MACHINE LEARNING
Science and Technology

Data-driven multiscale modeling for correcting dynamical systems

To cite this article: Karl Otness *et al* 2025 *Mach. Learn.: Sci. Technol.* **6** 045072

IOP Publishing *Mach. Learn.: Sci. Technol.* **6** (2025) 045072 https://doi.org/10.1088/2632-2153/ae1a36

MACHINE
LEARNING
Science and Technology

OPEN ACCESS

RECEIVED
14 May 2025

REVISED
21 October 2025

ACCEPTED FOR PUBLICATION
31 October 2025

PUBLISHED
19 December 2025

PAPER

Data-driven multiscale modeling for correcting dynamical systems

Karl Otness[*], **Laure Zanna** and **Joan Bruna**

Courant Institute School of Mathematics, Computing, and Data Science, New York University, New York, NY, United States of America
[*] Author to whom any correspondence should be addressed.

E-mail: karl.otness@nyu.edu

Keywords: data-driven modeling, subgrid parameterizations, closure modeling, multiscale modeling, climate modeling

Abstract

We propose a multiscale approach for predicting quantities in dynamical systems which is explicitly structured to extract information in both fine-to-coarse and coarse-to-fine directions. Our approach improves model accuracy and stability with minimally increased computation compared to non-multiscale approaches with analogous network architecture. We evaluate our approach on an idealized fluid subgrid parameterization (known as closure) task in which our multiscale networks correct chaotic underlying models to reflect the contributions of unresolved, fine-scale dynamics.

1. Introduction

Typical deep learning methods across a wide range of application areas make use of end-to-end learning. In such approaches, a neural network is trained such that it receives feedback which matches the requirements of the full task. However, in some applications this sort of training is impractical or even impossible. Applications of learning to scientific computing tasks—in particular to simulation problems—frequently involve real-world dynamics which may not be fully modeled or understood, or existing simulation software which is difficult to integrate with learned models and which does not admit backpropagation through simulation time steps. In these cases, training uses strictly offline data (either pre-computed or derived from real-world observations), while applying the network online after training. As a result, the network may learn behaviors which are successful on the offline training task, but which are unstable in online evaluation [1].

In this work we examine an approach to decomposing prediction tasks into separate prediction problems across scales, encouraging the network to more completely exploit available multiscale information. Networks making separate cross-scale predictions are trained separately and *not* end-to-end. We observe that this simple approach to offline training improves model stability, and improves accuracy in smaller architectures, allowing more efficient evaluation when integrated into end applications. While we believe that our approach may have wider applicability, we evaluate its performance on a set of idealized subgrid forcing prediction tasks similar to those which arise in real-world ocean, atmosphere, and climate modeling applications.

Climate models, which simulate the long-term evolution of the Earth's atmosphere, oceans, and terrestrial weather, are critical tools for projecting the impacts of climate change around the globe. Due to limits on available computational resources, these models must be run at a coarsened spatial resolution which cannot resolve all physical processes relevant to the climate system [2]. To reflect the contribution of these subgrid-scale processes, closure models are added to climate models to provide the needed subgrid-scale forcing. These parameterizations model the contribution of these fine-scale dynamics and are critical to high quality and accurate long term predictions [3, 4]. A variety of approaches to designing these parameterizations have been tested, ranging from hand-designed formulations [5], to modern machine learning with genetic algorithms [3], or neural networks trained on collected snapshots [6–9], or in an online fashion through the target simulation [1].

The problem of predicting these forcings is inherently multiscale; the subgrid dynamics which must be restored represent the impact of the subgrid and resolved scales on each other. Closure models for climate are designed to be resolution-aware [10], but even so existing deep learning subgrid models do not explicitly leverage the interactions between scales, leaving it to the neural networks to implicitly learn these relationships. Our approach makes these multiscale interactions explicit. We find that incorporating this structure into the network architecture, instead of requiring it to be learned, improves model accuracy and stability, and permits the use of smaller more efficient networks with improved performance. We introduce our approach in section 2 and discuss details of our experimental setup on two test tasks in section 3. Section 4 presents the results of our evaluations. We provide some thoughts on broader applicability and possible future extensions in section 5.

2. Approach

Recent methods for image generation have made use of diffusion modeling where an image is sampled from a learned distribution over several steps starting from noise, rather than training a network to produce the result in one shot [11, 12]. One can view this process as gradually filling in fine-scale details based on earlier coarse-scale features. This approach has found a wide variety of successful applications. Many simulation tasks, in which states must be evolved in time, have dynamics which might benefit from this approach—namely dynamics in which neighboring scales influence each other and in which coarser-scale features may be easier to predict and slower to evolve than finer-scale details. In many tasks, the coarser components of a state provide a useful foundation on which finer scale details can be predicted and filled in. Just as diffusion models achieve better results by decomposing a prediction task into several steps adding gradually sharper details, we propose to decompose correction tasks for simulation by first predicting coarser components of the state and later adding finer details. Our results show improved accuracy and stability when following such a multiscale prediction approach.

While these approaches have been successful and provide a useful inductive bias, this sampling method is generally quite expensive. Many applications of machine learning to existing simulation tasks insert a learned model as a *component* of an existing simulation and evaluate the learned model in each simulation step. In these applications, further adding multiple diffusion steps may be prohibitively costly. In this work, we try to keep many of the benefits of this generative approach while reducing the cost of evaluation by reducing the number of inference steps. More specifically, we divide the state into two scale ranges: a 'high resolution' segment containing the fine scale details and a 'low resolution' segment containing only the coarse scales. Our prediction process first produces a coarse, low resolution version of the target field, then a second step uses this initial prediction to fill in the fine scale details, yielding a high resolution output.

For a general prediction task, when predicting a high resolution field $S_{\mathsf{x}\ \mathrm{hr}}$ from a high resolution simulation state x_{hr} we can attempt to predict the quantity directly from x_{hr}. These quantities are often uncertain, but for our purposes, we will use *deterministic* models to predict the expectation of our target quantities. However, our approach could be extended to stochastic models in several natural ways. A deterministic neural network f_θ can be trained to perform this task:

$$f_\theta\left(\mathsf{x}_{\mathrm{hr}}\right) \approx \mathrm{E}[S_{\mathsf{x}\ \mathrm{hr}} | \mathsf{x}_{\mathrm{hr}}]. \tag{1}$$

In our *multiscale* approach to such a task we first predict a low resolution version of the target field, $S_{\mathsf{x}\ \mathrm{lr}}$ and condition on this prediction while producing the full resolution output:

$$f_{\theta_1}^{\mathrm{coarsened}}\left(\mathsf{x}_{\mathrm{hr}}\right) \approx \mathrm{E}[S_{\mathsf{x}\ \mathrm{lr}} | \mathsf{x}_{\mathrm{hr}}] \tag{2}$$

$$f_{\theta_2}^{\mathrm{buildup}}\left(\mathsf{x}_{\mathrm{hr}}, S_{\mathsf{x}\ \mathrm{lr}}\right) \approx \mathrm{E}[S_{\mathsf{x}\ \mathrm{hr}} | \mathsf{x}_{\mathrm{hr}}, S_{\mathsf{x}\ \mathrm{lr}}]. \tag{3}$$

We use the estimated expectation from $f_{\theta_1}^{\mathrm{coarsened}}(\mathsf{x}_{\mathrm{hr}})$ in equation (2) to provide a realization of $S_{\mathsf{x}\ \mathrm{lr}}$ in equation (3). Further scale segments could be introduced if desirable for a particular task. For our application, the low resolution field is produced by resampling the target at a lower resolution, yielding a field with smaller dimensions. That is, $S_{\mathsf{x}\ \mathrm{lr}} \triangleq D(S_{\mathsf{x}\ \mathrm{hr}})$ for some coarsening operator D. In addition to definitions provided in this section, collected definitions for notation used throughout this paper are included in appendix A.

To evaluate our multiscale approach, we consider specific instances of such prediction tasks. In particular, we examine the problem of learning subgrid forcings for fluid models, a problem which arises in climate modeling applications. In particular, we use two idealized simulations: (1) a two layer quasi-geostrophic model *QG*, and (2) Kolmogorov flow (*KF*). These two models will be introduced in further detail below and in appendix B. Each target simulation autonomously evolves a set of state variables

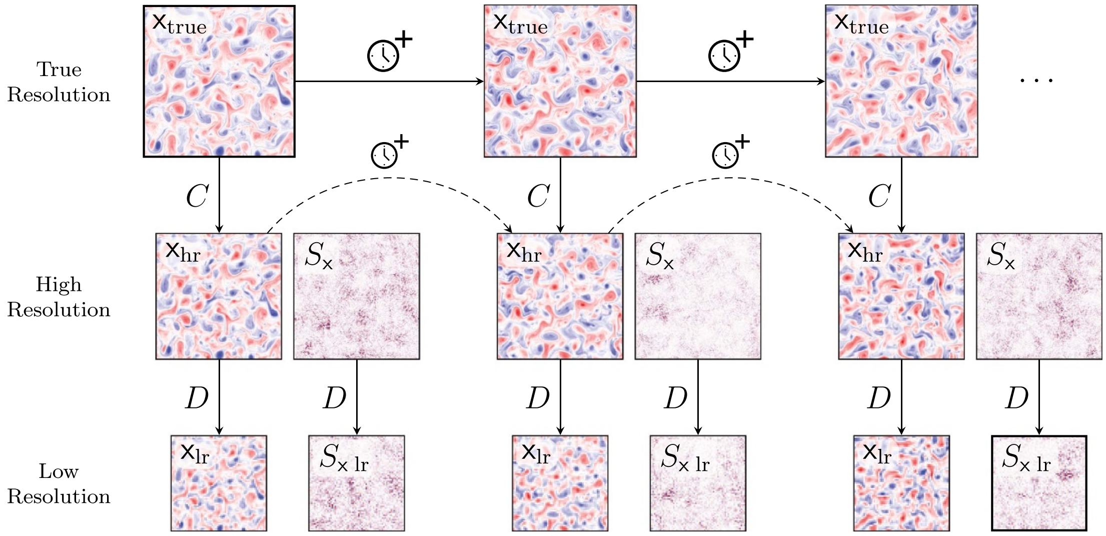


Figure 1. The several scales involved in the experiments described below. The true resolution is used only while rolling out a ground truth reference trajectory and then discarded. Samples are resampled to lower resolutions using operators C and D described in appendix D. The values S_x are computed following equation (4) using both x_{true} and x_{hr}. The high and low resolution samples are used for training. During evaluation the simulation is run online at the high resolution—following the dashed lines.

through time and can be evaluated with a configurable grid resolution. We refer to a general state variable x representing the states evolved by these two systems in the following introduction. For applications to these systems, the predicted quantity S_x is a subgrid forcing term, predicted as a field added to each layer of the base simulation. In each system states may be ported to lower resolutions by coarse-graining and filtering.

