t to produce $\hat{Y}_t$. The Mamba correction branch has separate parameters and a temporal memory state m_t, comprising the SSM states and causal-convolution caches of its four Mamba layers.

Learned forecast corrections. Learning corrections provides a way to improve an existing predictor while retaining its forecast model. δ-Adapter learns input adjustments and output residual corrections around frozen time-series forecasters (Liang et al., 2026). In numerical weather prediction, Chen et al. (2022) learn state-dependent corrections from analysis increments and apply them during FV3-GFS integration. Solver-in-the-Loop trains corrections through differentiable solver rollouts, accounting for their effects on subsequent states (Um et al., 2020). These approaches establish both frozen-model adaptation and correction during model integration. For our weather predictor, the remaining question is whether a correction benefits from temporal context beyond the backbone's two-frame input.

These precedents motivate our investigation of persistent temporal memory in a correction branch for a frozen pretrained global weather model. Our experiments compare corrections with and without retained state using the same architecture and parameter count.

3 Method: A hybrid GraphCast–Mamba architecture

We study whether a temporal memory state improves forecasts beyond the finite input history supplied to the frozen GraphCast backbone. GraphCast uses two successive atmospheric states as input; its forecast is Markovian with respect to this augmented state. Figure 1a shows how the Mamba correction branch corrects each GraphCast backbone forecast and carries its temporal memory state across successive six-hour steps. The GraphCast–Mamba forecast advances the meteorological input window, while the temporal memory state retains information from the preceding trajectory. Figure 2 details how graph message passing mixes information across mesh nodes and Mamba updates memory independently at each node. This introduces history dependence relative to the baseline input representation.

3.1 Architecture

GraphCast backbone The frozen GraphCast backbone uses a pretrained checkpoint and also serves as the standalone forecast baseline. This graph neural network operates on a latitude–longitude grid and an auxiliary mesh, with graph $\mathcal{G}(\mathcal{V}^{\mathrm{G}}, \mathcal{V}^{\mathrm{M}}, \mathcal{E}^{\mathrm{G2M}}, \mathcal{E}^{\mathrm{M}}, \mathcal{E}^{\mathrm{M2G}})$. The vertices $\mathcal{V}^{\mathrm{G}}$ represent the meteorological grid and $\mathcal{V}^{\mathrm{M}}$ the mesh. Encoding uses grid-to-mesh edges $\mathcal{E}^{\mathrm{G2M}}$, followed by $N = 16$ graph message-passing steps over $\mathcal{E}^{\mathrm{M}}$ and decoding over mesh-to-grid edges $\mathcal{E}^{\mathrm{M2G}}$. Latent node features have width $W_{\mathrm{GC}} = 512$. The mesh provides a more compact representation for spatial processing than the meteorological grid.

Mamba correction branch: temporal memory on the mesh. The Mamba correction branch uses Mamba (Gu & Dao, 2023) to process forecast time independently at each mesh node, with spatial exchange handled by graph message passing. Each layer normalizes the width-512 mesh features, projects them into a d_{inner}-dimensional temporal representation, and projects its output back to the mesh width through a residual connection. We use $d_{\mathrm{inner}} = 16$. Across autoregressive forecast calls, each layer retains both its state-space model (SSM) state and its three-token causal-convolution cache separately for each batch element and mesh node.

Our model uses an SSM state dimension of 16 per inner channel, a causal convolution width of 4, a rank-32 projection for the step-size parameter Δ_t, and one shared group of input-dependent B_t, C_t coefficients. Additional studies examine $d_{\text{inner}} = 32$. The memory-reset controls in Section 4.5 use $d_{\text{inner}} = 16$ and one shared B, C group to test the contribution of temporal memory.

An additive correction with temporal memory. Let X_t denote the two-frame meteorological input and the associated forcings, and let the temporal memory state m_t collect the SSM states and causal-convolution caches of all Mamba layers. As shown in Figure 1a, GraphCast–Mamba combines the frozen GraphCast backbone G_θ with the trainable Mamba correction branch R_ϕ, whose output c_t is the forecast correction:

$$(c_t, m_t) = R_\phi(X_t, m_{t-1}), \qquad \hat{Y}_t = G_\theta(X_t) + c_t, \qquad \theta \text{ fixed.} \tag{1}$$

The Mamba correction branch in Figure 2 has its own grid-to-mesh encoder, mesh processor, and mesh-to-grid decoder. It receives the same meteorological inputs as the GraphCast backbone and uses separate parameters throughout. Its mesh processor uses two graph message-passing steps, each followed by a two-layer Mamba module, giving four recurrent layers in total. Encoder grid features are also passed directly to the decoder. Suppressing this grid-feature connection in the notation, the mesh-processing path is

$$L_t^{(0)} = \text{Encode}_\phi(X_t), \tag{2}$$

$$\widetilde{L}_t^{(k)} = \text{GNN}_\phi^{(k)}(L_t^{(k-1)}), \tag{3}$$

$$(L_t^{(k)}, m_t^{(k)}) = \text{MambaModule}_\phi^{(k)}(\widetilde{L}_t^{(k)}, m_{t-1}^{(k)}), \qquad k = 1, 2, \tag{4}$$

$$c_t = \text{Decode}_\phi(L_t^{(2)}). \tag{5}$$

Each Mamba module contains two residual Mamba layers; $m_t^{(k)}$ comprises their SSM states and causal-convolution caches. The output head is initialized to zero, so the initial GraphCast–Mamba forecast equals the GraphCast backbone forecast. During autoregressive forecasting, the oldest input frame is dropped, the GraphCast–Mamba forecast $\hat{Y}_t$ is appended, and the known forcings are updated to form X_{t+1}. The temporal memory state is passed separately to the next correction-branch call. The cold-start rollout illustrated in Figure 1a begins with $m_{-1} = 0$ and retains the updated temporal memory state thereafter.

3.2 Training

Initialization and trainable parameters. The Mamba correction branch inherits pretrained spatial weights and uses a zero-initialized output head, so the initial GraphCast–Mamba forecast matches the GraphCast backbone forecast. We train all correction-branch parameters ϕ on ERA5 2015–2021 while keeping the separate GraphCast backbone parameters θ frozen.

Training segments and memory carry. We divide the training sequence into contiguous segments of 96 six-hour steps (24 days), each processed as four consecutive 24-step windows (Figure 3). The temporal memory state, comprising both the SSM states and causal-convolution caches, starts at zero for each segment and is carried through all 96 steps. Each window begins with a four-step teacher-forced prefix (ERA5 inputs), followed by 20 feedback-conditioned forecast steps that use the GraphCast–Mamba forecast as feedback. Meteorological inputs therefore return to ERA5 at the start of each window, while the correction branch retains the preceding window's temporal memory state. Gradients propagate through the temporal memory state within a window; at its end, the state is carried forward with its gradient detached. Gradients are also stopped through the meteorological feedback.

