

Banner appropriate to article type will appear here in typeset article

Addressing *A Posteriori* Performance Degradation in Neural Network Subgrid Stress Models

Andy Wu^{1†} and Sanjiva K. Lele^{1, 2}

¹Department of Aeronautics and Astronautics, Stanford University, Stanford, California, USA

²Department of Mechanical Engineering, Stanford University, Stanford, California, USA

(Received xx; revised xx; accepted xx)

Neural network subgrid stress models often have *a priori* performance that is far better than the *a posteriori* performance, leading to neural network models that look very promising *a priori* completely failing in *a posteriori* Large Eddy Simulations (LES). This performance gap can be decreased by combining two different methods, training data augmentation and reducing input complexity to the neural network. Augmenting the training data with two different filters before training the neural networks has no performance degradation *a priori* as compared to a neural network trained with one filter. *A posteriori*, neural networks trained with two different filters are far more robust across two different LES codes with different numerical schemes. In addition, by ablating away the higher order terms input into the neural network, the *a priori* versus *a posteriori* performance changes become less apparent. When combined, neural networks that use both training data augmentation and a less complex set of inputs have *a posteriori* performance far more reflective of their *a priori* evaluation.

Key words: Authors should not enter keywords on the manuscript

1. Introduction

Large Eddy Simulations (LES) provides an economical paradigm for high fidelity flow prediction by resolving the larger, energy containing scales and modelling the spatial scales of turbulence that are smaller than the grid scale through a subgrid scale (SGS) closure (Sagaut 2006). Spatially filtering the Navier-Stokes equations results in an unclosed term called the subgrid stress tensor, τ_{ij} , which represents the effect of the subgrid spatial scales of turbulence on the resolved flow (Pope 2000).

Recently, with the availability of Direct Numerical Simulation (DNS) data online and advances in deep learning theory, neural networks have been explored as subgrid stress models (Sarghini *et al.* 2003; Beck *et al.* 2019; Park & Choi 2021; Cho *et al.* 2024; Xie *et al.* 2020a,b; Stoffer *et al.* 2021; Kang *et al.* 2023; Maejimam & Kawai 2024; Cheng *et al.* 2022; Wu & Lele 2025b). A persistent challenge is the discrepancy in neural network *a priori* and *a posteriori* performance (Beck *et al.* 2019; Beck & Kurz 2021; Stoffer *et al.* 2021; Park &

† Email address for correspondence: awu1018@stanford.edu

Choi 2021), which can be interpreted as a distribution shift between the training data and the LES simulation (Hu *et al.* 2025). Due to this distribution shift, ad-hoc methods are sometimes used, such as adding an additional Smagorinsky term to help the neural network dissipate energy so that simulation remains stable (Beck *et al.* 2019). Two mechanisms may contribute to the distribution shift. First, the input fields provided to the neural network during training, which are generated assuming an explicit filter (oftentimes a box filter), can differ from the LES resolved fields, since the LES filter is implicit and is a product of both the grid and the numerical method (Sagaut 2006). Second, the output SGS stress statistics also differ across filtering operations. For example, the omission or use of two-thirds dealiasing in spectral methods modifies the effective filter transfer function, producing different SGS statistics. This motivates the hypothesis that neural networks trained on one filter may struggle to generalize to LES solvers *a posteriori*. To address these two mechanisms of distribution shift, this work introduces a multi-filter data augmentation strategy that exposes the neural network to various plausible filtered inputs and SGS stress distributions.

Reinforcement learning approaches (Kim *et al.* 2022; Bae & Koumoutsakos 2022; Kurz *et al.* 2023) aim to bypass these issues by learning the SGS model (or wall model) directly in a LES solver. However, models learned in this manner are limited in their transferability to other solvers, as they learn the combined effect of solver-specific numerical effects and turbulence physics. In contrast, the present supervised learning approach aims to not learn solver peculiarities, and incorporates known filtering operations into the training data. The filters used do not mimic the specific transfer function of any specific LES solver and instead are inspired by common LES numerical behavior.