For each system we generate ground truth data by running the model at a very high ('true') resolution. This produces trajectories $x_{true}(t)$ and time derivatives $\partial x_{true}(t)/\partial t$. Next we generate training data at a high resolution by applying a system-dependent coarsening and filtering operator C giving variables $\bar{x} \triangleq C(x)$. These operators combine a coarsening step that resamples the input, with further filtering and processing. Details are included in appendix D.1. Given nonlinearities in the target simulations, this coarsening does not commute with the dynamics of the models. To correct for this we must apply a subgrid forcing term S_x to the evolution of each state variable:

$$S_x \triangleq \overline{\frac{\partial x}{\partial t}} - \frac{\partial \bar{x}}{\partial t}. \tag{4}$$

Note that formally the forcing S_x is a function of the state x_{true}. In a climate modeling application we do not have access to this variable and so we train a model $f_\theta(\bar{x}) \approx S_x$ which may be stochastic.

We continue this process, introducing another coarsening operator D and fine-scaling operator $D^\dagger$. These operators perform spectral resampling to produce coarser or finer versions of the input states. See equation (12) in appendix D.2 for the full definition. Taking $x_{hr} \triangleq \bar{x}$ as our high resolution samples, we produce low resolution samples $x_{lr} \triangleq D(x_{hr})$ and $S_{x\,lr} \triangleq D(S_x)$. This allows a decomposition $x = D^\dagger D x + x'$ where x' are the details removed by D. Our experiments thus involve three resolutions, from fine to coarse: a 'true' resolution; a high resolution, hr; and a low resolution, lr. The closures S_x try to update hr to match the 'true' resolution. The relationship between these scales and the process of generating reference samples across them are illustrated in figure 1.

Just as predicting S_x from x_{true} is fully deterministic, while predicting it from x_{hr} involves uncertainty, we anticipate a similar trend to hold for $D(S_x)$. In other words, predicting $D(S_x)$ from x_{hr} should be easier than predicting $D(S_x)$ directly from x_{lr}. Then, using this coarse-grained prediction $D(S_x)$ as a foundation, we can learn to predict only the missing details and add them. This process splits the problem of predicting S_x into two phases: (1) a 'coarsened' prediction to form $D(S_x)$, and (2) a 'buildup' prediction combining x_{hr} and $D(S_x)$ to predict S_x, adding the missing details. This decomposition takes advantage of self-similarity in the closure problem to pass information between the coarse and fine scales and improve predictions.

3. Experiments

To test this approach to predicting subgrid forcings we compare our *multiscale* approach against single-scale baselines. We apply our method to two test systems: a QG system and a KF system which is a configuration of Navier–Stokes. We select the 'high' resolution size in each target system (QG or KF) so that the system requires closure (there are sufficient dynamics below the grid-scale cutoff), but does not diverge [3]. An overview of the two test systems is provided below with further details in appendix B.

Both of the systems we use in our experiments provide simplified, idealized versions of the closure problems which arise in real climate models. While idealized, the models are nevertheless chaotic and, when run on our coarser grids, require closure to maintain the proper energy distribution. Further, the KF model in particular is sensitive to the parameterization and can become unstable, another issue which frequently arises in real climate model applications.

3.1. Test systems

We carry out our experiments on our two test systems: a two-layer QG system, and a single layer KF system [13].

QG system This system is a two-layer QG model as implemented in PyQG and provides a simplified approximation of fluid dynamics [14, 15]. This system tracks the evolution of a potential vorticity q divided into two layers with periodic boundary conditions. Corrections S_q are applied to the time evolution of this field. Models receive the current potential vorticity state q as inputs and predict S_q across both layers directly. This model has been used as an evaluation task in previous work designing and evaluating subgrid parameterizations for turbulence problems in climate models [3, 9].

KF system We also test on a KF system which is a single-layer incompressible Navier–Stokes flow with periodic boundary conditions and a periodic forcing. We use an implementation from JAX-CFD configured to have Reynolds number 7000 [16]. The state of this system is tracked by velocity components u and v and all networks are also provided a computed vorticity for each state ω. Networks predict output quantities S_u and S_v, which are corrections applied to the velocity components.

Using these systems we carry out two sets of experiments: a small set of 'separated' experiments carried out on the QG model only (these neural networks are trained and evaluated offline both with and without access to the additional multiscale information); and 'combined' experiments which run on both the QG and KF models which provide an implementable closure model trained end to end. In both cases we compare results against networks of equivalent architectures without the additional multiscale structure.

For each experiment, a set of feedforward convolutional neural networks is trained and evaluated separately. Details of the models and the tests run vary in each experiment and are described in greater detail below. However, in each case we train several independently-initialized networks to capture the variance due to initialization. Each experiment also compares the performance of two architectures, a 'small' architecture and a 'large' architecture which are modifications of networks used in past research which also investigated subgrid parameterizations [7]. Our network architectures vary in the sizes of the convolution kernels and, in some cases, network depth. Architectural details are provided below as well as in appendix C. To more completely explore the efficiency improvements which are possible using our multiscale prediction approach, a more compact 'small' architecture for the combined experiments was chosen through an architecture search process. Because the separated experiments do not provide a parameterization which can be run online (providing results useful for the search) we did not perform an architecture search for these experiments. Results are included in section 4, and information on the network architectures and training procedure is included in appendix C.

3.2. Separated experiments

We first carry out a set of preliminary tests on the QG system only. In these experiments, we examine the accuracy of the learned forcings offline (that is, without rolling out trajectories) while separating the two steps in our multiscale prediction process. In particular, we examine the ability of networks to learn the coarsened and buildup prediction tasks, and check for the advantages we intuitively expect from the additional multiscale information.

In these experiments, we train neural networks separately to predict quantities between different scales. In particular we train coarsening networks which predict only the low-resolution components of the target forcing quantity while observing a high resolution state, and 'buildup' networks which work in the opposite direction, predicting higher-resolution forcing details with access to the low-resolution forcing. These illustrate some of the advantage provided by the additional information and measure

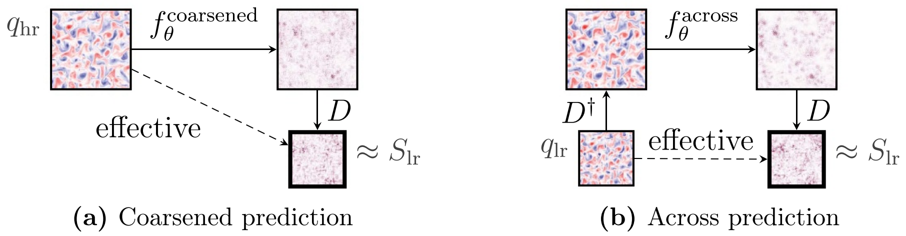


Figure 2. Coarsened vs. across separated prediction tasks. The networks referenced in equation (5) are combinations of an inner network f_θ with the fixed rescaling operators D, $D^\dagger$. The overall prediction is indicated with a dashed line.

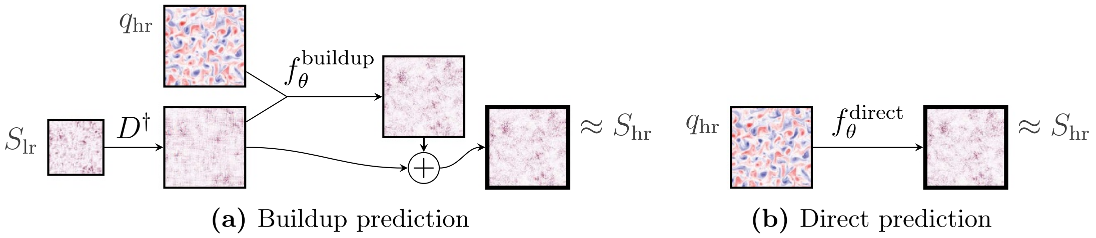


Figure 3. Buildup vs. direct separated prediction. The networks in equation (6) are combinations of the networks f_θ with the indicated fixed operations. In figure (a) f_θ predicts the details which are combined with S_{lr} from an oracle.

performance on snapshots only (offline testing) as these separated networks do not provide a fully-implementable closure model, since they require access to an oracle to provide the additional input features.

Because these experiments target the QG system the target quantity is the potential vorticity q. Each trained network receives a q input at the active (high resolution) simulation scale and predicts an S_q output at the target scale.

3.2.1. Coarsened prediction

We compare the task of predicting $S_{\mathrm{lr}} \triangleq D(S_{\mathrm{hr}})$ with access to high resolution information q_{hr} or restricted to low resolution q_{lr}. This provides an estimate of the advantage gained by predicting the target forcing with access to details at a scale finer than that of the network's output. We train two networks f_θ with the same architecture to perform one of two prediction tasks:

$$D \circ f_\theta^{\mathrm{coarsened}}(q_{\mathrm{hr}}) \approx S_{\mathrm{lr}} \quad \text{and} \quad D \circ f_\theta^{\mathrm{across}} \circ D^\dagger(q_{\mathrm{lr}}) \approx S_{\mathrm{lr}}. \tag{5}$$

To ensure that the convolution kernels process information at the same spatial size, and differ only in the spectral scales included, we first upsample all inputs to the same fixed size using a spectral fine-scaling operator $D^\dagger$ described in appendix D. The full prediction process including the re-sampling operators is illustrated in figure 2.