Training loss. At each step, the forecast correction is supervised against the residual target $r_t = Y_t - G_\theta(X_t)$, where Y_t is ERA5 truth and X_t is the current input. Since $c_t - r_t = \hat{Y}_t - Y_t$, this minimizes the GraphCast loss ℓ_{GC} of the GraphCast–Mamba forecast. We average the loss over all $H = 24$ steps, including the teacher-forced prefix, and the B batch elements:

$$\mathcal{J}(\phi) = \frac{1}{BH} \sum_{b=1}^{B} \sum_{k=1}^{H} \ell_{\text{GC}}(\hat{Y}_{b,k}, Y_{b,k}). \tag{6}$$

Appendix A.2 gives the full loss definition and weights.

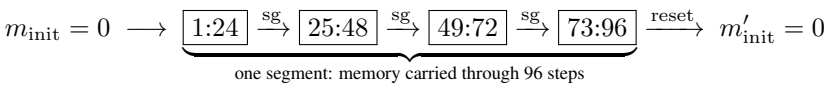


Figure 3: Memory carry during training. Boxes show the four 24-step windows of one segment. The arrows marked sg carry the SSM states and causal-convolution caches unchanged across window boundaries while stopping their gradients. Both are reset for the next segment.

Training schedule. We train with AdamW for 2,000 post-warm-up updates after a 500-update warm-up, or 2,500 optimizer updates in total. The evaluated model uses stochastic weight averaging (SWA); Appendix A gives the detailed configuration, learning-rate and optimizer comparisons, and checkpoint-selection procedure.

4 RESULTS

We evaluate whether GraphCast–Mamba, with its correction branch trained on seven years of ERA5 data (2015–2021), improves on the frozen GraphCast backbone forecasts, how broadly the gains extend across the atmospheric state, and whether temporal memory contributes beyond the correction branch's additional capacity. Separate seven-year memory-reset controls test the contribution of temporal memory.

4.1 EVALUATION PROTOCOL AND METRICS

The principal experiments use GraphCast–Mamba with the original pretrained 1° GraphCast backbone and the Mamba correction branch trained in Section 3.2. The same checkpoint supplies the lead-time, variable-group, and channel-coverage results. The principal results use the same 32 forecast initializations drawn at random from the 2022 development set, hereafter the evaluation initializations, on the native 1° grid at 40 six-hour forecast steps from 6 to 240 h. The training schedule and SWA window were fixed before evaluation. The only choice made on these forecast initializations is among the six learning-rate arms in Appendix A.3. All six reduce the ten-day loss by 14.70–16.34% and the day-10 loss by 21.98–23.68%, so the reported gains do not depend on this choice.

Each forecast takes two ERA5 states as input. GraphCast–Mamba initializes the temporal memory state to zero, and advances for ten days using its own predictions as feedback. GraphCast–Mamba feeds back its full prediction, while GraphCast follows an independent uncorrected trajectory. Stateful models retain their updated temporal memory state between steps; the controls without persistent temporal memory reset it at every step.

We first examine aggregate forecast performance using the GraphCast loss, with difference-standard-deviation normalization, latitude weights, pressure-level weights, and variable weights. For forecast initializations indexed by s, let $\mathcal{L}_{s,k}$ denote this loss at six-hour lead step k. The ten-day rollout loss reduction is

$$\mathcal{I}_{1:40} = 100 \left(1 - \frac{\sum_s \sum_{k=1}^{40} \mathcal{L}_{s,k}^{\text{model}}}{\sum_s \sum_{k=1}^{40} \mathcal{L}_{s,k}^{\text{GC}}} \right). \tag{7}$$

Day-10 loss corresponds to $k = 40$ only. Losses are aggregated before taking the ratio; the rollout score is not an average of per-lead percentages.

For each variable and lead, RMSE pools squared physical-unit errors across forecast initializations using normalized cosine-latitude weights before taking the square root. Upper-air variable-group RMSE additionally pools squared errors equally over 13 pressure levels; channel RMSE scores each surface variable and each upper-air pressure level separately. We report $(1-\text{RMSE}_{\text{model}}/\text{RMSE}_{\text{GC}})\times 100\%$, with positive values denoting lower error.

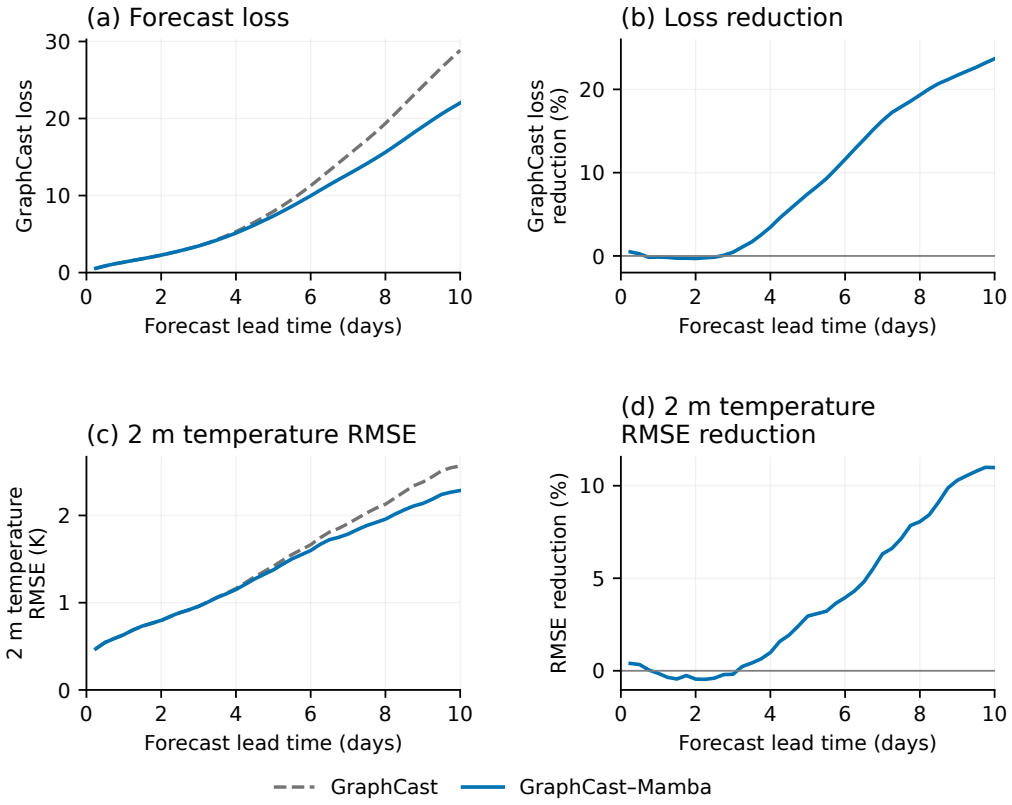


Figure 4: GraphCast–Mamba forecast skill on the evaluation initializations (Section 4.1). Top: GraphCast loss and its per-lead reduction. Bottom: 2 m temperature RMSE and RMSE reduction on the same forecast initializations. The Mamba correction branch uses $d_{\text{inner}} = 16$, a threefold Mamba learning rate (LR), and SWA over post-warm-up updates 500–2,000. Every reduction uses the paired GraphCast baseline shown in the absolute-error panels.