Furthermore, another potential source of distribution shift can come from the use of increasingly complex neural network inputs, such as higher order powers of the velocity gradient tensor. For example, the second invariant $Q = 0.5(P^2 - \text{trace}(V_{ij}^2))$ where $P = \text{trace}(V_{ij})$ and V_{ij} is the velocity gradient tensor, and the third invariant $R = \det(V_{ij})$ are more sensitive to aliasing and differences in numerical methods (where \det denotes the determinant). Therefore, a systematic ablation of neural network input complexity is performed to analyze the effect of inputs to neural network performance *a posteriori*.

The contributions of this work are twofold. First, multi-filter data augmentation is shown to significantly improve the robustness of learned SGS models across small architecture perturbations and across LES solvers with different numerical methods. Second, an ablation of input feature complexity reveals that more complex neural network inputs suffer more from *a posteriori* performance degradation. Together, these results enable the training of neural networks that are far more robust *a posteriori*, and are more indicative of their *a priori* performance.

2. Numerical Details

2.1. LES

LES simulations solve the filtered Navier-Stokes equations. The filtering operation is defined below:

$$\bar{\chi}(\mathbf{x}, t) = \int G(\mathbf{r})\chi(\mathbf{x} - \mathbf{r}, t)d\mathbf{r}, \int G(\mathbf{r})d\mathbf{r} = 1 \quad (2.1)$$

where χ is the flow variable to be filtered, $G(\mathbf{r})$ is the filtering kernel, and $\bar{\chi}$ is the filtered flow variable. In all sections, the overbar operator is used to denote filtered quantities. For robust subgrid stress modeling, the filtering operation must not be oscillatory in physical space, as this would create additional small scales purely through the act of filtering (Sagaut

2006). This means that the spectral cutoff filter, which is in the shape of a sinc function in physical space (while being sharp in spectral space) is not appropriate as a filter for subgrid stress modeling. With the definition of the filter above, the LES governing equations are written below: (in Einstein's summation notation)

$$\begin{aligned} \frac{\partial \bar{u}_i}{\partial x_i} &= 0 \\ \frac{\partial \bar{u}_i}{\partial t} + \bar{u}_j \frac{\partial \bar{u}_i}{\partial x_j} &= -\frac{1}{\rho} \frac{\partial \bar{p}}{\partial x_i} + \nu \frac{\partial^2 \bar{u}_i}{\partial x_j \partial x_j} - \frac{\partial \tau_{ij}}{\partial x_j} \\ \tau_{ij} &= \overline{\bar{u}_i \bar{u}_j} - \bar{u}_i \bar{u}_j \end{aligned} \quad (2.2)$$

where \bar{p} is the filtered pressure, ρ is the density, \bar{u} is the filtered velocity, and ν is the kinematic viscosity. As seen, an unclosed term, τ_{ij} , is obtained by filtering the momentum equation. In LES, only the filtered values of the dependent variables are available, thus, τ_{ij} can't be computed and has to be modelled. In the present work, the neural network will learn to approximate τ_{ij} directly.

2.2. A Posteriori Flow Solvers

Since this work involves drawing conclusions about *a posteriori* performance, two in-house LES flow solvers, PadeOps and PadeLibs (with different numerical methods) are used (Ghate & Lele 2017; Song *et al.* 2024). When running forced homogenous isotropic turbulence (HIT), PadeOps is a Fourier-Spectral code with RK4-5 time stepping (Bogacki & Shampine 1996). Therefore, two-thirds dealiasing is applied due to the non-linear component in the convective acceleration term. Meanwhile, PadeLibs is a 6th order compact code with RK3 time stepping (with no 2/3 dealiasing). These two codes have vastly different numerical methods, so the *a posteriori* conclusions obtained hold for different numerical schemes.