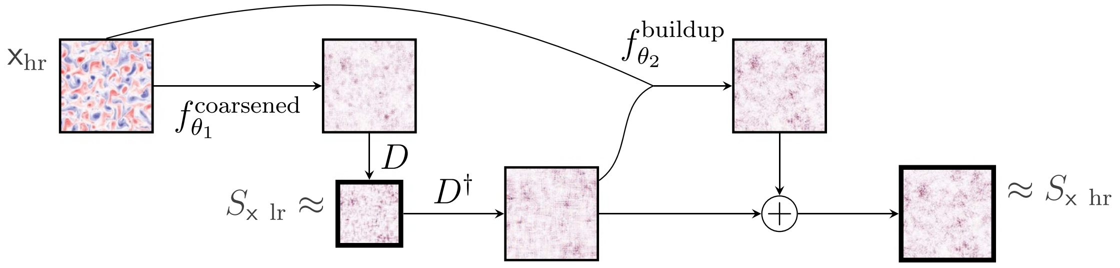
3.2.2. Buildup prediction

We also test a prediction problem in the opposite direction, predicting finer-scale details with access to lower-resolution predictions, similar to a learned super-resolution process used in recent generative modeling works [17, 18]. We train neural networks:

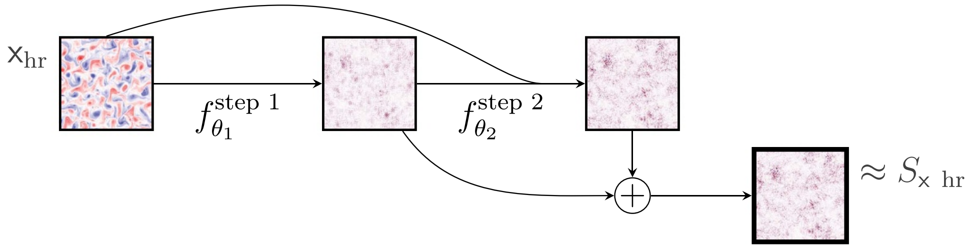
$$f_\theta^{\mathrm{buildup}}(q_{\mathrm{hr}}, D^\dagger(S_{\mathrm{lr}})) \approx S_{\mathrm{hr}} - D^\dagger(S_{\mathrm{lr}}) \quad \text{and} \quad f_\theta^{\mathrm{direct}}(q_{\mathrm{hr}}) \approx S_{\mathrm{hr}}, \tag{6}$$

where $S_{\mathrm{hr}} - D^\dagger(S_{\mathrm{lr}})$ are the details of S_{hr} which are not reflected in S_{lr}. The additional input S_{lr} is given by an oracle using ground truth data in the training and evaluation sets.

This experiment estimates the value in having a high-quality, higher-confidence prediction S_{lr}, in addition to q_{hr}, when predicting the details of S_{hr}. That is, the experiment estimates the value in starting the prediction of S_{hr} by first locking in a coarse-grained version of the target, and separately enhancing it with finer-scale features. The two prediction tasks are illustrated in figure 3.



(a) Full combined prediction flow



(b) Structure of the baseline for the combined prediction task

Figure 4. Flow for the combined prediction experiments. For the full combined flow each network f_θ is trained sequentially, and earlier weights are frozen and used to train networks later in the pipeline. The overall flow no longer requires an oracle for any additional inputs. Note also that the networks in the baseline flow in figure (b) have the same residual prediction structure. The baseline network is trained end-to-end and differs only in that it misses the additional supervision from the multiscale training. The variable x represents the differing scalar fields for the QG and KF systems.

3.3. Combined experiments

In these experiments, we combine the networks trained in the 'coarsened' and 'buildup' experiments, passing the coarsened prediction as an input to the buildup network. This removes the oracle providing lower resolution predictions used to train the separate networks. In each test, we choose two scale levels and first predict a coarsened version of the subgrid forcing at the lowest resolution, then gradually enhance it with missing scales using the buildup process discussed above. These tests are carried out for both the QG and KF systems. The overall flow of this combined approach is illustrated in figure 4 along with the associated baseline architecture.

Because this configuration yields an implementable closure model, we perform our evaluations *online*. That is, while the networks are trained on snapshots, we evaluate the accuracy and stability of the forcing by rolling out multiple trajectories using the trained networks. As in the separated experiments we consider two network architectures, a 'large' architecture based on other works and a 'small' architecture with fewer layers and smaller convolution kernels. As discussed above, the small architecture in these experiments was chosen through an architecture search. These tests allow us to study some of the efficiency gains which are possible by building a multiscale prediction approach into the network architecture rather than requiring all relationships across scales to be learned.

For these experiments we retrain new neural networks building out the training pipeline sequentially. We first train the coarsening network and then fix its weights and use its outputs to train the subsequent buildup network. In this way, later networks see realistic inputs during training rather than unrealistically clean data from a training set oracle.

For a combined prediction across two scales lr and hr, we predict $S_{\mathsf{x}\ \mathrm{hr}}$ from only x_{hr} following the procedure below:

$$\begin{aligned} \tilde{S}_{\mathsf{x}\ \mathrm{lr}} &\triangleq D \circ f_{\theta_1}^{\text{coarsened}}\left(\mathsf{x}_{\mathrm{hr}}\right) \approx S_{\mathsf{x}\ \mathrm{lr}} \\ S_{\mathsf{x}\ \mathrm{hr}} &\approx f_{\theta_2}^{\text{buildup}}\left(\mathsf{x}_{\mathrm{hr}}, D^{\dagger}\left(\tilde{S}_{\mathsf{x}\ \mathrm{lr}}\right)\right) + D^{\dagger}\left(\tilde{S}_{\mathsf{x}\ \mathrm{lr}}\right). \end{aligned} \tag{7}$$

The quantity $\tilde{S}_{\mathsf{x}\ \mathrm{lr}}$ is an approximate neural network output used in subsequent predictions. Equation (7) composes the prediction tasks described in equations (5) and (6). See figure 4(a) for an illustration of the above flow. Each network f_θ is trained separately against either $S_{\mathsf{x}\ \mathrm{lr}}$ or $S_{\mathsf{x}\ \mathrm{hr}}$ as appropriate.

The networks are evaluated on their ability to produce a stable trajectory when rolled out, and their ability to correct the energy spectrum—adding and removing energy across spectral scales as needed to correct issues from coarse-graining. For these experiments, we examine the trends in the system's total kinetic energy across time and distributions in the spectral error of the system, computed by adding errors in the average kinetic energy spectrum of a trajectory. These measurements are distinct from those used to compare the quality of snapshots used for the offline tests conducted as part of the separated experiments.

4. Results

In this section we describe the results of our experiments and measurements. Details of the approach taken in each experiment are provided in section 3.

4.1. Separated experiments

For both the coarsened and buildup prediction tasks, we train three neural networks. Once trained, we evaluate their performance on a held out evaluation set measuring performance with three metrics: a mean squared error (MSE), a relative ℓ_2 loss, and a relative ℓ_2 of the spectra of the predictions.

The MSE is a standard MSE evaluated over each sample and averaged. The other two metrics measure relative errors, the first computes this as a relative ℓ_2 error directly over the forcing, and the second computes the relative error across all scales in the isotropic power spectrum denoted *sp*. These measures are derived from previous work evaluating neural network parameterizations [9] where they were called $\mathcal{L}_{\mathrm{rmse}}$ and $\mathcal{L}_{\mathrm{S}}$. The other metrics used in that work collapse to trivial values for deterministic models and are not computed. These metrics are defined as:

$$\text{Rel } \ell_2 \triangleq \frac{||S-\tilde{S}||_2}{||S||} \qquad \text{and} \qquad \text{Rel Spec } \ell_2 \triangleq \frac{||\mathrm{sp}\left(S\right)-\mathrm{sp}\left(\tilde{S}\right)||_2}{||\mathrm{sp}\left(S\right)||_2} \tag{8}$$

where S is the true target forcing, $\tilde{S}$ is a neural network prediction being evaluated, and *sp* is the isotropic power spectrum. See `calc_ispec` in PyQG for calculation details [15]. Lower values in each of these three metrics reflect a more accurate parameterization. Comparing relative quality across them reflects either absolute errors, or relative errors in state space or in the energy distribution reflected by the spectrum. Each of these three metrics is averaged across the same batch of 1024 samples selected at random from the set of held out trajectories in the evaluation set for each of the three independently initialized and trained networks. We report these results in the tables described below.

Table 1 shows the results for the coarsened prediction experiments, comparing against 'across' prediction which accesses only coarse-scale information. In these results we observe an advantage to performing the predictions with access to higher-resolution data (the 'coarsened' columns), suggesting potential advantages and a decrease in uncertainty in such predictions.

Results for experiments examining prediction in the opposite direction—predicting a high-resolution forcing with access to a low-resolution copy of the target from an oracle—are included in table 2. We also observe an advantage in this task from having access to the additional information. The low resolution input in the buildup experiments yields lower errors on average at evaluation. This advantage is greater when the additional input is closer in scale to the target output. The predictions building up from 96 × 96 to 128 × 128 have lower errors than those which access an additional 64 × 64 input. This is not unexpected given that the input with nearer resolution resolves more of the target value, leaving fewer details which need to be predicted by the network.

The results for both separated experiments (those reported in tables 1 and 2) for the MSE metric are illustrated in figure 5. The points in this figure represent the average MSE across the 1024 samples grouped for each trained network, illustrating the variation due to the training process.

4.2. Combined experiments

We also carry out tests comparing the performance of networks using our multiscale prediction approach to a network predicting only at one scale throughout. Figure 4 illustrates the overall flows of these networks. In the combined task, each independent cross-scale network is trained separately in phases. Earlier networks in the pipeline are trained and have their weights frozen and these are used to produce inputs during the training of networks later in the pipeline. The baseline for this configuration is trained end-to-end and has the same residual structure and number of weights as the multiscale network, but without any enforced predictions across scales.