4.2 Forecast skill improves at longer forecast lead times

Figure 4 shows that the benefit of the correction is concentrated at longer forecast lead times. Relative to the GraphCast baseline, the GraphCast loss decreases by 16.3% over the full ten-day rollout and by 23.68% at the final 240-hour lead.

Physical-unit errors show a similar pattern. For 2 m temperature, RMSE remains within 0.5% of the baseline through day 3, before the reduction grows at longer forecast lead times. At day 10, RMSE falls from 2.57 K to 2.29 K, an 11.0% reduction.

4.3 Gains extend across the predicted atmospheric state

The gains at longer forecast lead times extend across the predicted atmospheric state. All 11 variable groups have lower RMSE at every evaluated six-hour lead from day 5 through day 10, with day-10 reductions of 10.2–17.1% (Figure 1b). These improvements span circulation, thermodynamic, moisture, and surface quantities. Table 3 in Appendix A.6 reports the reductions by variable at selected leads, together with absolute day-10 RMSEs.

Appendix D complements these aggregate results with individual forecast trajectories at Paris and New York City, including both improvements and larger local errors.

4.4 Comparison with higher-resolution GraphCast

Figure 5 compares the GraphCast–Mamba model with its GraphCast backbone and the GraphCast ($0.25°$) reference (Lam et al., 2023) for geopotential at 500 hPa, 2 m temperature, and zonal wind at 850 hPa. All forecasts are verified on a common $1.5°$ grid. The higher-resolution reference has lower errors at short leads, while GraphCast–Mamba shows slower error growth and achieves lower RMSE at longer forecast lead times in all three quantities. This comparison shows that the learned correction can improve skill at longer forecast lead times even relative to a model with finer output resolution. Appendix C provides the complete set of variables.

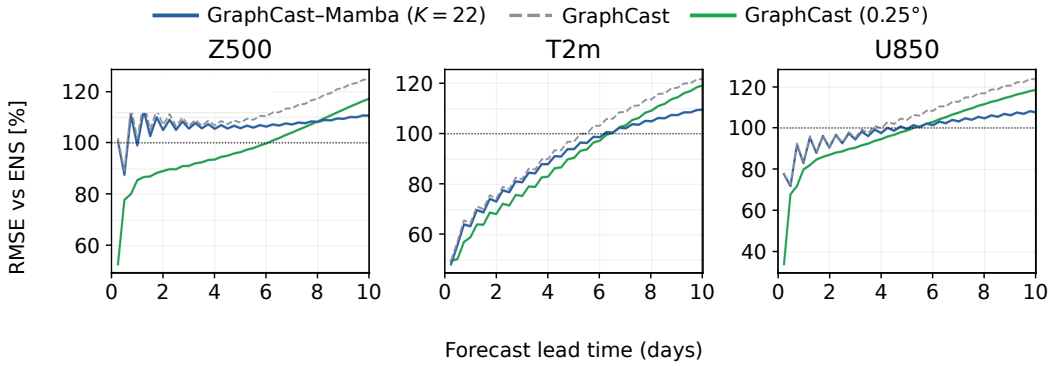


Figure 5: Comparison with higher-resolution GraphCast. From left to right: 500 hPa geopotential (Z500), 2 m temperature (T2m), and 850 hPa zonal wind (U850). Blue shows GraphCast–Mamba model at $1°$, dashed gray its GraphCast baseline, and green the GraphCast ($0.25°$) reference. Forecasts are verified on a common $1.5°$ grid; RMSE is expressed as a percentage of the ECMWF ensemble (ENS) reference, with 100% indicating equal error.

4.5 Temporal memory contributes beyond correction capacity

The preceding results show that GraphCast–Mamba improves forecasts at longer lead times across the predicted atmospheric state. The correction branch introduces both additional trainable capacity and temporal memory. To isolate the contribution of temporal memory in the Mamba correction branch, we test whether retaining this temporal context improves forecasts relative to a correction branch with the same architecture but no persistent temporal memory.

Table 1 summarizes three policies obtained from two training runs. Policy names specify memory handling during training first and forecasting second. Carry passes the updated SSM state and causal-convolution cache to the next step; reset zeros both before every step. Resetting leaves all learned weights and the two-frame meteorological input in place: the full correction branch still computes a correction, but cannot retain its temporal memory state between calls. Every training segment and independent forecast begins with a zero temporal memory state under all three policies.

Policy	Temporal memory retained	
Training/forecasting	Training	Forecasting
Carry/carry	✓	✓
Carry/reset	✓	×
Reset/reset	×	×

Table 1: Memory retained between successive steps: ✓ denotes retention; × denotes resetting both the SSM state and causal-convolution cache before every step. Carry/carry and carry/reset share the same trained weights. Reset/reset is trained separately with the same architecture and parameter count.

The two training runs use the same seven-year dataset, architecture, optimizer, and training seed, differing in their training/forecasting memory policy. Both use a training tail of $K = 20$ feedback-

conditioned steps, inner dimension $d_{\text{inner}} = 16$. These are separate runs from the principal forecast-skill experiment. We evaluate all three policies on the forecast initializations and lead times of Section 4.1. Appendix A.2 describes the loss calculation for these controls.

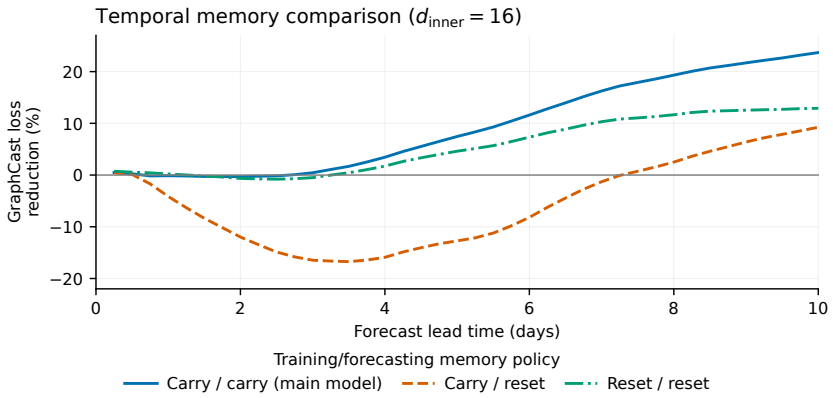


Figure 6: Comparison of carry/carry, carry/reset, and reset/reset memory policies defined in Table 1. Curves show GraphCast loss reduction relative to each model's paired GraphCast baseline over ten forecast days.