3. Filtering and Neural Network Details

Neural network SGS models depend on explicitly filtered DNS data, making the choice of filters critical. Mismatches between the explicit filters chosen and the implicit filters encountered in *a posteriori* LES can produce both input-side and output-side distribution shifts. To reduce such mismatches, the present work adopts multi-filter training, selecting explicit filters to diversify the training distribution so that the filters span representative LES filtering behaviors without approximating a specific LES solver's transfer function. The first explicit filter chosen is the box filter, as it is a common filter used in previous studies. The box filter is defined as follows:

$$\bar{\chi}(\mathbf{x}, t) = \int_V \frac{\chi(\mathbf{x} - \mathbf{r}, t)}{|V|} d\mathbf{r} \quad (3.1)$$

where V is the volume (or region) of the box filter, $|V|$ is its measure. The box filter is not oscillatory in physical space, but is oscillatory in spectral space.

3.0.1. DSCF: A Custom Localized Low-Pass Filter

To approximate a “cleaner” grid-cutoff at the Nyquist wavenumber, a discrete spectral cutoff filter (DSCF) is introduced. The DSCF approximates the spectral cutoff filter and avoids the oscillatory kernel of an ideal spectral cutoff filter in physical space. To construct this localized low-pass filter kernel, a discrete rectangular transfer function on a uniform frequency grid was constructed, and then the inverse discrete Fourier transform was taken. The discrete

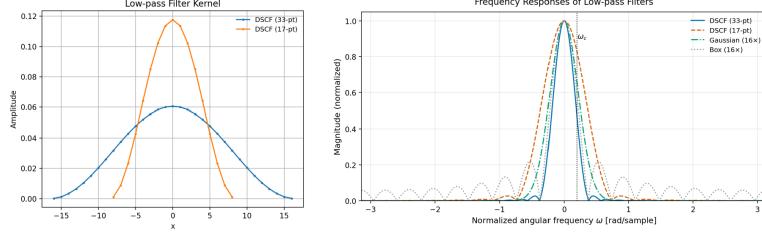


Figure 1: Characterizations of the DSCF in both physical and spectral space. $\omega_c = 2\pi f_c$ corresponds to the filter cutoff point, here designated at a $16\Delta_{DNS}$ filter width, while $\omega_c = 2\pi f_n$, $f_n \in [-0.5, 0.5]$ denotes the normalized angular frequency. Two different DSCF versions are shown, one with a 17 and the other with a 33 point support.

frequency response (from the rectangular transfer function) was specified as

$$G_k = \begin{cases} 1, & |u_k| \leq f_c, \\ 0, & \text{otherwise,} \end{cases} \quad (3.2)$$

where $u_k \in [-0.5, 0.5]$ denotes the normalized discrete frequency and f_c is the prescribed cutoff. This would be the grid scale cutoff of LES, for example, if one wanted to filter the DNS data at a $\Delta_{LES} = 16\Delta_{DNS}$ filter width, then $f_c = \frac{1}{16} = 0.03125$. The corresponding spatial filter kernel is obtained using the inverse discrete Fourier transform, followed by a normalization to ensure $\sum_n g_n = 1$.

The filtering operation applied to a discrete field χ_n is then given by the discrete convolution

$$\bar{\chi}_n = \sum_{m=-N/2}^{N/2} g_m \chi_{n-m}. \quad (3.3)$$

Although the construction in (3.2) is formally based on an ideal spectral cutoff filter, the use of a discrete frequency grid and discrete inverse transform produce a spatial kernel that is compactly supported and strictly positive in physical space. Both the physical space and spectral space characterizations of the DSCF are shown in figure 1. As seen in figure 1, the frequency response is not perfectly rectangular, instead exhibiting a smoother roll-off with mild high wavenumber oscillations when compared to a box filter. Meanwhile, in physical space, the filter is strictly positive, non-oscillatory, and has finite support. A drawback to DSCF is that achieving good frequency behavior requires roughly twice as many points in the physical space kernel. For example, if filtering at a $16\Delta_{DNS}$ filter width, a box filter uses 17 points, while the DSCF requires the use of 33 points. Using only 17 points causes the DSCF to “roll off” more slowly, removing less high frequency content. Meanwhile, the 33 point DSCF matches the initial decay of the box filter but significantly suppresses the oscillations (peak oscillation magnitude of 0.028 vs 0.22), and attenuates frequencies beyond the grid cutoff more than the Gaussian filter. As such, if trying to filter at a LES grid cutoff frequency of $x\Delta_{DNS}$, the corresponding DSCF support is always set to $2x + 1$ (to maintain symmetry of the kernel). Training datasets are formed using (i) only the box filter and (ii) both the box filter and the DSCF. Neural networks trained with these datasets are evaluated in PadeLibs, whose implicit filtering properties differ from both the box filter and the DSCF.