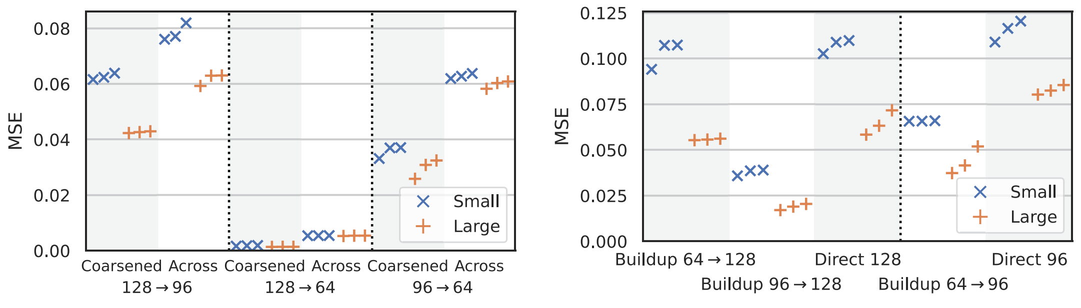
This setting produces a trained parameterization network which can be applied to a real simulation and tested online. As a result, we evaluate these networks by rolling out trajectories from a held out test set. These are compared against a reference trajectory which was originally produced at a 'true' resolution. Because these systems are chaotic we do not expect to reproduce exact states in these trajectories

Table 1. Evaluation results for coarsened vs. across generation. In all metrics, lower is better. The numbers in the first row of the table heading show the different scales involved in both prediction tasks. The results contributing to the MSE averages in this table are illustrated in figure 5(a). For each metric, the second row provides the standard deviation across all samples from all networks.

		128 → 96		128 → 64		96 → 64	
NN Size	Metric	Coarsened	Across	Coarsened	Across	Coarsened	Across
Small	MSE	0.063	0.078	0.002	0.005	0.036	0.063
	σ	0.0106	0.0130	0.0003	0.0009	0.0055	0.0089
	Rel ℓ_2	0.330	0.368	0.422	0.797	0.360	0.478
	σ	0.0129	0.0161	0.0264	0.0439	0.0189	0.0222
	Rel Spec ℓ_2	0.145	0.175	0.201	0.626	0.168	0.260
	σ	0.0242	0.0240	0.0338	0.0634	0.0259	0.0324
Large	MSE	0.043	0.062	0.001	0.005	0.030	0.060
	σ	0.0068	0.0096	0.0002	0.0009	0.0051	0.0083
	Rel ℓ_2	0.268	0.322	0.371	0.784	0.324	0.461
	σ	0.0104	0.0143	0.0221	0.0438	0.0213	0.0220
	Rel Spec ℓ_2	0.092	0.138	0.159	0.636	0.131	0.268
	σ	0.0213	0.0269	0.0303	0.0594	0.0246	0.0279

Table 2. Evaluation results from buildup vs. direct experiments. In all metrics, lower is better. The numbers in the second row of the table heading show the different scales involved in both prediction tasks. The results contributing to the MSE averages in this table are illustrated in figure 5(b). For each metric, the second row provides the standard deviation across all samples from all networks.

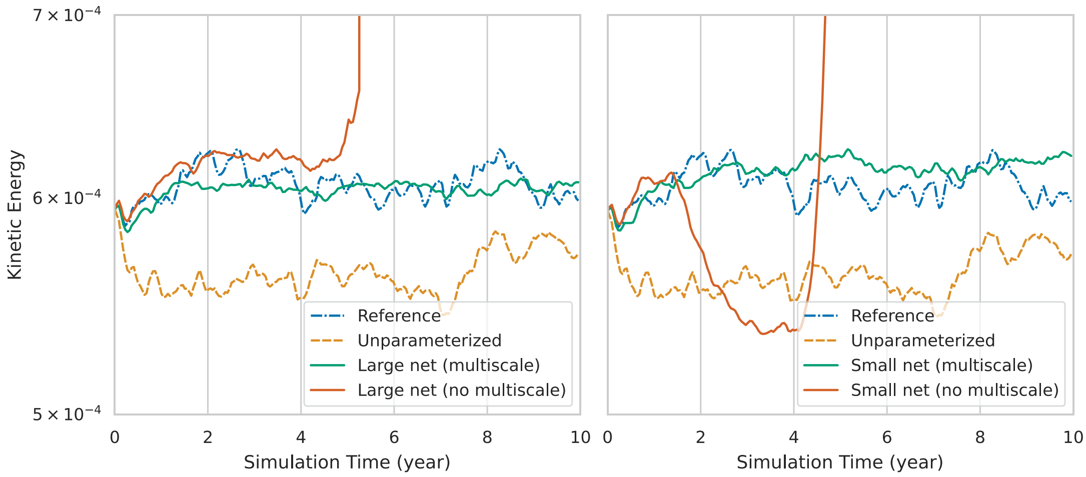
NN Size	Metric	Buildup 64 → 128	Buildup 96 → 128	Direct 128	Buildup 64 → 96	Direct 96
Small	MSE	0.103	0.038	0.107	0.066	0.115
	σ	0.0207	0.0076	0.0194	0.0114	0.0190
	Rel ℓ_2	0.319	0.193	0.324	0.255	0.334
	σ	0.0139	0.0075	0.0112	0.0071	0.0119
	Rel Spec ℓ_2	0.147	0.060	0.148	0.104	0.179
	σ	0.0295	0.0092	0.0212	0.0168	0.0243
Large	MSE	0.056	0.019	0.064	0.044	0.083
	σ	0.0101	0.0040	0.0123	0.0097	0.0128
	Rel ℓ_2	0.233	0.136	0.248	0.207	0.280
	σ	0.0075	0.0070	0.0139	0.0159	0.0091
	Rel Spec ℓ_2	0.070	0.030	0.085	0.069	0.109
	σ	0.0154	0.0073	0.0165	0.0179	0.0209



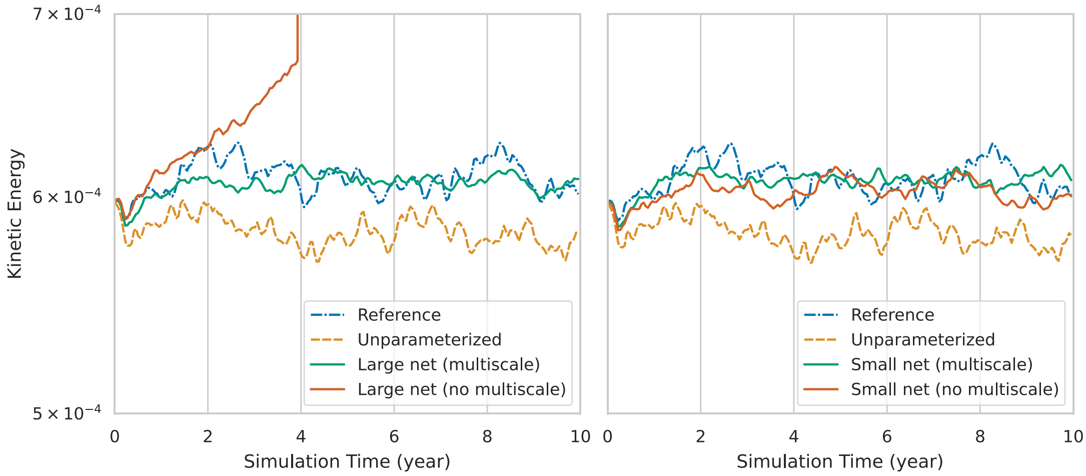
(a) Coarsened vs. across MSE (b) Buildup vs. direct MSE

Figure 5. Evaluation results from both of the separated experiments for the MSE metric. These are the same numbers which are reported as averages in tables 1 and 2. The plot here displays independent averages for each of the trained networks.

and we do not examine snapshot errors over long time horizons. Instead we compare statistical properties of these trajectories including their total kinetic energy (which gives a sense of stability), errors in the trajectory's energy spectrum (providing a sense of the quality of the parameterization), and for the KF system, the vorticity decorrelation time. Overall, we find that the multiscale approach improves the stability of the learned parameterizations while also permitting smaller neural network architectures to be used for these tasks. Results of our experiments for the QG and KF systems are described below.



(a) Scale 96—multiscale networks predict between 96 ← 64



(b) Scale 128—multiscale networks predict between 128 ← 96

Figure 6. Time evolution of mean kinetic energy for the QG system simulated at different scales. Trajectories are simulated for 16 held out trajectories using 4 independently-trained networks.

4.2.1. QG results

For the QG system we simulate the evolution of a held out trajectory and conduct tests over four independently trained neural networks and 16 held out trajectories for each network. For these tests we are concerned with two aspects: the stability of the system running with the neural network closure, and the quality of the resulting parameterization—in particular the extent to which it improves the energy spectrum of the trajectory.

Figure 6 shows variation in the mean kinetic energy over time (the average of $\frac{1}{2}||u||^2$ for the velocity u at each grid point). We intend this as a rough measure of the stability of the parameterized system providing a rough idea of whether the system is under- or over-energized which can lead to instability and eventual collapse in the system. The QG system is simulated at a grid size of 96×96 and 128×128 with separate sets of neural networks trained for each case. The simulations at a size of 96 have a greater range of unrealized dynamics leading to a more challenging closure problem. We note in particular that the trajectories which display instability are those using networks without the multiscale component. The smaller networks for the system at size 128 all display general kinetic energy stability; however, with only kinetic energy statistics this is difficult to distinguish from a network which applies no parameterization at all. The QG system is stable without a parameterization due to a fixed spectral filter which attenuates high frequencies, and the target parameterization values have zero mean which can in some cases lead a network to learn to apply no correction which still yields a stable trajectory. The 'reference' trajectory shows values obtained from the trajectory in the held out evaluation set. 'Unparameterized' values are computed by running the model without a parameterized network, showing the behavior of the uncorrected system. Other lines show values using the trained neural network parameterizations.