The memory controls distinguish reliance on the temporal memory state from the benefit of training with persistent temporal memory (Table 1). In the separate control runs, resetting the memory-trained model during forecasting (carry/reset) reduces the day-10 GraphCast loss reduction from 23.68% to 9.24%. This shows that the model uses its temporal memory state, although the drop may partly reflect the mismatch between training and forecasting. Training the correction with its temporal memory state reset at every step (reset/reset) removes this mismatch and yields a 12.91% reduction with the same architecture and parameter count. The control run's carry/carry policy therefore improves on a correction trained without persistent temporal memory by 10.77 percentage points, supporting a benefit from temporal context beyond additional trainable capacity. Figure 6 shows the principal model's carry/carry curve in place of the control run's carry/carry curve; its comparison with the reset controls also includes differences in training configuration and loss measurement.

Together, the two comparisons show that the memory-trained model uses its temporal memory state and outperforms the separately trained control without persistent temporal memory in this setting. This supports useful history dependence beyond the baseline's two-frame input.

5 Conclusion

Our results show that a small correction branch with persistent temporal memory can improve forecasts from a frozen pretrained weather model at longer forecast lead times. Memory controls support a benefit from retained history beyond the branch's additional trainable capacity. This suggests that compact temporal extensions offer a promising way to improve existing forecasters with limited additional training. How this benefit depends on spatial resolution and the choice of backbone remains an open question.

Limitations. Deterministic AI weather models can produce overly smooth fields and struggle to predict extremes despite good average skill. Zhang et al. (2025) discuss this smoothing tendency and find that several leading models underestimate the frequency and intensity of record-breaking heat, cold, and wind events. We have not systematically evaluated whether GraphCast–Mamba reduces smoothing or improves extreme-event prediction. The aggregate loss and RMSE reductions reported here therefore do not establish improved skill for rare extremes.

AI USE STATEMENT

We used a large language model for minor editorial assistance, including correcting grammar and typo and refining formatting. It did not contribute to the development of research ideas, the analysis, or the claims.

ETHICS STATEMENT

All datasets, models in this work are publicly available. The usage of these datasets, models, code strictly follows the terms and conditions set by the providers.

REPRODUCIBILITY STATEMENT

To ensure reproducibility of our results, we provide detailed documentation of our experimental setup and methodology in the main paper and appendix. The code for data, evaluation, and training will be open-sourced upon publication.

REFERENCES

Kaifeng Bi, Lingxi Xie, Hengheng Zhang, Xin Chen, Xiaotao Gu, and Qi Tian. Accurate medium-range global weather forecasting with 3d neural networks. *Nature*, 619:533–538, 2023. doi: 10.1038/s41586-023-06185-3. URL https://doi.org/10.1038/s41586-023-06185-3.

Ricardo Buitrago Ruiz, Tanya Marwah, Albert Gu, and Andrej Risteski. On the benefits of memory for modeling time-dependent PDEs. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/89379d5fc6eb34ff98488202fb52b9d0-Abstract-Conference.html.

Lei Chen, Fei Du, Yuan Hu, Zhibin Wang, and Fan Wang. SwinRDM: Integrate SwinRNN with diffusion model towards high-resolution and high-quality weather forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 322–330, 2023a. doi: 10.1609/aaai.v37i1.25105. URL https://ojs.aaai.org/index.php/AAAI/article/view/25105.

Lei Chen, Xiaohui Zhong, Feng Zhang, Yuan Cheng, Yinghui Xu, Yuan Qi, and Hao Li. FuXi: a cascade machine learning forecasting system for 15-day global weather forecast. *npj Climate and Atmospheric Science*, 6:190, 2023b. doi: 10.1038/s41612-023-00512-1. URL https://doi.org/10.1038/s41612-023-00512-1.

Tse-Chun Chen, Stephen G. Penny, Jeffrey S. Whitaker, Sergey Frolov, Robert Pincus, and Stefan Tulich. Correcting systematic and state-dependent errors in the NOAA FV3-GFS using neural networks. *Journal of Advances in Modeling Earth Systems*, 14:e2022MS003309, 2022. doi: 10.1029/2022MS003309. URL https://doi.org/10.1029/2022MS003309.

Xander M De Wit, Alessandro Gabbana, Michael Woodward, Yen Ting Lin, Federico Toschi, and Daniel Livescu. Data-driven mori–zwanzig modeling of lagrangian particle dynamics in turbulent flows. *Proceedings of the National Academy of Sciences*, 123(13):e2525390123, 2026.

D. A. Donzis, P. K. Yeung, and K. R. Sreenivasan. Dissipation and enstrophy in isotropic turbulence: Resolution effects and scaling in direct numerical simulations. *Physics of Fluids*, 20(4):045108, 04 2008. ISSN 1070-6631. doi: 10.1063/1.2907227. URL https://doi.org/10.1063/1.2907227.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Abhinav Gupta and Pierre F. J. Lermusiaux. Neural closure models for dynamical systems. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 477(2252):20201004, 2021. doi: 10.1098/rspa.2020.1004. URL https://doi.org/10.1098/rspa.2020.1004.

Yuan Hu, Lei Chen, Zhibin Wang, and Hao Li. SwinVRNN: A data-driven ensemble forecasting model via learned distribution perturbation. *Journal of Advances in Modeling Earth Systems*, 15: e2022MS003211, 2023. doi: 10.1029/2022MS003211. URL https://doi.org/10.1029/2022MS003211.

Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization, 2019. URL https://arxiv.org/abs/1803.05407.

Nodens F. Koren and Samuel Lanthaler. Merging memory and space: A state space neural operator. arXiv preprint arXiv:2507.23428, 2025. URL https://arxiv.org/abs/2507.23428v5. Version 5, revised March 2026.

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, and Peter Battaglia. Learning skillful medium-range global weather forecasting. *Science*, 382 (6677):1416–1421, 2023. doi: 10.1126/science.adi2336. URL https://doi.org/10.1126/science.adi2336.

Matthew E. Levine and Andrew M. Stuart. A framework for machine learning of model error in dynamical systems. *Communications of the American Mathematical Society*, 2:283–344, 2022. doi: 10.1090/cams/10. URL https://doi.org/10.1090/cams/10.

Daojun Liang, Qi Li, Yinglong Wang, Jing Chen, Hu Zhang, Xiaoxiao Cui, Qizheng Wang, and Shuo Li. The forecast after the forecast: A post-processing shift in time series. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/6f7d90b1198fec96defd80b5ebd5bc81-Abstract-Conference.html.

