Data-driven multiscale modeling of subgrid parameterizations in climate models

Karl Otness, Laure Zanna & Joan Bruna
Courant Institute of Mathematical Sciences
New York University
karl.otness@nyu.edu

Abstract

Subgrid parameterizations, which represent physical processes occurring below the resolution of current climate models, are an important component in producing accurate, long-term predictions for the climate. A variety of approaches have been tested to design these components, including deep learning methods. In this work, we evaluate a proof of concept illustrating a multiscale approach to this prediction problem. We train neural networks to predict subgrid forcing values on a testbed model and examine improvements in prediction accuracy that can be obtained by using additional information in both fine-to-coarse and coarse-to-fine directions.

1 Introduction

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 [4]. 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 [14, 5]. A variety of approaches to designing these parameterizations have been tested, ranging from hand-designed formulations [16], to modern machine learning with genetic algorithms [14], or neural networks trained on collected snapshots [17, 7, 12, 13], or in an online fashion through the target simulation [6].

In this work we examine the impact of decomposing the problem of predicting subgrid forcings into prediction problems across scales. The problem of learning subgrid forcing 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 [9], 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. Explicitly including this structure as part of a deep learning approach may help regularize the learned closure models and support learning in regimes with limited training data or in the presence of underlying uncertainty. We explore the impact of this decomposition, providing proof of concept results illustrating the potential of imposing this prediction structure on a simple fully-convolutional neural network closure model.

2 Approach

We consider the problem of learning subgrid forcings for an idealized fluid model. In particular we study a two layer quasi-geostrophic model as implemented in PyQG [1] which we have ported to JAX[1] [3]. In this model the variable of interest is the potential vorticity q which evolves in two layers each with two dimensions and periodic boundary conditions. The model can be evaluated with a configurable grid resolution and states can be ported to lower resolutions by coarse-graining and filtering. Further details of this model are included in Appendix A.

To generate ground truth data, we run the model at a very high ("true") resolution. This produces trajectories $q_{\text{true}}(t)$ and time derivatives $\partial q_{\text{true}}(t)/\partial t$. Next we generate training data at a high resolution by applying a coarsening and filtering operator C giving variables $\bar{q} \triangleq C(q)$. Given nonlinearities in the model, this coarsening does not commute with the dynamics of the model. To correct for this we must apply a subgrid forcing term S:

$$S \triangleq \overline{\frac{\partial q}{\partial t}} - \frac{\partial \bar{q}}{\partial t}. \tag{1}$$

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

We continue this process, introducing another downscaling[2] operator D and upscaling D^+. Taking $q_{\text{hr}} \triangleq \bar{q}$ as our high resolution samples, we produce low resolution samples $q_{\text{lr}} \triangleq D(q_{\text{hr}})$ and $S_{\text{lr}} \triangleq D(S)$. For any of these quantities v we have a decomposition $v = D^+ D v + \text{details}(v)$, where details$(v)$ 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 try to update hr to match the "true" resolution.

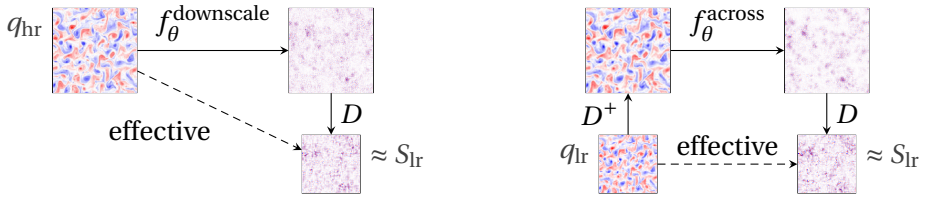
Just as predicting S from q_{true} is fully deterministic, while predicting it from q_{hr} involves uncertainty, we anticipate a similar trend to hold for $D(S)$. In other words, predicting $D(S)$ from q_{hr} should be easier than predicting $D(S)$ directly from q_{lr}. Then, using this coarse-grained prediction $D(S)$ as a foundation, we can learn to predict only the missing details and add them. This process splits the problem of predicting S into two phases: (1) a "downscale" prediction to form $D(S)$, and (2) a "buildup" prediction combining q_{hr} and $D(S)$ to predict S, 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 the "downscale" and "buildup" processes discussed above against baselines without the additional information. In our experiments on the quasi-geostrophic model, data is generated at a "true" resolution of 256×256, and high and low resolutions are selected from 128×128, 96×96, and 64×64. When describing the neural network tasks below, subscripts hr and lr stand in for one of these three resolutions. These scales were chosen so that the system requires closure (there are sufficient dynamics below the grid-scale cutoff), but does not diverge [14]. In the experiments that follow we test all combinations of distinct high and low resolutions. See Appendix A for model parameters and further details on the quasi-geostrophic data generation.