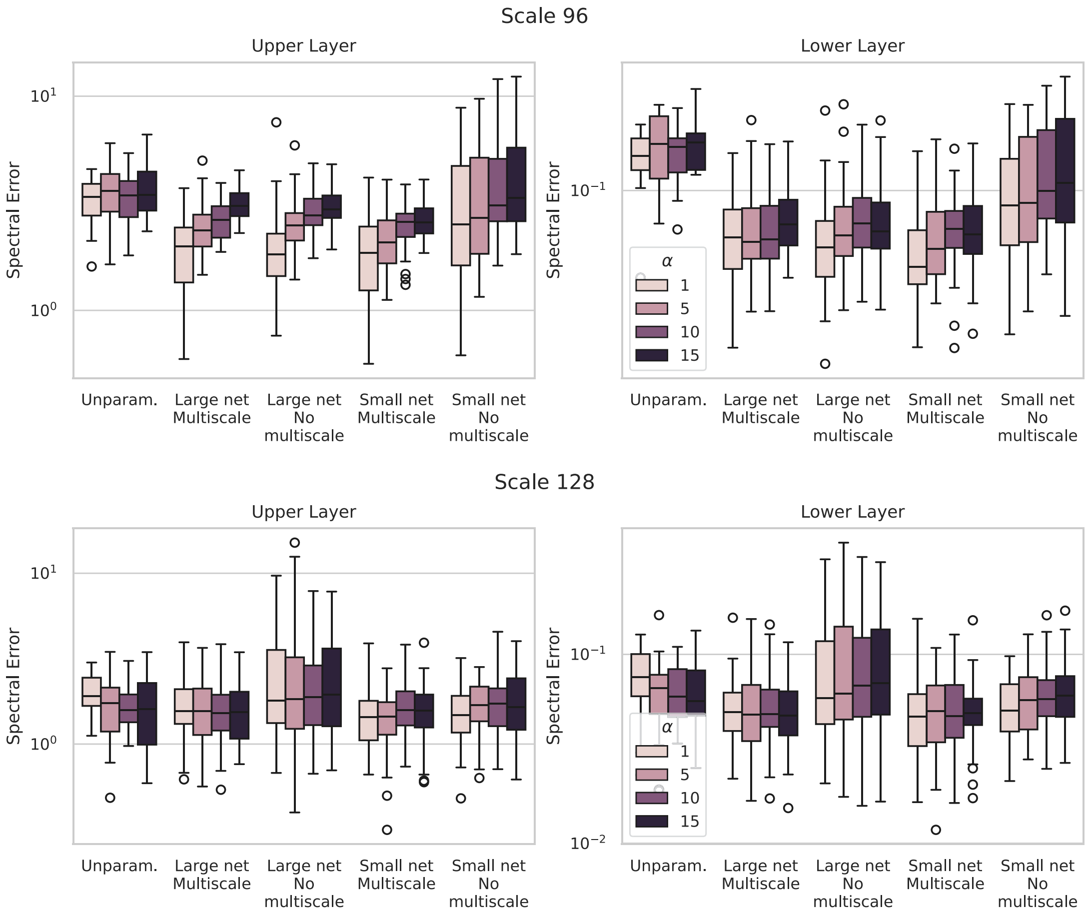


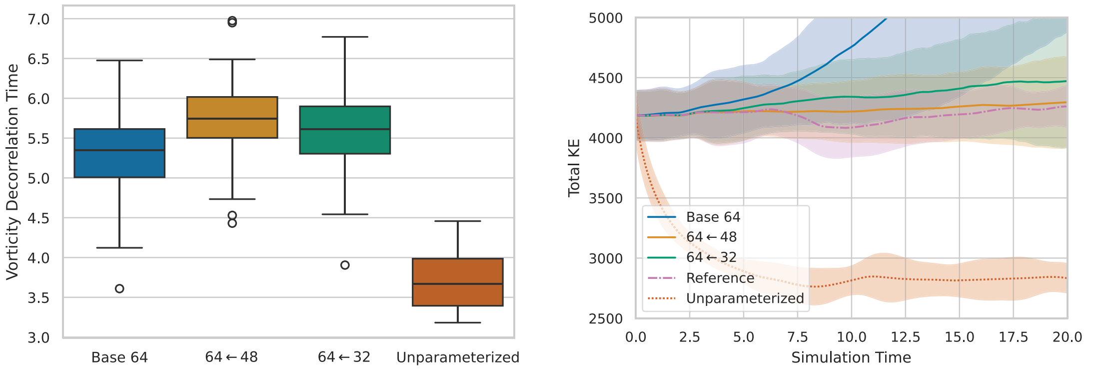
Figure 7. Spectral errors for trained architectures averaged over 5400 steps. Increasing α values reduce the sharpness of the QG model's filter, which in turn reduces built-in model stability. In each row, the separate panels show values for the two layers of the QG system.

To distinguish these cases we also examine the spectral error of these trajectories, distributions of which are plotted in figure 7. In these box plots, the midpoint line is the median of the samples, the box covers the first quartile through the third quartile, and the whiskers extend beyond the box by 1.5 times the inter-quartile range. Outlier points beyond the whiskers are plotted as open circles. For this, we compute the isotropic spectrum of the kinetic energy which divides the energy across wavenumbers k. We keep wavenumbers up to 2/3 of the Nyquist frequency, and compute the RMSE between the reference and predicted spectra [3]. We also run trajectories with varying values for α which controls the sharpness of the QG model's internal spectral filter. Higher values of α reduce the attenuation of higher frequencies, making the QG model less stable and making errors in the parameterization more evident over time. The results in this figure suggest that the multiscale training generally improves model performance. For large architectures on the system at scale 128, multiscale training reduces instabilities. For the results on scale 96—where more parameterization is required—adding multiscale training allows the small architecture to achieve the same results as the large architecture, allowing for a smaller more efficient network to yield a more accurate and stable parameterization.

4.2.2. KF results

In addition to our experiments on the QG model we also test our method on a KF system. Unlike the QG model, the KF system does not include an internal filter. This results in a simulation which can be highly unstable, and many of the learned parameterizations we trained and tested did not successfully complete a stable trajectory. To reduce this problem we modify our training procedure to inject Gaussian noise to inputs in order to encourage the learned parameterizations to be stable to corruptions in the system states. This noise has mean zero and a configurable scale parameter β. Further discussion of our experiments adjusting this parameter are included later in this section.

Using the best calibrated noise scales, we train a series of small architecture networks both with multiscale training and with the single scale baseline architecture. The large architectures displayed very



(a) Vorticity decorrelation times from online experiments for the KF system. A longer decorrelation time generally reflects a better parameterization.

(b) Kinetic energy evolution. Solid lines are means over all test trajectories and trained networks and shaded area is a 1σ region.

Figure 8. Results of online experiments for the KF system. Values are computed over a collection of 10 trained networks evaluated on 16 reference trajectories each.

high instability irrespective of the noise level. As a result further results on the KF in this section are produced for the small architecture only. An illustration of this noise calibration issue is included in appendix E. For each of these networks we roll out a series of trajectories online. Figure 8 shows the results of this online testing for the KF experiments. Results are averaged across 10 trained networks of each type and 20 held out test trajectories.

Figure 8(a) reports the observed decorrelation time in the vorticity channel ω of the KF system. The decorrelation time is the simulation time where the correlation coefficient between the predicted and reference trajectories drops below 0.5. Each test trajectory is run for a length of 20.0 units starting from a random offset in one of 16 held out reference trajectories, each of total length 70.0. We select shorter portions of these trajectories to reduce the computational cost of gathering online evaluation results. Starting at random offsets ensures the networks are evaluated on time segments both early and late in a longer simulation. Longer decorrelation times reflect a parameterization which better reproduced the statistical properties of the reference trajectory even though, due to the chaos in these systems, the trajectories will eventually decorrelate. We note that the multiscale runs have a longer decorrelation time than both the baseline network and the unparameterized system with the multiscale network using two scales closer in size having a slightly better performance.

Trends in the evolution of kinetic energy over time are plotted in figure 8(b). The lines in this figure are the mean kinetic energy across the test trajectories and networks and the shaded regions are a 1σ range. In these results, the multiscale networks remain more stable over time and show smaller variance at later time steps. All networks prevent the significant energy loss of the unparameterized trajectories even though the baseline network displays significant instability.

The results above were produced using separately chosen values of the training noise level β, one noise level for each network architecture and set of prediction scales. The values of β were set as a fraction of the empirical standard deviation of input values from the training set. Higher noise levels injected during training reduce instability but decrease parameterization accuracy, while lower noise levels fail to correct the problem of instability during online rollouts.

Figures 9 and 10 show measurements used in this calibration. We select our target noise level based in part on the offline network validation losses and online kinetic energy errors observed after training. We also show results for a system scale of size 64 as larger state sizes did not have sufficient unresolved dynamics to close on the KF system. Each network is provided the two velocity components u, v as well as a vorticity ω computed from these two.

Based on the results in figure 9 we see that higher noise levels reduce variance in offline validation quality, but can increase instabilities in online testing as evidenced by the kinetic energy errors in figure 10. As a result we selected a noise level of 0.035 for baseline runs, 0.075 for multiscale runs between scales $64 \leftarrow 48$ and 0.1 for $64 \leftarrow 32$ runs (the lowest kinetic energy error point in each curve). These values were used to train the networks used in the further KF experiments discussed earlier in this section.