Yen Ting Lin, Yifeng Tian, Daniel Livescu, and Marian Anghel. Data-driven learning for the mori–zwanzig formalism: A generalization of the koopman learning framework. *SIAM Journal on Applied Dynamical Systems*, 20(4):2558–2601, 2021. doi: 10.1137/21M1401759. URL https://doi.org/10.1137/21M1401759.

Jonathan Pietarila Graham and Todd Ringler. A framework for the evaluation of turbulence closures used in mesoscale ocean large-eddy simulations. *Ocean Modelling*, 65:25–39, 2013. doi: 10.1016/j.ocemod.2013.01.004. URL https://doi.org/10.1016/j.ocemod.2013.01.004.

Kiwon Um, Robert Brand, Yun (Raymond) Fei, Philipp Holl, and Nils Thuerey. Solver-in-the-loop: Learning from differentiable physics to interact with iterative PDE-solvers. In *Advances in Neural Information Processing Systems*, volume 33, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/43e4e6a6f341e00671e123714de019a8-Abstract.html.

Ziye Wang, Yiran Qin, Lin Zeng, and Ruimao Zhang. High-dynamic radar sequence prediction for weather nowcasting using spatiotemporal coherent Gaussian representation. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/4d1344317478ad99ff5f4e414aeab689-Abstract-Conference.html.

Greg Yang, Edward J. Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tensor programs V: Tuning large neural networks via zero-shot hyperparameter transfer. arXiv preprint arXiv:2203.03466, 2022. URL https://arxiv.org/abs/2203.03466.

Xiang IA Yang and Kevin P Griffin. Grid-point and time-step requirements for direct numerical simulation and large-eddy simulation. *Physics of Fluids*, 33(1), 2021.

Zhongwei Zhang, Erich Fischer, Jakob Zscheischler, and Sebastian Engelke. Numerical models outperform AI weather forecasts of record-breaking extremes. arXiv preprint arXiv:2508.15724, 2025. URL https://arxiv.org/abs/2508.15724.

A TRAINING DETAILS AND MODEL SELECTION

A.1 TRAINING CONFIGURATION

The configuration below supplies the principal GraphCast loss and RMSE results in Section 4.

Initialization and batching. We initialize the Mamba correction branch from the frozen GraphCast backbone checkpoint: its grid-to-mesh encoder (G2M), mesh-to-grid decoder (M2G), and two graph message-passing steps copy the corresponding pretrained weights, using the first two of GraphCast's 16 graph message-passing steps. We insert newly initialized Mamba modules after these graph message-passing steps and append a linear output head with weights and biases initialized to zero. Consequently, the initial forecast correction is zero and the GraphCast–Mamba forecast equals the pretrained GraphCast backbone forecast. The copied spatial parameters, Mamba modules, and output head form the trainable parameters ϕ; the separate GraphCast backbone parameters θ remain frozen throughout training on ERA5 2015–2021. Each optimizer update processes one window from each of four independent segments, giving batch size $B = 4$ under the memory-carry procedure in Section 3.2.

Optimizer and checkpoint settings. The reported model uses a 500-update warm-up followed by 2,000 post-warm-up updates. We use AdamW with moments $(0.9, 0.98)$ and a base peak learning rate of 10^{-4}; the Mamba parameters receive three times the spatial learning rate. The evaluated model averages the parameters saved at post-warm-up updates 500, 1,000, 1,500, and 2,000 using SWA. These checkpoint indices count updates after warm-up. The configuration uses spatial width 512, inner dimension 16, and one B, C group.

A.2 COMPLETE LOSS DEFINITION

The per-step loss in Equation 6 is the GraphCast loss:

$$\ell_{\mathrm{GC}}(\hat{Y}, Y) = \sum_{v} \frac{\lambda_v}{N_{\mathrm{g}} L_v} \sum_{i=1}^{N_{\mathrm{g}}} \sum_{\ell=1}^{L_v} a_i q_{v,\ell} \left(\frac{\hat{Y}_{v,\ell,i} - Y_{v,\ell,i}}{\sigma_{v,\ell}} \right)^2 . \tag{8}$$

Here N_{g} is the number of grid points and L_v is the number of levels for variable v. The scales $\sigma_{v,\ell}$ are GraphCast's standard deviations of six-hour differences. The latitude weights a_i have unit mean and include the pole-cell correction. For upper-air variables, $q_{v,\ell} = p_\ell / \overline{p}$ weights levels in proportion to pressure; surface variables have $L_v = 1$ and $q_{v,1} = 1$. The variable weights λ_v are 1 for upper-air variables and 2 m temperature, and 0.1 for the two 10 m wind components, mean sea-level pressure, and six-hour accumulated precipitation.

Evaluation variants. We use GraphCast loss as the common label for the weighted MSE diagnostics. The principal results evaluate Equation 8 directly. The memory controls and closed-loop training-tail sweep reconstruct it from channel RMSEs using per-level normalization and pressure weights, without the pole-cell correction. The archived open-loop sweep instead uses variable-group RMSEs and level-averaged normalization scales. These reconstructions therefore approximate the loss above.

A.3 MAMBA CAPACITY AND LEARNING-RATE SELECTION

Learning-rate choice depends on network width and parameterization. Yang et al. (2022) derive width-dependent scaling rules under maximal update parameterization (μP), including an inverse-width scaling for Adam learning rates on hidden weight matrices. In our correction branch, the newly initialized Mamba modules contain approximately 0.11M and 0.22M parameters at inner dimensions 16 and 32, respectively, compared with 10.24M parameters in the spatial components. Motivated by these scaling results and the difference in module size, we test whether the compact Mamba modules benefit from a larger learning rate than the spatial components initialized from pretrained GraphCast weights. We treat this as an empirical hypothesis; the μP scaling rule does not directly prescribe a learning-rate multiplier for our architecture.

We compare Mamba learning-rate multipliers of $0.3\times$, $1\times$, and $3\times$ at each inner dimension, holding the spatial peak learning rate fixed at 10^{-4}. A multiplier of $1\times$ gives equal learning rates to both parameter groups. All six configurations use spatial width 512, one B, C group, batch size four, and AdamW moments $(0.9, 0.98)$. Each run has its own 500-update warm-up followed by 2,000 post-warm-up updates, with SWA over post-warm-up updates 500, 1,000, 1,500, and 2,000. The GraphCast backbone remains frozen throughout. Warm-up parameter values differ across runs despite the common training seed, so the comparisons assess complete training runs.