3.0.2. Two-Thirds Dealiasing as an Additional Filter

PadeOps uses a two-thirds dealiasing procedure, a filtering operation that modifies the effective transfer function. To study how multi-filter training interacts with this procedure, a filter is derived that is consistent with two-thirds dealiasing. The total subgrid stress can

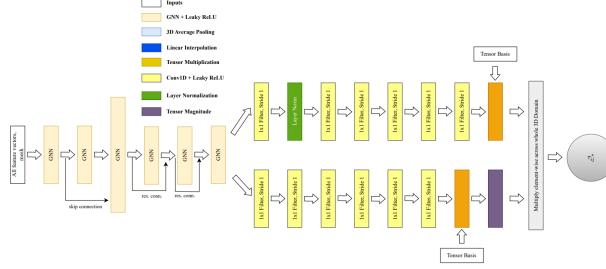


Figure 2: Overall graph neural network architecture. Note that “res. conn.” denotes residual connections.

be calculated by applying the two-thirds dealiasing operation on top of the already filtered Navier-Stokes equations, defined as the tilde operation:

$$\tau_{ij}^f = \widetilde{u_i u_j} - \tilde{u}_i \tilde{u}_j \quad (3.4)$$

As seen in equation 3.4, the subgrid stress tensor, after accounting for this two-thirds dealiasing operation, is denoted as τ_{ij}^f . The two-thirds de-aliasing operation is computed as an additional spectral cutoff filter on top of the box filtered data. This filtering operation will be called box and two-thirds (BTF), and is used alongside the box filter to form the augmented dataset for PadeOps. Note that the BTF is not intended to exactly replicate the transfer function of PadeOps, and it is unlikely that the transfer function of a fourier-spectral code with 2/3 dealiasing is identical to BTF. Rather, BTF provides the neural network with exposure to the type of behavior that two-thirds dealiasing qualitatively introduces, to reduce the training data distribution shift compared to *a posteriori* PadeOps.

3.1. Neural Network details

The neural network used is a hybrid tensor basis neural network (TBNN) from Ling *et al.* (2016) and graph neural network (GNN). In a classic TBNN, an Artificial Neural Network (ANN) maps inputs to scalar coefficients of the tensor basis expansion. Here, the ANN front-end architecture is substituted with a GNN, which allows for learned input stencils instead of prescribed input stencils (Abekawa *et al.* 2023). To recap, the tensor basis expansion for LES, where the neural network predicts 8 scalar coefficients $c_1 - c_8$, can be written as (Stallcup *et al.* 2022; Wu & Lele 2025b):

$$\tau_{ij} = c_1 \mathbf{I} + c_2 \bar{\mathbf{S}} + c_3 \bar{\mathbf{S}}^2 + c_4 \bar{\Omega}^2 + c_5 (\bar{\mathbf{S}} \bar{\Omega} - \bar{\Omega} \bar{\mathbf{S}}) + c_6 \bar{\Omega} \bar{\mathbf{S}} \bar{\mathbf{R}} + c_7 (\bar{\mathbf{S}}^2 \bar{\Omega} - \bar{\Omega} \bar{\mathbf{S}}^2) + c_8 (\bar{\Omega} \bar{\mathbf{S}} \bar{\Omega}^2 - \bar{\Omega}^2 \bar{\mathbf{S}} \bar{\Omega}) \quad (3.5)$$