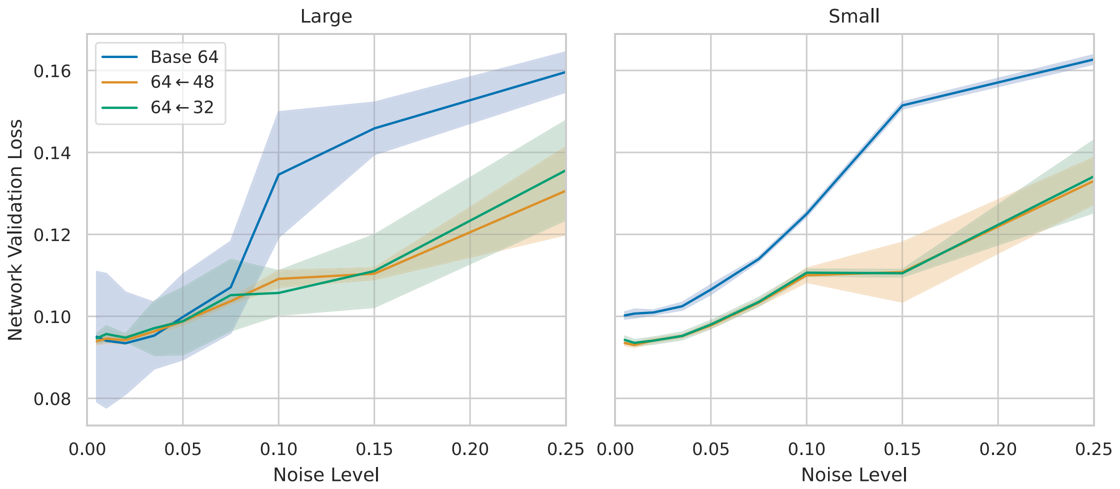


Figure 9. Offline noise calibration for the KF system. Solid lines are a mean validation loss observed during network training and shaded regions show a 1σ range. The large architecture shows significantly greater variance than the small architecture. Increasing noise scale during training increases the validation loss but can improve robustness against noise during online rollouts.

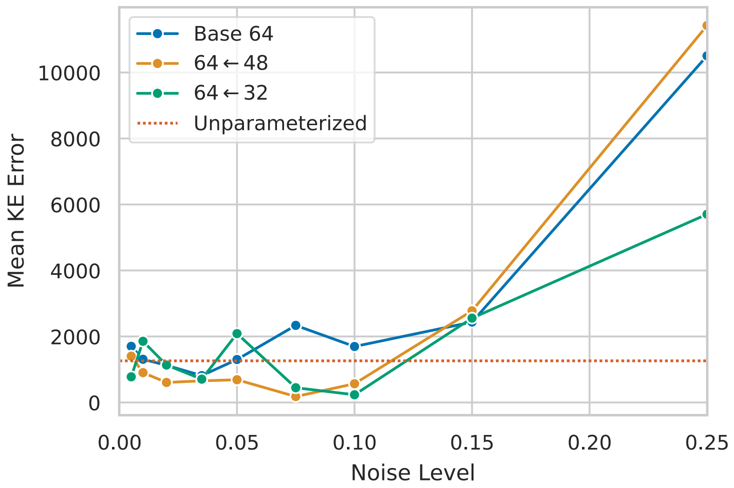


Figure 10. Online noise calibration for the KF system (small architecture only). Increasing noise scale eventually reduces the networks' ability to correct kinetic energy. We select the lowest online kinetic energy error point on each curve for each network as the noise scale used during training.

5. Conclusion

Our experiments in this work illustrate the potential advantages resulting from decomposing the subgrid forcing problem into one across scales. In particular, our results show that this decomposition improves stability and accuracy, especially for smaller network architectures. Such improvements could support the deployment of machine learning methods to tasks which are constrained by available computational resources, as is common in climate applications. Our results show improvements made possible by structuring prediction tasks to expose important structures of the task. For the fluid problems considered here, and in other tasks, a multiscale decomposition is natural and makes use of links between scales in the model dynamics, and better handles underlying uncertainty in the parameterization task.

We believe that our method is also applicable to problems beyond turbulence and climate modeling. It imposes few requirements on the target task, and only minimally increases computational cost relative to non-multiscale architectures of a similar size. Simulation tasks with chaotic dynamics, which involve dynamics operating across a range of scales, or in which fine-scale details can be overlaid onto perhaps more predictable coarse scales, may be amenable to a similar treatment. This would include a variety of fluid modeling tasks, and other simulation problems where predictions might be analogously decomposed across scales.

Future work In addition to using a multiscale decomposition in future learned parameterizations, future work could explore other applications of this approach. In particular, this decomposition could be advantageous for stochastic parameterizations, perhaps using a deterministic coarsened prediction as a foundation for later stochastic or generative outputs, or further integrating the multiscale prediction into the network architecture to realize greater efficiency.

Data availability statement

The JAX port of the QG model is available at: https://github.com/karlotness/pyqg-jax. The data that support the findings of this study are openly available at the following URL/DOI: https://github.com/karlotness/multiscale-closure.

Acknowledgments

This research received support from Schmidt Sciences through the M^{2}LInES project. This work was also supported by NYU IT High Performance Computing resources, services, and staff expertise.

Author contributions

Karl Otness was responsible for project planning, implementation of software and experiments, as well as data generation, analysis, and presentation.

Laure Zanna and Joan Bruna assisted with project conceptualization and planning and provided key support, guidance, and supervision throughout all phases of research and presentation.

All authors contributed to the preparation of this document.

Competing interests

The authors declare no competing interests.

Appendix A. Notation

Table 3 collects notation used throughout this document.

Table 3. Collected definitions of notation used throughout this document.

Symbol	Definition
x	A generic system state, the quantity whose evolution will be corrected by the trained networks. For our experiments this is the potential vorticity for the QG model, and velocity for the KF model.
S_{x}	The correction term applied to x. For our experiments this is a subgrid parameterization.
x_{true}	A reference sample generated at a very high resolution to provide a ground truth. Used only to compute target quantities through coarsening using operators C. Not used for training. See figure 1.
x_{lr} and x_{hr}	High and low resolution quantities, respectively. The subscripts 'hr' and 'lr' denote high and low resolution. Values can be ported between them using D and $D^{\dagger}$. See figure 1.
$\bar{\mathsf{x}}$	Alternate notation for $C(\mathsf{x})$
f_{θ}	A trained neural network. Superscript and subscript labels are used to distinguish independently initialized and trained networks.
C	System-dependent coarsening and filtering operators. These are used for processing x_{true} samples after generating them. See appendix D.1 for details.
D and $D^{\dagger}$	D is a simple coarsening operator porting values between high and low resolutions. $D^{\dagger}$ is its pseudoinverse which interpolates low resolution values to a higher resolution. See appendix D.2 for details.

Appendix B. Models

In this work we consider two models: a quasi-geostrophic model (QG) and a Kolmogorov flow (KF) model. Further information on these systems is provided below.

B.1. QG model

For our experiments we target the two-layer QG model implemented in PyQG which is a simplified approximation of fluid dynamics [15]. This model follows the evolution of a potential vorticity q, divided into two layers $q = [q_1, q_2]$. This system is pseudo-spectral and has periodic boundaries along the edges of each layer. The evolution of the quantities in Fourier space (indicated by a hat) is:

$$\frac{\partial \hat{q}_1}{\partial t} = -\widehat{J(\psi_1, q_1)} - ik\beta_1\hat{\psi}_1 + \widehat{\text{ssd}} \tag{9}$$

$$\frac{\partial \hat{q}_2}{\partial t} = -\widehat{J(\psi_2, q_2)} - ik\beta_2\hat{\psi}_2 + r_{\text{ek}}\kappa^2\hat{\psi}_2 + \widehat{\text{ssd}} \tag{10}$$

where $J(A,B) \triangleq A_xB_y - A_yB_x$, 'ssd' is a small scale dissipation, and the quantity ψ is related to q by:

$$\begin{bmatrix} -\left(\kappa^2 + F_1\right) & F_1 \\ F_2 & -\left(\kappa^2 + F_2\right) \end{bmatrix} \begin{bmatrix} \hat{\psi}_1 \\ \hat{\psi}_2 \end{bmatrix} = \begin{bmatrix} \hat{q}_1 \\ \hat{q}_2 \end{bmatrix}. \tag{11}$$

The values κ are the radial wavenumbers $\sqrt{k^2 + l^2}$ while k and l are wavenumbers in the zonal and meridional directions (the axis-aligned directions in our grid), respectively [3].

We use the 'eddy' configuration from [3] which sets the following values for model constants:

$$\begin{aligned} r_{\text{ek}} &= 5.787 \times 10^{-7} & F_1 &= \frac{1}{r_d^2(1+\delta)} \\ \delta &= \frac{H_1}{H_2} = 0.25 & F_2 &= \delta F_1 \\ \beta &= 1.5 \times 10^{-11} & W &= 10^6 \\ r_d &= 15000 & L &= 10^6 \end{aligned}$$

where H_1, H_2 are the heights of each of the two layers of q and r_d is a deformation radius. For more information on the model configuration, consult [3] and the documentation for the PyQG package.

We generate our data at a 'true' resolution on a grid of dimension 256 × 256 using the PyQG default third order Adams-Bashforth method for time stepping. We use a time step of $\Delta t = 3600$ generating 86400 steps from which we keep every eighth leaving 10800 per trajectory. Our training set consists of 100 such trajectories, and our evaluation set contains 10.

Each step produces a ground truth potential vorticity q_{true} along with a spectral time derivative $\partial\hat{q}_{\text{true}}/\partial t$. From these we apply our family of coarsening operators C (described in appendix D) to produce filtered and coarsened values $q_{\text{lr}} \triangleq C_{\text{lr}}(q_{\text{true}})$ at resolutions of 128 × 128, 96 × 96, and 64 × 64.

For each of these, we recompute spectral time derivatives in a coarsened PyQG model $\partial\hat{q}_{\text{lr}}/\partial t$, and we pass each time derivative to spatial variables and compute the target forcing for this scale:

$$S_{\text{lr}} = C_{\text{lr}}\left(\frac{\partial q_{\text{true}}}{\partial t}\right) - \frac{\partial q_{\text{lr}}}{\partial t}.$$

These forcings—at each of the three scales—along with the high resolution variables are stored in the training and evaluation sets for each step.