Table 2: Mamba capacity and learning-rate comparison at 1°. Entries are GraphCast loss reductions (%) on the evaluation initializations (Section 4.1); bold marks the largest value in each metric column. The multiplier applies only to Mamba parameters, with the spatial learning rate fixed. All six configurations use the same training schedule and SWA window.

d_{inner}	Mamba LR multiplier	Ten-day aggregate	Day 10
16	$0.3\times$	14.70	21.98
16	$1\times$	15.67	22.44
16	$3\times$	**16.34**	**23.68**
32	$0.3\times$	15.82	22.96
32	$1\times$	16.29	23.32
32	$3\times$	15.73	22.75

Table 2 shows that, for inner dimension 16, increasing the Mamba learning-rate multiplier from $1\times$ to $3\times$ raises the ten-day aggregate loss reduction from 15.67% to 16.34% and the day-10 reduction from 22.44% to 23.68%. For inner dimension 32, the best tested multiplier is $1\times$; increasing it to $3\times$ lowers the aggregate reduction from 16.29% to 15.73%. Dimension 32 performs better at $0.3\times$ and $1\times$, whereas dimension 16 performs better at $3\times$. Thus, the smaller GraphCast-Mamba head benefits from a larger learning rate in these runs, supporting joint selection of Mamba capacity and learning rate. The results do not establish a monotonic benefit from increasing either quantity.

A.4 Optimizer configuration study

The optimizer study varies peak learning rate (10^{-4} or 5×10^{-5}), Adam moments, Mamba initialization scheme (legacy or Mamba1), and the paired inner-dimension/group settings (16/1 or 64/4). The latter changes two factors simultaneously and does not isolate inner dimension. Runs use 10,000 updates, seed 22, a 200-update warm-up, cosine decay to one tenth of the peak learning rate, and weight decay 10^{-4}. The correction branch is trained with closed-loop feedback, 24-step backpropagation windows, a training tail of 20 feedback-conditioned steps, and loss across all 24 steps. The forecast correction contribution to the feedback is stop-gradient, while temporal memory state gradients propagate within the backpropagation window.

Day-10 reductions for the 22 completed selected SWA configurations range from 13.47% to 23.21%. The configuration at the upper end uses Mamba1 initialization, inner dimension 16, one B, C group, learning rate 10^{-4}, and AdamW moments $(0.9, 0.98)$; its aggregate loss reduction is 16.28%.

A.5 Checkpoint averaging and model selection

Stochastic weight averaging. SWA averages correction-branch parameter values from saved checkpoints to reduce dependence on a single training iterate (Izmailov et al., 2019). Every configuration in Table 2 averages four checkpoints at post-warm-up updates 500, 1,000, 1,500, and 2,000. The dimension-16 Mamba1-initialized optimizer-study configuration described above averages seven checkpoints, saved every 1k updates from 2k through 8k. These are parameter averages, not ensembles of independent forecast predictions. Evaluation initializes the temporal memory state to zero, including for SWA models.

Training methods and model selection. Table 2 reports all six SWA configurations from the Mamba capacity and learning-rate comparison. Figure 4 uses the dimension-16, $3\times$ Mamba learning-rate SWA configuration, which has the largest aggregate and day-10 GraphCast loss reductions among the completed SWA configurations in the reported optimizer and learning-rate studies.

In the optimizer study, each run screens periodic checkpoints and three SWA windows on eight forecast initializations. The selected checkpoint and SWA are then evaluated on the evaluation initializations.

A.6 VARIABLE-GROUP RMSE

Table 3 provides the numerical results underlying Figure 1b at selected forecast leads. It uses the principal seven-year correction and its paired GraphCast baseline on the 32 development-set forecast initializations in 2022, evaluated on the native 1° grid as described in Section 4.1.

	Day-10 RMSE		RMSE reduction (%)		
Variable [unit]	GraphCast	GraphCast–Mamba	Day 5	Day 7	Day 10
Upper-air variables (13 pressure levels)					
Geopotential [$m^2\,s^{-2}$]	774.74	695.67	2.23	6.16	10.21
Temperature [K]	3.10	2.76	2.59	7.24	11.22
Specific humidity [$g\,kg^{-1}$]	1.08	0.94	4.64	8.48	12.83
Zonal wind [$m\,s^{-1}$]	8.69	7.53	3.28	8.60	13.35
Meridional wind [$m\,s^{-1}$]	9.06	7.82	3.69	8.75	13.66
Pressure vertical velocity [$Pa\,s^{-1}$]	0.154	0.136	3.46	7.67	11.60
Surface variables					
10 m zonal wind [$m\,s^{-1}$]	4.31	3.57	5.56	11.37	17.06
10 m meridional wind [$m\,s^{-1}$]	4.48	3.80	5.78	11.90	15.18
2 m temperature [K]	2.57	2.29	2.96	6.32	10.98
Mean sea-level pressure [hPa]	7.16	6.27	3.87	8.80	12.42
6 h accumulated precipitation [mm]	2.35	2.04	4.54	8.66	13.44

Table 3: Physical-unit RMSE and relative RMSE reduction for the principal GraphCast–Mamba model. Days 5, 7, and 10 denote leads of 120, 168, and 240 h. RMSE pools cosine-latitude-weighted squared errors across forecast initializations and, for upper-air groups, equally across 13 pressure levels before taking the square root. Each reduction is $100(1 - \mathrm{RMSE}_{\mathrm{model}}/\mathrm{RMSE}_{\mathrm{GC}})$, computed before rounding. Positive values indicate lower RMSE; these percentages are distinct from GraphCast loss reductions.

B TRAINING TAIL LENGTH

We examine how forecast skill depends on the training tail length K under two input-feedback policies. Each 24-step training window begins with a *teacher-forced prefix*: the model receives ERA5 states as meteorological inputs while updating and retaining its temporal memory. For $K > 1$, the final K steps instead use forecast feedback, leaving $24 - K$ teacher-forced steps. The label $K = 1$ denotes fully teacher-forced training, with ERA5 inputs at every step. We evaluate the resulting models over 40 six-hour forecast steps (ten days) and report GraphCast loss reduction relative to GraphCast; positive values indicate improved forecast skill.

B.1 CLOSED-LOOP TRAINING(OUR MAIN MODEL)

In *closed-loop training*, the corrected GraphCast–Mamba forecast supplies the next meteorological input during the training tail. The learned correction therefore affects both the current prediction and the inputs encountered at subsequent steps. Figure 7 compares models trained with different K, showing how the duration of this feedback changes forecast skill.