where $\bar{\mathbf{S}}$ and $\bar{\Omega}$ denote the filtered strain rate or rotation rate tensor. The invariants, which are costly to compute, can be substituted with the velocity gradient tensor with no change in neural network accuracy (Wu & Lele 2025a). This is taken one step further, where one can use the invariants of the velocity gradient tensor (P, Q, R), as well as the magnitudes of the strain rate and rotation rate tensor as inputs instead ($|S|, |\Omega|$). The neural network architecture is shown in figure 2, which has many similarities to Wu & Lele (2025a). As seen, there are two different inputs to the neural network. The first group of inputs (such as P, Q, R) pass through layers with learnable weights and nonlinear activation functions. In contrast, the tensor basis inputs are passed directly to the model without undergoing any nonlinearities or learnable transformations that alter their overall structure. Instead, the tensor basis serve as an inductive bias for the neural network so that any output of the neural network obeys the structure of τ_{ij} . This distinction is important since the neural network is more sensitive to inputs that go through nonlinear transformations (Novak *et al.* 2018). Simplifying these

Table 1: Neural Network Configurations

Number of Filters	Inputs	Neural Network Structure	Flow Solver Integrated	Normalization	Neural Network Name
One filter (box)	$P, Q, R, S , \Omega $	Original	PadeOps, Padelibs	Global	NN-Box-Complex-Original-G
Two filter (box and BTF)	$P, Q, R, S , \Omega $	Original	PadeOps	Global	NN-BoxBTF-Complex-Original-G
Two filter (box and DSCF)	$P, Q, R, S , \Omega $	Original	Padelibs	Global	NN-BoxDSCF-Complex-Original-G
One filter (box)	$P, Q, R, S , \Omega $	Additional LN	PadeOps, Padelibs	Global	NN-Box-Complex-ALN-G
Two filter (box and BTF)	$P, Q, R, S , \Omega $	Additional LN	PadeOps	Global	NN-BoxBTF-Complex-ALN-G
Two filter (box and DSCF)	$P, Q, R, S , \Omega $	Additional LN	Padelibs	Global	NN-BoxDSCF-Complex-ALN-G
One filter (box)	$P, S , \Omega $	Original	PadeOps, Padelibs	Global	NN-Box-Simple-Original-G
Two filter (box and BTF)	$P, S , \Omega $	Original	PadeOps	Global	NN-BoxBTF-Simple-Original-G
Two filter (box and DSCF)	$P, S , \Omega $	Original	Padelibs	Global	NN-BoxDSCF-Simple-Original-G
One filter (box)	$P, S , \Omega $	Additional LN	PadeOps, Padelibs	Global	NN-Box-Simple-ALN-G
Two filter (box and BTF)	$P, S , \Omega $	Additional LN	PadeOps	Global	NN-BoxBTF-Simple-ALN-G
Two filter (box and DSCF)	$P, S , \Omega $	Additional LN	Padelibs	Global	NN-BoxDSCF-Simple-ALN-G
One filter (box)	$P, S , \Omega $	Original	PadeOps, Padelibs	Local	NN-Box-Simple-Original-L
Two filter (box and BTF)	$P, S , \Omega $	Additional LN	PadeOps	Local	NN-BoxBTF-Simple-ALN-L
Two filter (box and DSCF)	$P, S , \Omega $	Additional LN	Padelibs	Local	NN-BoxDSCF-Simple-ALN-L

inputs (less higher powers of the velocity gradient tensor) would reduce their susceptibility to distribution shifts, leading to better *a posteriori* performance. To investigate this, neural networks are trained with two different sets of inputs, $P, Q, R, |S|, |\Omega|$ (“complex inputs”) and $P, |S|, |\Omega|$ (“simple inputs”). The simpler set is hypothesized to perform better *a posteriori*.

The neural network model inputs and outputs should be dimensionless. A global normalization (max-min normalization) is used for the input fields:

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}} * 2 - 1 \quad (3.6)$$

while a local normalization is adopted for the tensor basis and the subgrid stress tensor:

$$W_{ij}^{norm} = \frac{W_{ij}}{|V_{ij}|^x}, \tau_{ij}^* = \frac{\tau_{ij}}{\Delta_{LES}^2 |V_{ij}|^2} \quad (3.7)$$

where W_{ij} is the tensor to be normalized, V_{ij} denotes the velocity gradient tensor, $|V_{ij}|$ its magnitude, x the exponent to ensure dimensionless-ness, Δ_{LES} corresponds to the LES grid spacing, and repeated indices do not imply summation.