B.2. KF model

We also consider a KF system as implemented by the JAX-CFD package. Details of the model are included in its associated paper [16]. This is a configuration of an incompressible Navier–Stokes system with a forcing of $\sin(4y)\hat{x} - 0.1u$ where the subtracted term is a small velocity-dependent drag. This package implements a finite volume 2D Navier–Stokes system on a staggered grid (velocities in each direction are sampled on different edges of each finite volume cell).

$$\frac{\partial u}{\partial t} = -\nabla \cdot (u \otimes u) + \frac{1}{\text{Re}}\nabla^2 u - \frac{1}{\rho}\nabla p + \sin(4y)\hat{x} - 0.1u \qquad \text{with} \qquad \nabla \cdot u = 0$$

where $u(x,y,t)$ is the velocity field, p is the pressure, $\rho = 1$ is the constant density, Re is the configured Reynolds number, and the final two terms are the KF forcing [16].

We generate our ground truth data at a true resolution of 2048 × 2048 on a domain of size 4π on each side (double the default size of 2π). We configure the system to have a viscosity of $1/3500$ which, with the domain, produces a Reynolds number of 7000. States are ported to lower resolutions using the `downsample_staggered_velocity` function in the JAX-CFD package. This routine averages velocities across the edges of the finite volume cells being aggregated during the coarsening, and accounts for the staggering of the destination grid.

Reference trajectories are computed with a time step of 4.38×10^{-4} across 70.0 simulation time units. The time step was chosen using JAX-CFD's `stable_time_step` routine. The first 91266 steps are skipped to warm up the system and every 15th subsequent step is saved for training, validation, and test datasets. Each stored trajectory contains 4564 saved snapshots. During evaluation, models begin at one of these stored snapshots and run using the same time step used during data generation. Every 15th step is again used in performance calculations matching the separation of steps stored in the reference trajectories. We generate 128 trajectories for training, 3 for validation, and 16 for evaluation.

Appendix C. Network architecture and training

The network architectures used in this paper are all feedforward convolutional neural networks. The structure and training parameters of each network vary with the target system and were adjusted for the online combined experiments using the results of the offline separated experiments.

C.1. Separated experiment architectures

For our *separated* experiments on the QG system we use two different feedforward CNN architectures from previous work without batch norm [7]. We take the architectural parameters from this work as our default 'small' architecture, while the 'large' architecture for these experiments roughly doubles the size of each convolution kernel. This produces the architectures listed in Table 4. We use ReLU activations between each convolution. Each convolution is performed with periodic padding, matching the boundary conditions of the system. All convolutions are with bias. The input and output channel counts are determined by the inputs of the network. For the QG system each input has two layers, each of which is handled as a separate channel. Quantities for the KF system have only a single layer each. These parameters are adjusted for each task to accommodate the inputs and make the required predictions. We implement our networks with Equinox [19].

We train each network with the Adam optimizer [20] as implemented in Optax [21]. The learning rate is set to a constant depending on architecture size: the small networks use 5×10^{-4}, while the large networks use 2×10^{-4}. The networks are trained to minimize MSE loss. Large chunks of 10850 steps are sampled with replacement from the dataset which is pre-shuffled uniformly. Then each of these chunks is shuffled again and divided into batches of size 256 without replacement. One epoch consists of 333 such batches. We train the small networks for 132 epochs, and the large networks for 96 epochs. We store the network weights which produced the lowest training set loss and use these for evaluation.

For all input and target data, we compute empirical means and standard deviations and standardize the overall distributions by these values before passing them to the network. The means and standard deviations from the training set are used in evaluation as well.

C.2. Combined experiment architectures

For our combined experiments, which carry out online tests on both the QG and KF systems, we made some modifications to the tested architectures in order to explore the efficiency improvements which could be realized by our multiscale approach. We kept the same 'large' architecture as was used in the separated experiments, but carried out an architecture search to select a 'small' architecture.

Because each overall parameterization combines two sub-networks (see figure 4) we describe each network by giving a description of the architectures of each component network. These are described by the sizes of the convolution kernels ('pure small (psm),' 'small (sm),' 'pure medium (pmd),' and 'medium (md),' respectively) and the number of convolution layers, either 4 or 8. The 'large' architecture described in table 4 would be described as 'md8' and the combination of two of these is 'md8md8.'

Using these options, four networks of each architecture were trained on the QG system and tested online for the resulting spectral errors. The results of each of the two network scales are reported in figure 11. These were ranked by increased spectral errors separately for each layer, then each architecture choice was ranked by the worst of these two layer ranks.

The winning 'small' architecture for both QG scales was 'psm4pmd8' and the 'large' architecture is 'md8md8.' These architectures were also used in experiments on the KF system. Details of these parameters are provided in table 5.

The training parameters also varied from the separated experiments due to the new end-to-end training configuration. For the QG system training was conducted using the Adam optimizer following a cosine annealing schedule with one epoch of linear learning rate warmup. The 'md' and 'pmd' networks were trained for 50 epochs at a learning rate of 0.0004 while the 'psm' network was trained for 100 epochs (with 374 batches per epoch) with a learning rate of 0.001. All batches had size 64.

Table 4. Architecture specifications for each neural network used in the separated experiments. Convolution kernel sizes vary between the architecture sizes. The channel counts are adjusted to accommodate the inputs and outputs of each task.

Conv. Layer	Chans. Out	Small Kernel Size	Large Kernel Size
1	128	$(5,5)$	$(9,9)$
2	64	$(5,5)$	$(9,9)$
3	32	$(3,3)$	$(5,5)$
4	32	$(3,3)$	$(5,5)$
5	32	$(3,3)$	$(5,5)$
6	32	$(3,3)$	$(5,5)$
7	32	$(3,3)$	$(5,5)$
8	out layers	$(3,3)$	$(5,5)$

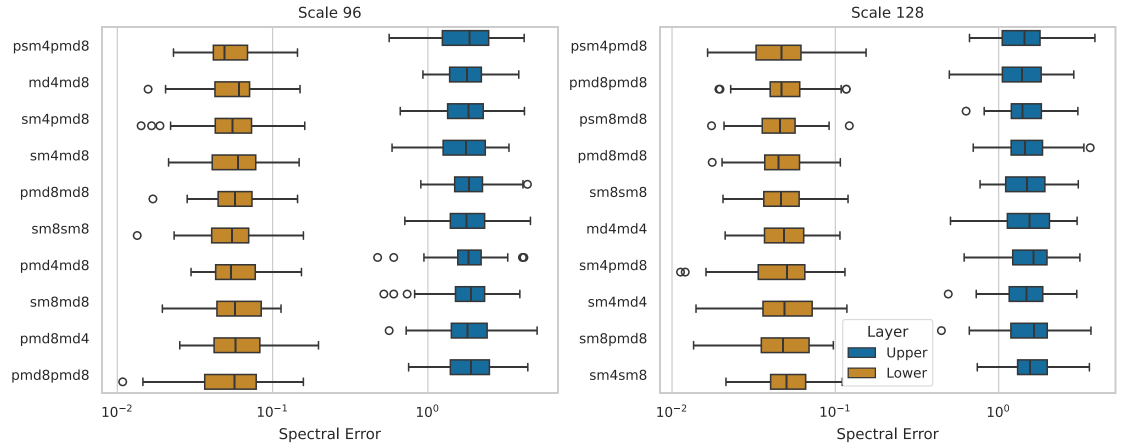


Figure 11. Architecture selection by scale size on QG system for the combined experiments.

Table 5. Architecture specifications for each neural network used in the combined experiments including those selected through the architecture search.

Conv. Layer	md8		psm4		pmd8	
	Chans. out	Kernel	Chans. out	Kernel	Chans. out	Kernel
1	128	$(9,9)$	128	$(3,3)$	128	$(5,5)$
2	64	$(9,9)$	64	$(3,3)$	64	$(5,5)$
3	32	$(5,5)$	32	$(3,3)$	32	$(5,5)$
4	32	$(5,5)$	out layers	$(3,3)$	32	$(5,5)$
5	32	$(5,5)$			32	$(5,5)$
6	32	$(5,5)$			32	$(5,5)$
7	32	$(5,5)$			32	$(5,5)$
8	out layers	$(5,5)$			out layers	$(5,5)$

For the KF system, training was carried out using the Adam optimizer with $\epsilon = 0.001$ (modified from the default). These networks followed the same cosine annealing with warmup schedule as the QG systems but had an ending learning rate of 0.0001 and a peak learning rate of 7.5×10^{-4} for 150 epochs with batches of size 32. Each epoch consisted of 374 batches. Each training run selected the network with the best validation loss.

Appendix D. Coarsening operators

In this work we make use of two families of coarsening operators to transform system states across scales. The first, denoted C, is used when generating our data. This operator is applied to the 'true' resolution system outputs q_{true} and $\partial q_{\text{true}}/\partial t$ to produce training and evaluation set samples as well as target forcings S. The second operator D (with associated fine-scaling $D^{\dagger}$) is applied as a part of each prediction task to adjust scales around the neural networks as needed. These are the operators referenced in figures 2 and 3.

Each of these operators is built around a core spectral truncation operation, $\mathcal{D}$. For an input resolution hr and an output resolution lr, this operator truncates the 2D-Fourier spectrum to the wavenumbers which are resolved at the output resolution, then spatially resamples the resulting signal for the target size lr. These operators also apply a scalar multiplication to adjust the range of the coarsened values. We define a ratio $\rho \triangleq \text{hr}/\text{lr}$.