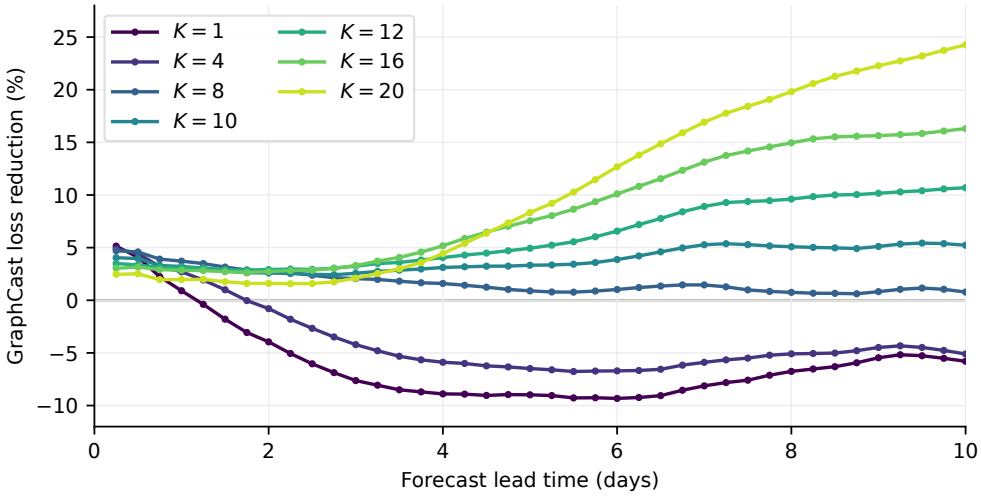


Figure 7: Forecast skill after closed-loop training with different training tail lengths K. During the tail, corrected GraphCast–Mamba predictions supply subsequent inputs. Each curve shows GraphCast loss reduction relative to GraphCast over ten forecast days; positive values indicate improvement.

Short training tails primarily improve early forecasts. Models trained with $K = 1$ and $K = 4$ reduce error during the first one to two days but increase it thereafter, whereas longer training tails yield substantial gains at longer lead times. At day 10, the error reduction relative to GraphCast increases from 5.2% for $K = 10$ to 24.3% for $K = 20$. These curves reveal a crossover: shorter tails favor early forecast skill, while longer tails become more effective as the forecast advances. To examine this crossover in a more controlled setting, we remove correction feedback and train the Mamba correction branch on precomputed forecast trajectories from the frozen GraphCast backbone. In this open-loop setting, corrections do not affect subsequent backbone predictions. As shown in the next section, the crossover becomes more pronounced under this training scheme.

B.2 Open-loop training and a sharper crossover

In *open-loop training*, the teacher-forced prefix is followed by a tail driven by the uncorrected GraphCast forecasts. The correction branch learns along this trajectory, but its predictions do not alter subsequent meteorological inputs. Thus, open loop refers to the absence of correction feedback, while teacher forcing refers to the use of ERA5 inputs in the prefix.

In our experiments, training with this teacher-forced prefix and a tail independent of the learned correction was more stable but yielded smaller improvements at longer lead times. Figure 8 shows that short-tail models remain close to GraphCast at long lead times, avoiding the substantial degradation seen in the closed-loop comparison. The crossover is more pronounced here: long-lead gains change markedly between $K = 16$ and $K = 18$ and increase further for $K = 20$ and $K = 22$.

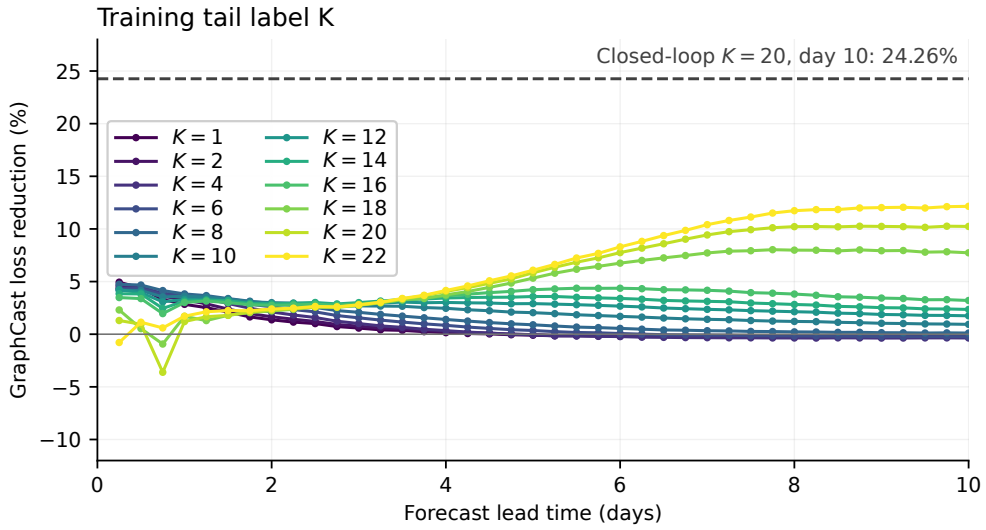


Figure 8: Forecast skill after open-loop training with a teacher-forced prefix and different training tail lengths K. During the tail, uncorrected GraphCast forecasts supply subsequent inputs. GraphCast loss reduction exhibits a pronounced crossover, with substantially larger long-lead gains for $K \geq 18$. The dashed line marks the day-10 reduction of 24.26% for the closed-loop $K = 20$ model in Figure 7. Both figures use the same vertical scale. This is a reference across separate sweeps with different experimental settings and loss reconstructions (Appendix A.2).

This crossover raises an interesting question about how temporal memory, training tail length, and feedback interact. Its mechanism, including why it is sharper under open-loop training, remains unclear. We regard it as an empirical finding that motivates further investigation, rather than evidence for a universal critical value of K.