[1] The ported QG model is available at `https://github.com/karlotness/pyqg-jax/`

[2] We use "downscale" and "downscaling" to refer to *coarsening* a target variable, removing finer scales.



(a) Downscale prediction **(b)** Across prediction

Figure 1: Downscale vs. across prediction tasks. The networks referenced in Equation 2 are combinations of an inner network f_θ with the fixed rescaling operators D, D^+. The overall prediction is indicated with a dashed line.

Our experiments are divided into two categories: a first set of separated experiments, where the predictions are in one direction across scales (either down or up scale alone), the neural networks are trained and evaluated separately, and all required inputs are provided by an oracle backed by a ground truth data set; and a second set of combined experiments where these networks are composed, eliminating the need for the data set oracle.

For all experiments we train a feedforward convolutional neural network to perform the prediction task, three copies of each network. These neural networks have one of two architectures, a "small" architecture used in related research [7] and a "large" architecture with larger convolution kernels. Details of these experiments are provided below, results are included in Section 4, and information on the network architectures and the training procedure are included in Appendix B.

3.1 Separated Experiments

In these experiments, we train neural networks separately to predict quantities between different scales. In particular we train "downscale" 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.

Downscale Prediction

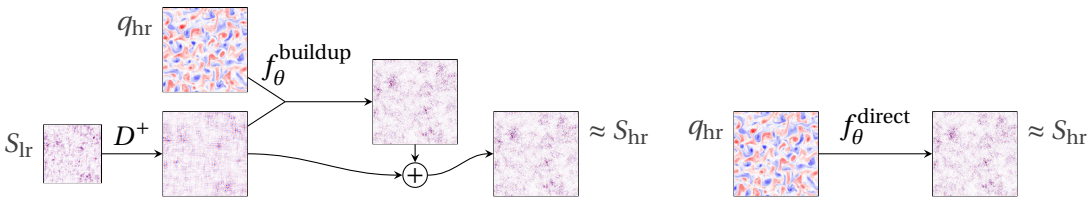
We compare the task of predicting $S_{\text{lr}} \triangleq D(S_{\text{hr}})$ with access to high resolution information q_{hr} or restricted to low resolution q_{lr}. This provides an estimate of the advantage gained by predicting the target forcing with access to details at a scale finer than that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

$$D \circ f_\theta^{\text{downscale}}(q_{\text{hr}}) \approx S_{\text{lr}} \qquad \text{and} \qquad D \circ f_\theta^{\text{across}} \circ D^+(q_{\text{lr}}) \approx S_{\text{lr}}. \tag{2}$$

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 upscaling operator D^+ described in Appendix C. The full prediction process including the re-sampling operators is illustrated in Figure 1 and experimental results are included in Table 1 discussed below.

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



(a) Buildup prediction **(b)** Direct prediction

Figure 2: Buildup vs. direct prediction. The networks in Equation 3 are combinations of the networks f_θ with the indicated fixed operations. In Figure 2a f_θ predicts the details which are combined with S_{lr} from an oracle.

recent generative modeling works [15, 8]. We train neural networks:

$$f_\theta^{\text{buildup}}\big(q_{\text{hr}}, D^+(S_{\text{lr}})\big) \approx S_{\text{hr}} - D^+(S_{\text{lr}}) \qquad \text{and} \qquad f_\theta^{\text{direct}}(q_{\text{hr}}) \approx S_{\text{hr}}\,, \tag{3}$$

where $S_{\text{hr}} - D^+(S_{\text{lr}})$ 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 2 and results are included in Table 2, discussed below.

3.2 Combined Experiments

In these experiments, we combine the networks trained in the "downscale" and "buildup" experiments, passing the downscale 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 at least 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. We test all valid subsets of our three scale levels. Results for these experiments are included in Table 3.