D.1. Data filtering

The data filtering operator C for the QG system is 'Operator 1' as described in [3]. It is a combination of the truncation operator $\mathcal{D}$ with a spectral filter $\mathcal{F}$

$$C \triangleq \rho^{-2} \cdot \mathcal{F} \circ \mathcal{D}$$

where the filter $\mathcal{F}$ acts on the 2D-Fourier spectrum of the truncated value. $\mathcal{F}$ is defined in terms of the radial wavenumber $\kappa = \sqrt{k^2 + l^2}$ where k and l are the wavenumbers in each of the two dimensions of the input. For an input $\hat{v}_\kappa$ at radial wavenumber κ we define:

$$\mathcal{F}(\hat{v}_\kappa) = \begin{cases} \hat{v}_\kappa & \text{if } \kappa \leq \kappa^c \\ \hat{v}_\kappa \cdot e^{-23.6\alpha(\kappa - \kappa^c)^4 \Delta x_{\text{lr}}^4} & \text{if } \kappa > \kappa^c \end{cases}$$

where $\Delta x_{\text{lr}} \triangleq L/\text{lr}$ (L is a system parameter; see appendix B.1 for details), and $\kappa^c \triangleq (0.65\pi)/\Delta x_{\text{lr}}$ is a cutoff wavenumber where decay begins.

For the KF system, we used the routine `jax_cfd.base.resize.downsample_staggered_velocity` from the JAX-CFD package which computes means of the velocity values along a specific face of control volumes composed of groups of grid squares.

D.2. Rescaling operator

For scale manipulations as part of our learned model we make use of a scaled spectral truncation operator. We define a coarsening operator D as well as a fine-scaling operator $D^{\dagger}$:

$$D \triangleq \rho^{-2}\mathcal{D} \qquad \text{and} \qquad D^{\dagger} \triangleq \rho^{2}\mathcal{D}^{T}. \tag{12}$$

Note that $D^{\dagger}$ is a right side inverse $DD^{\dagger} = I$, and that $D^{\dagger}$ is the pseudoinverse $D^{\dagger} = D(DD^{T})^{-1}$ because $\mathcal{D}\mathcal{D}^{T} = I$. This operator omits the filtering $\mathcal{F}$ performed as part of coarsening operator C to avoid numerical issues when inverting the additional spectral filtering. This operator was used with both QG and KF systems.

Appendix E. Extended online results

Extended KF calibration results for large architectures are presented in figures 12 and 13. The large architectures had persistent stability problems on the KF system that could not be resolved with added noise during training.

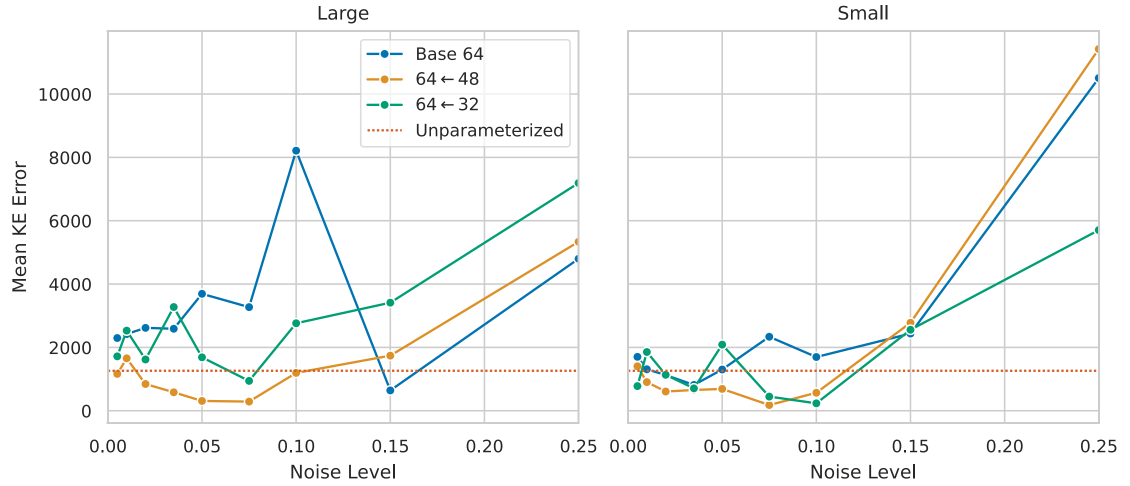


Figure 12. An extended version of figure 10 including results for the large network architecture. These architectures show significant instability, particularly for the baselines, even with significant added noise.

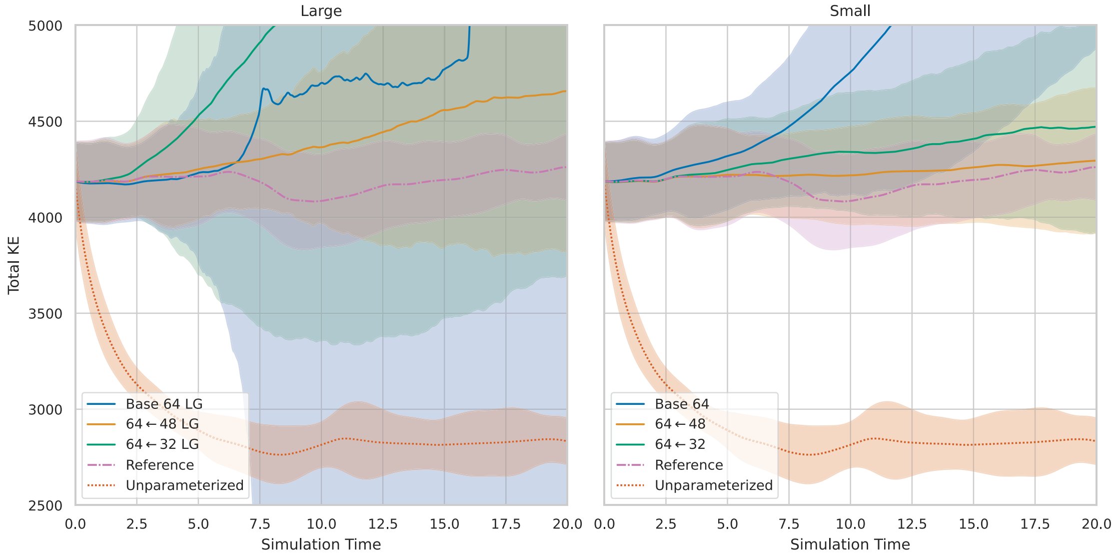


Figure 13. An extended version of figure 8(b) including results for the large network architecture which produced generally unstable results even with attempts to improve stability. However even here it appears that the added information from a scale of size 48 may help improve network stability.

ORCID iDs

Karl Otness 0000-0001-8534-2648
Laure Zanna 0000-0002-8472-4828
Joan Bruna 0000-0002-2847-1512

References

[1] Frezat H, Sommer J L, Fablet R, Balarac G and Lguensat R 2022 A posteriori learning for quasi-geostrophic turbulence parametrization *J. Adv. Model. Earth Syst.* **14** e2022MS003124
[2] Fox-Kemper B, Bachman S, Pearson B and Reckinger S 2014 Principles and advances in subgrid modelling for eddy-rich simulations *CLIVAR Exchanges (WGOMD Workshop on High Resolution Ocean Climate Modeling)* vol 19 pp 42–46
[3] Ross A, Li Z, Perezhogin P, Fernandez-Granda C and Zanna L 2023 Benchmarking of machine learning ocean subgrid parameterizations in an idealized model *J. Adv. Model. Earth Syst.* **15** e2022MS003258
[4] Fox-Kemper B *et al* 2019 Challenges and prospects in ocean circulation models *Front. Mar. Sci.* **6** 65
[5] Smagorinsky J 1963 General circulation experiments with the primitive equations: I. The basic experiment *Mon. Weather Rev.* **91** 99–164
[6] Zanna L and Bolton T 2020 Data-driven equation discovery of ocean mesoscale closures *Geophys. Res. Lett.* **47** e2020GL088376
[7] Guillaumin A P and Zanna L 2021 Stochastic-deep learning parameterization of ocean momentum forcing *J. Adv. Model. Earth Syst.* **13** e2021MS002534

[8] Maulik R, San O, Rasheed A and Vedula P 2019 Subgrid modelling for two-dimensional turbulence using neural networks *J. Fluid Mech.* **858** 122–44

[9] Perezhogin P, Zanna L and Fernandez-Granda C 2023 Generative data-driven approaches for stochastic subgrid parameterizations in an idealized ocean model *J. Adv. Model. Earth Syst.* **15** e2023MS003681

[10] Jansen M F, Adcroft A, Khani S and Kong H 2019 Toward an energetically consistent, resolution aware parameterization of ocean mesoscale eddies *J. Adv. Model. Earth Syst.* **11** 2844–60

[11] Yang L, Zhang Z, Song Y, Hong S, Xu R, Zhao Y, Zhang W, Cui B and Yang M-H 2023 Diffusion models: a comprehensive survey of methods and applications *ACM Comput. Surv.* **56** 11

[12] Song Y, Sohl-Dickstein J, Kingma D P, Kumar A, Ermon S, and Poole B, 2021 Score-based generative modeling through stochastic differential equations (ICLR) (arXiv:2011.13456)

[13] Otness K 2025 Source code for multiscale-closure (https://doi.org/10.5281/zenodo.17645365)

[14] Otness K 2024 pyqg-jax: v0.8.1 (https://doi.org/10.5281/zenodo.10719906)

[15] Abernathey R *et al* 2022 pyqg: v0.7.2 (https://doi.org/10.5281/zenodo.6563667)

[16] Kochkov D, Smith J A, Alieva A, Wang Q, Brenner M P and Hoyer S 2021 Machine learning–accelerated computational fluid dynamics *Proc. Natl Acad. Sci.* *118* e2101784118

[17] Singer U *et al* 2023 Make-a-video: text-to-video generation without text-video data ICLR (arXiv:2209.14792)

[18] Ho J *et al* 2022 Imagen video: high definition video generation with diffusion models (arXiv:2210.02303)

[19] Kidger P and Garcia C 2021 Equinox: neural networks in JAX via callable PyTrees and filtered transformations *Differentiable Programming Workshop at Neural Information Processing Systems 2021* (arXiv:2111.00254)

[20] Kingma D P and Ba J 2015 Adam: a method for stochastic optimization *Int. Conf. on Learning Representations* (arXiv:1412.6980)

[21] DeepMind *et al* 2020 The DeepMind JAX ecosystem (available at: https://github.com/google-deepmind)