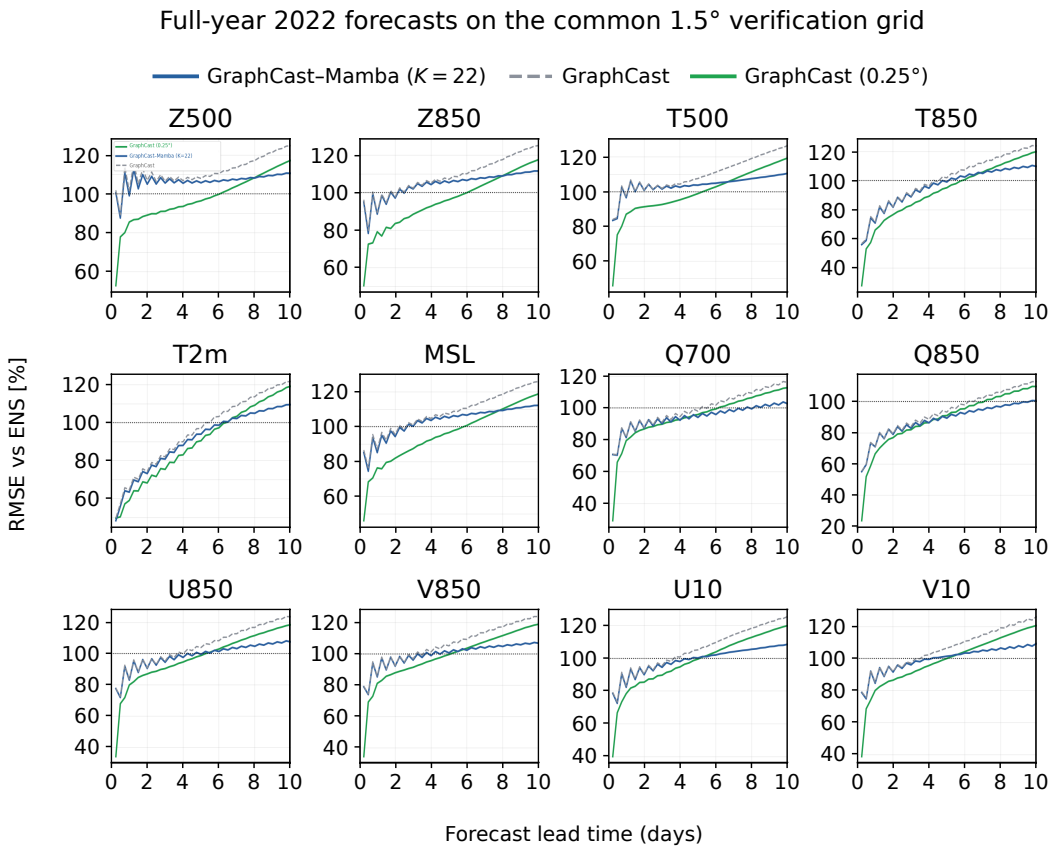


Figure 9: Complete common-grid comparison. The quantities span geopotential, temperature, mean sea-level pressure, specific humidity, and wind. The panel labels use Z for geopotential, T for temperature, Q for specific humidity, U for zonal wind, and V for meridional wind; a three-digit suffix gives pressure in hPa. T2m denotes 2 m temperature, U10 and V10 denote the 10 m wind components, and MSL denotes mean sea-level pressure. Blue is our GraphCast–Mamba model at 1° and $K = 22$, dashed gray its GraphCast baseline, and green GraphCast (0.25°). Forecasts are verified on a common 1.5° grid, with RMSE normalized to the ECMWF ensemble (ENS) reference (100% denotes equal RMSE).Figure 5 presents three representative quantities in the main text.

C Comparison on a common verification grid

We further assess forecast skill for the meteorological quantities shown in Figure 9, following the variables considered in the original GraphCast study (Lam et al., 2023). Although the models operate at different native spatial resolutions, all forecasts are projected and evaluated on the same 1.5° verification grid, so that the comparison reflects forecast accuracy rather than differences in verification resolution. Across these variables, our 1° GraphCast–Mamba model exhibits slower error growth with forecast lead time than its uncorrected GraphCast baseline. Remarkably, at longer lead times it also attains lower errors than the GraphCast (0.25°) reference, despite operating at a substantially coarser native resolution. The GraphCast (0.25°) reference itself was trained with considerable computational resources, approximately four weeks on 32 TPU v4 devices (Lam et al., 2023).

D FORECAST EXAMPLES AT CITY LOCATIONS

To complement the aggregate seven-year results in Section 4, Figure 10 illustrates local forecast behavior for temperature, wind, and precipitation at Paris and New York City. These examples use a separate correction trained on 15 years of ERA5 data (2007–2021), with the configuration specified in the figure caption. Over 120–240 h, the correction reduces mean absolute error (MAE) in four of the six displayed city–variable pairs. Errors increase for Paris temperature and New York precipitation, illustrating that aggregate skill gains can coexist with deterioration in individual forecasts.

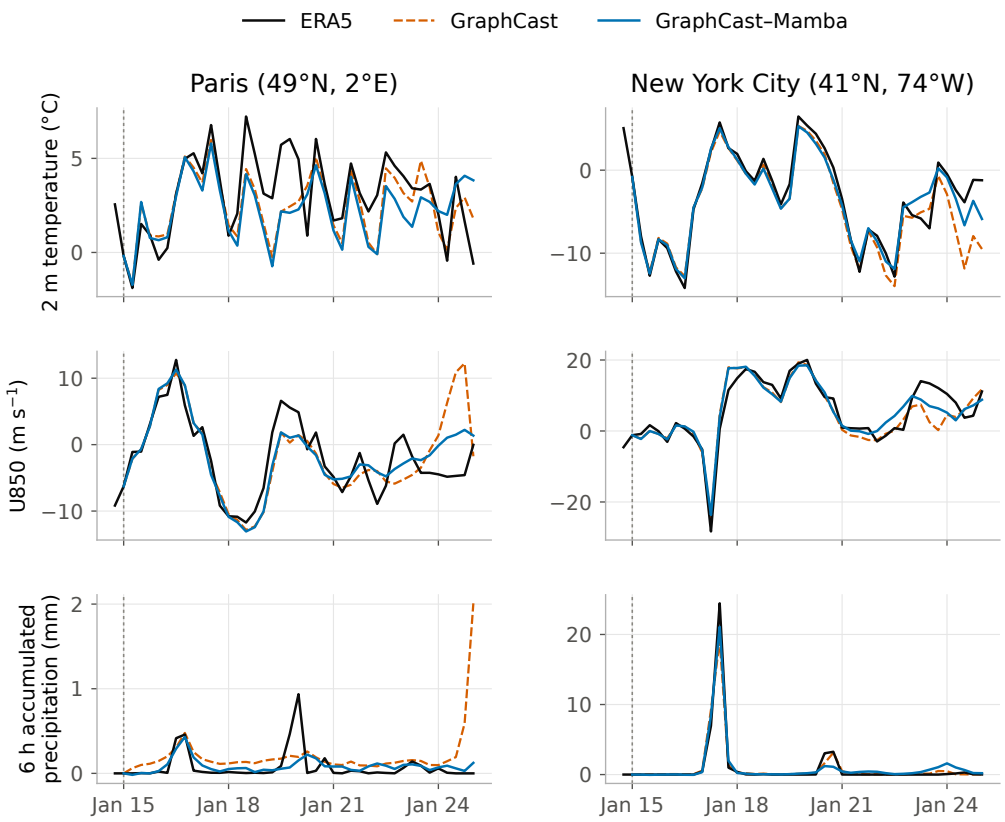


Figure 10: Ten-day forecasts from 00 UTC on 15 January 2022 at the 1° grid points nearest Paris (49°N, 2°E; left) and New York City (41°N, 74°W; right). From top to bottom, rows show 2 m temperature (T2m), 850 hPa zonal wind (U850), and 6 h accumulated precipitation in physical units. ERA5 is shown in black, the frozen GraphCast forecast in dashed orange, and the GraphCast–Mamba forecast in blue. The correction was trained separately on 15 years of ERA5 data (2007–2021), with a training tail of $K = 20$ feedback-conditioned steps, Mamba inner dimension 128, spatial width 512, and two graph message-passing steps; SWA averages checkpoints from updates 6k–11k. Both forecasts use the ERA5 states at -6 and 0 h (before and on the dotted line) as inputs, then feed back their own predicted fields at each step; the GraphCast–Mamba model begins with a zero temporal memory state. Precipitation is shown as predicted and may take small negative values.