For these experiments we retrain new neural networks building out the training pipeline sequentially. That is, we first train the first downscale network, and then use that trained network to provide needed inputs for the subsequent buildup network. In this way, later networks see realistic inputs during training rather than unrealistically clean data from a training set oracle. The training process is otherwise unmodified; each network is trained separately following the same training procedure as in the separated experiments. The only change is that some of the training inputs are provided by fixed neural networks earlier in the full pipeline.

For a combined prediction across two scales lr and hr, we predict S_{hr} from only q_{hr} following the procedure below:

$$\begin{aligned} \tilde{S}_{\text{lr}} &= D \circ f_\theta^{\text{downscale}}(q_{\text{hr}}) \\ S_{\text{hr}} &\approx f_\theta^{\text{buildup}}\big(q_{\text{hr}}, D^+(\tilde{S}_{\text{lr}})\big) + D^+(\tilde{S}_{\text{lr}}). \end{aligned} \tag{4}$$

The quantities $\tilde{S}$ are approximate neural network outputs used in intermediate predictions. We can extend this to perform a prediction using all three of our scale levels. For concreteness we

discuss this using the relevant dimension numbers, but these could of course be generalized for other settings. The three level cascaded procedure predicts S_{128} from only q_{128} by first performing a downscale prediction to a resolution of 64×64 followed by two buildup steps:

$$\begin{aligned}
\tilde{S}_{64} &= D^2_{\to 64} \circ f_\theta^{\text{downscale}}(q_{128}) \\
\tilde{S}_{96} &= f_\theta^{\text{buildup 1}}\big(D_{\to 96}(q_{128}),\, D^+_{\to 96}(\tilde{S}_{64})\big) + D^+_{\to 96}(\tilde{S}_{64}) \\
S_{128} &\approx f_\theta^{\text{buildup 2}}\big(q_{128},\, D^+_{\to 128}(\tilde{S}_{96})\big) + D^+_{\to 128}(\tilde{S}_{96}).
\end{aligned} \tag{5}$$

In the above, the different scaling operators D, D^+ are distinguished by subscripts with arrows toward the scale of the result. Downscaling across two levels is denoted D^2. Equation 4 and Equation 5 compose the prediction tasks described in Equation 2 and Equation 3.

4 Results

For each of the prediction tasks described in Section 3, 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 mean squared error evaluated over each sample and averaged. The other two metrics are derived from previous work evaluating neural network parameterizations [13] (where they were called $\mathcal{L}_{\text{rmse}}$ and $\mathcal{L}_{\text{S}}$). These were originally designed to measure performance for stochastic subgrid forcings. Here we use the two metrics from that work which do not collapse to trivial results for deterministic models. These are defined as:

$$\text{Rel } \ell_2 \triangleq \frac{\|S - \tilde{S}\|_2}{\|S\|} \qquad \text{and} \qquad \text{Rel Spec } \ell_2 \triangleq \frac{\|\text{sp}(S) - \text{sp}(\tilde{S})\|_2}{\|\text{sp}(S)\|_2} \tag{6}$$

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 [1]. 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.

Table 1 shows the results for the downscale 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 "downscale" 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 Table 1 and Table 2) for the MSE metric are illustrated in Figure 3.

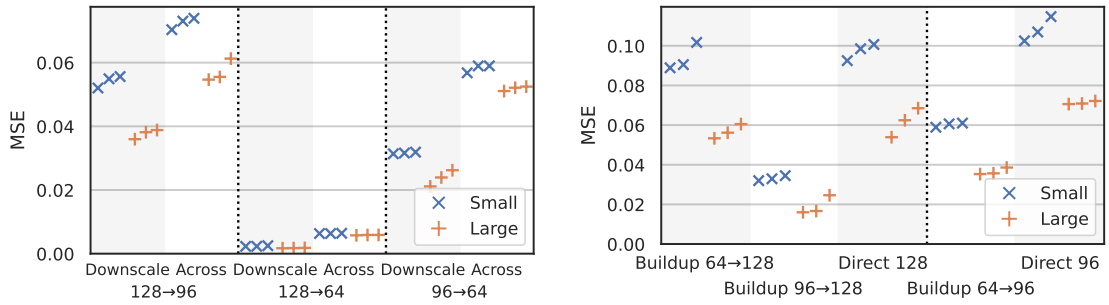
Results for the combined experiments, reported in Table 3, illustrate the early potential of this approach. In general, splitting the subgrid forcing prediction across scales slightly improved

NN Size	Metric	128 → 96		128 → 64		96 → 64	
		Downscale	Across	Downscale	Across	Downscale	Across
Small	MSE	0.054	0.072	0.002	0.006	0.032	0.058
	Rel ℓ_2	0.317	0.364	0.394	0.629	0.348	0.469
	Rel Spec ℓ_2	0.133	0.145	0.154	0.471	0.154	0.254
Large	MSE	0.038	0.057	0.002	0.006	0.024	0.052
	Rel ℓ_2	0.259	0.316	0.335	0.595	0.297	0.436
	Rel Spec ℓ_2	0.092	0.129	0.125	0.443	0.103	0.212

Table 1: Evaluation results for downscale 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 3a.

NN Size	Metric	Buildup 64 → 128	Buildup 96 → 128	Direct 128	Buildup 64 → 96	Direct 96
Small	MSE	0.094	0.033	0.097	0.060	0.108
	Rel ℓ_2	0.314	0.187	0.319	0.251	0.333
	Rel Spec ℓ_2	0.138	0.054	0.139	0.095	0.162
Large	MSE	0.057	0.019	0.062	0.037	0.071
	Rel ℓ_2	0.242	0.141	0.251	0.195	0.268
	Rel Spec ℓ_2	0.074	0.029	0.084	0.041	0.091

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 3b.



(a) Downscale vs. across MSE

(b) Buildup vs. direct MSE

Figure 3: Evaluation results from both of the separated experiments for the MSE metric. These are the same numbers which are reported as averages in Table 1 and Table 2. The plot here shows the three samples—one from each trained network—used to compute the means.

NN Size	Metric	Combined 64, 128	Combined 96, 128	Combined 64, 96, 128	Direct 128	Combined 64, 96	Direct 96
Small	MSE	0.098	**0.078**	**0.085**	0.097	**0.100**	0.108
	Rel ℓ_2	0.320	**0.284**	**0.297**	0.319	**0.320**	0.333
	Rel Spec ℓ_2	**0.139**	**0.091**	**0.104**	0.139	**0.139**	0.162
Large	MSE	**0.061**	**0.053**	**0.058**	0.062	**0.065**	0.071
	Rel ℓ_2	**0.249**	**0.232**	**0.243**	0.251	**0.256**	0.268
	Rel Spec ℓ_2	**0.082**	**0.061**	**0.063**	0.084	**0.081**	0.091

Table 3: Evaluation results for the combined experiments, composing "downscale" and "buildup" networks, eliminating the input forcing oracle. Bolded values show where the combined network outperformed the associated baseline (repeated from Table 2). The results contributing to the MSE averages in this table are illustrated in Figure 4.

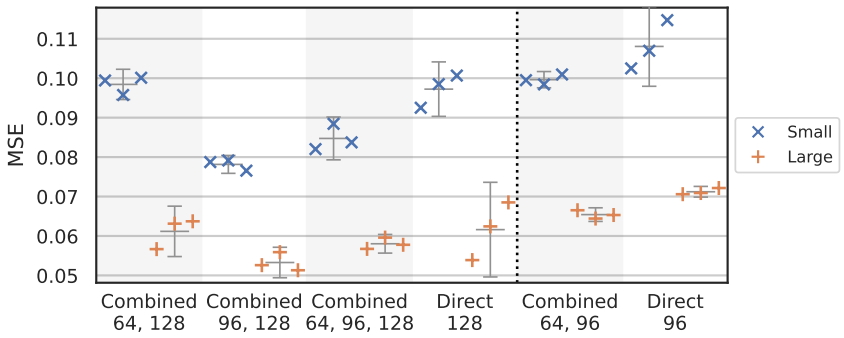


Figure 4: Evaluation results from the combined experiments for the MSE metric. These are the same values reported as averages in Table 3. The bars behind each cluster of points show the mean (horizontal bar in the center) along with a 2σ range on either side, computed from the empirical standard deviations.

performance, particularly in cases with a smaller scale step down. The improvement is particularly pronounced for the small size networks, suggesting there may be potential to improve architectural efficiency using a multiscale prediction process.

We find that adding additional buildup steps slightly hurt performance relative to a single lower scale level. We expect this may be due to accumulated errors propagated between the networks. Adding a small amount of noise during the sequential training process may help improve robustness. The points contributing to the MSE averages in Table 3 are illustrated in Figure 4.

5 Conclusion

Our proof of concept experiments in this work illustrate the potential advantages from decomposing the subgrid forcing problem into one across scales. We see this as an approach which may have regularization advantages, explicitly representing multiscale aspects of this prediction problem, and supporting learning in scarce data regimes and better handling underlying uncertainty in this task.

The results reported here represent an initial step in an ongoing project. In our continuing work we will further investigate combining these prediction tasks to increase robustness to errors in intermediate predictions. We anticipate that adding noise and other perturbations during

training may improve the performance of the combined experiments. We will further work to quantify the regularization benefits of this approach in limited-data regimes, and investigate other ways to structure the multiscale prediction task, in particular tailoring the neural network architecture to minimize computational cost and taking advantage of the multiscale prediction process to maintain prediction quality. We also plan to expand evaluation to include online tests, using the learned parameterizations across simulation time steps, and test generalization to other quasi-geostrophic simulation parameter settings, and on other climate-modeling tasks.

References

[1] Ryan Abernathey et al. *pyqg: v0.7.2*. Version v0.7.2. May 2022. DOI: 10.5281/zenodo.6563667. URL: https://doi.org/10.5281/zenodo.6563667.

[2] Igor Babuschkin et al. *The DeepMind JAX Ecosystem*. 2020. URL: http://github.com/deepmind.

[3] James Bradbury et al. *JAX: composable transformations of Python+NumPy programs*. Version 0.3.13. 2018. URL: https://github.com/google/jax.

[4] Baylor Fox-Kemper et al. "Principles and advances in subgrid modelling for eddy-rich simulations". In: *CLIVAR Exchanges (WGOMD Workshop on High Resolution Climage Modeling)*. Vol. 19. 2014, pp. 42–46.

[5] Baylor Fox-Kemper et al. "Challenges and Prospects in Ocean Circulation Models". In: *Frontiers in Marine Science* 6 (2019). ISSN: 2296-7745. DOI: 10.3389/fmars.2019.00065. URL: https://www.frontiersin.org/articles/10.3389/fmars.2019.00065.

[6] Hugo Frezat et al. "A Posteriori Learning for Quasi-Geostrophic Turbulence Parametrization". In: *Journal of Advances in Modeling Earth Systems* 14.11 (2022). e2022MS003124 2022MS003124, e2022MS003124. DOI: 10.1029/2022MS003124. eprint: https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1029/2022MS003124. URL: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2022MS003124.

[7] Arthur P. Guillaumin and Laure Zanna. "Stochastic-Deep Learning Parameterization of Ocean Momentum Forcing". In: *Journal of Advances in Modeling Earth Systems* 13.9 (2021). e2021MS002534 2021MS002534, e2021MS002534. DOI: 10.1029/2021MS002534. eprint: https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1029/2021MS002534. URL: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2021MS002534.

[8] Jonathan Ho et al. "Imagen Video: High Definition Video Generation with Diffusion Models". In: *arXiv Preprint* (2022). URL: https://arxiv.org/abs/2210.02303.

[9] Malte F. Jansen et al. "Toward an Energetically Consistent, Resolution Aware Parameterization of Ocean Mesoscale Eddies". In: *Journal of Advances in Modeling Earth Systems* 11.8 (2019), pp. 2844–2860. DOI: https://doi.org/10.1029/2019MS001750. URL: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2019MS001750.

[10] Patrick Kidger and Cristian Garcia. "Equinox: neural networks in JAX via callable PyTrees and filtered transformations". In: *Differentiable Programming workshop at Neural Information Processing Systems 2021* (2021).

[11] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv Preprint* (2014). URL: https://arxiv.org/abs/1412.6980.

[12] R. Maulik et al. "Subgrid modelling for two-dimensional turbulence using neural networks". In: *Journal of Fluid Mechanics* 858 (2019), 122–144. DOI: 10.1017/jfm.2018.770.

[13] Pavel Perezhogin, Laure Zanna, and Carlos Fernandez-Granda. "Generative data-driven approaches for stochastic subgrid parameterizations in an idealized ocean model". In: *arXiv Preprint* (2023). URL: https://arxiv.org/abs/2302.07984.

[14] Andrew Ross et al. “Benchmarking of Machine Learning Ocean Subgrid Parameterizations in an Idealized Model”. In: *Journal of Advances in Modeling Earth Systems* 15.1 (2023). e2022MS003258 2022MS003258, e2022MS003258. DOI: 10.1029/2022MS003258. eprint: https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1029/2022MS003258. URL: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2022MS003258.

[15] Uriel Singer et al. “Make-A-Video: Text-to-Video Generation without Text-Video Data”. In: *arXiv Preprint* (2022). URL: https://arxiv.org/abs/2209.14792.

[16] J. Smagorinsky. “General Circulation Experiments with the Primitive Equations: I. The Basic Experiment”. In: *Monthly Weather Review* 91.3 (1963), pp. 99 –164. DOI: 10.1175/1520-0493(1963)091<0099:GCEWTP>2.3.CO;2.

[17] Laure Zanna and Thomas Bolton. “Data-Driven Equation Discovery of Ocean Mesoscale Closures”. In: *Geophysical Research Letters* 47.17 (2020). e2020GL088376 10.1029/2020GL088376, e2020GL088376. DOI: 10.1029/2020GL088376. eprint: https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1029/2020GL088376. URL: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2020GL088376.

A Quasi-Geostrophic Model

For our experiments we target the two-layer quasi-geostrophic model implemented in PyQG which is a simplified approximation of fluid dynamics [1]. 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:

$$\frac{\partial \hat{q}_1}{\partial t} = -\widehat{J(\psi_1, q_1)} - ik\beta_1\hat{\psi}_1 + \widehat{\text{ssd}} \tag{7}$$

$$\frac{\partial \hat{q}_2}{\partial t} = -\widehat{J(\psi_2, q_2)} - ik\beta_2\hat{\psi}_2 + r_{\text{ek}}\kappa^2\hat{\psi}_2 + \widehat{\text{ssd}} \tag{8}$$

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:

$$\begin{bmatrix} -(\kappa^2 + F_1) & F_1 \\ F_2 & -(\kappa^2 + F_2) \end{bmatrix} \begin{bmatrix} \hat{\psi}_1 \\ \hat{\psi}_2 \end{bmatrix} = \begin{bmatrix} \hat{q}_1 \\ \hat{q}_2 \end{bmatrix}. \tag{9}$$

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 [14].

We use the “eddy” configuration from [14] which sets the following values for model constants:

$$r_{\text{ek}} = 5.787 \times 10^{-7} \qquad F_1 = \frac{1}{r_d^2(1+\delta)}$$

$$\delta = \frac{H_1}{H_2} = 0.25 \qquad F_2 = \delta F_1$$

$$\beta = 1.5 \times 10^{-11} \qquad W = 10^6$$

$$r_d = 15\,000 \qquad L = 10^6$$

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 [14] 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 $\Delta t = 3\,600$ generating 86 400 steps from which we keep every eighth leaving 10 800 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 C) 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:

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

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 Network Architecture and Training

We use the feedforward CNN architecture from [7] without batch norm as our standard "small" architecture. The "large" size 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. Each input and output quantity has two layers, each of which is handled as a separate channel. These parameters are adjusted for each task to accommodate the inputs and make the required predictions. We implement our networks with Equinox [10].

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)

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

We train each network with the Adam optimizer [11] as implemented in Optax [2]. 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 10 850 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 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 upscaling D^+) is applied as a part of each prediction task to adjust scales around the neural networks as needed. These are the operators referenced in Figure 1 and Figure 2.

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 $\rho \triangleq \text{hr}/\text{lr}$.

C.1 Data Filtering

The data filtering operator C is "Operator 1" as described in [14]. 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:

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

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

C.2 Rescaling Operator

For scale manipulations as part of our learned model we make use of a scaled spectral truncation operator. We define a downscaling operator D as well as an upscaling operator D^+:

$$D \triangleq \rho^{-2}\mathcal{D} \qquad \text{and} \qquad D^+ \triangleq \rho^2 \mathcal{D}^T. \tag{10}$$

Note that D^+ is a right side inverse $DD^+ = I$, and that D^+ is the pseudoinverse $D^+ = D(DD^T)^{-1}$ 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.