To ensure that the results are consistent across slight perturbations to the neural network architecture, a variant architecture is introduced with additional layer normalization operations between select GNN layers and the first 1x1 convolution layer. Input normalization is also varied, using local normalization. Training follows (Wu & Lele 2025b) with a composite loss function that minimizes the root mean squared error between the predicted and actual subgrid stress tensor (denoted as non-bolded RMSE), and the RMSE between the predicted and actual energy dissipation, $\epsilon = -\tau_{ij} \bar{S}_{ij}$ (denoted as non-bolded DRMSE):

$$L = \text{RMSE}(\tau_{ij,pred} - \tau_{ij})/\text{RMS}(\tau_{ij}) + \text{RMSE}(\epsilon_{pred} - \epsilon)/\text{RMS}(\epsilon) \quad (3.8)$$

where **RMSE** is the root-mean squared error operation and **RMS** is the root-mean square operation.

3.2. All Neural Network Configurations

A summary of all the neural networks trained are seen in table 1. Neural networks are trained with different training data (one filter versus augmented training data), different inputs (complex versus simple), and slightly different architectures (more or less layer normalization). All neural networks are trained on forced HIT at a filter width of $16\Delta_{DNS}$.

Table 2: *A priori* testing, each value is given as mean (standard deviation).

Neural Network	RMSE	DRMSE	Correlation
NN-Box-Complex-Original-G	0.751 (0.017)	0.664 (0.026)	0.660 (0.020)
NN-BoxBTF-Complex-Original-G	0.772 (0.017)	0.693 (0.028)	0.636 (0.020)
NN-BoxDSCF-Complex-Original-G	0.755 (0.017)	0.675 (0.028)	0.660 (0.020)
NN-Box-Complex-ALN-G	0.750 (0.017)	0.661 (0.026)	0.661 (0.020)
NN-BoxBTF-Complex-ALN-G	0.753 (0.017)	0.664 (0.026)	0.659 (0.021)
NN-BoxDSCF-Complex-ALN-G	0.758 (0.017)	0.675 (0.028)	0.660 (0.021)
NN-Box-Simple-Original-G	0.751 (0.017)	0.662 (0.024)	0.660 (0.020)
NN-BoxBTF-Simple-Original-G	0.758 (0.017)	0.668 (0.028)	0.657 (0.021)
NN-BoxDSCF-Simple-Original-G	0.757 (0.017)	0.672 (0.027)	0.661 (0.020)

4. A Priori Analysis

4.1. One Filter versus Two Filter Results

As seen in table 2, where the neural networks are evaluated on the test set box filtered data, there is no *a priori* performance degradation when training on two filters versus training on one filter since all RMSE, DRMSE, and correlation coefficient values are within 5 percent of each other. For example, neural networks trained with a max-min normalization and using the set of inputs $P, Q, R, |S|, |\Omega|$ have all have RMSE values in the 0.75-0.78 range, DRMSE values in the 0.66-0.70 range, and correlation coefficients in the 0.63-0.66 range. Also, using a complex or simple set of inputs has negligible difference on the neural network performance, suggesting that just using $|S|, |\Omega|$ is sufficient ($P = 0$ for incompressible flow, and is manually masked to be zero to prevent the neural network from overfitting to noise. It is kept so that the model can generalize to compressibility in future work).

5. A Posteriori Analysis

The neural networks are evaluated on two different flow solvers running forced HIT on a $64 \times 64 \times 64$ grid (corresponding to $16\Delta_{DNS}$) and a Taylor Reynolds number of 820. A quantitative metric is also given, sum spectral error ($SSE = \sum_k |(\log(E_{DNS_k}) - \log(E_{LES_k}))|$, $k_{PadeLibs} \in [0, 35]$, $k_{PadeOps} \in [0, 24]$), where $E(k)$ denotes the energy spectra, with differences in “k” since PadeOps has 2/3 dealiasing. While SSE is a good first-order metric, it must be interpreted in a physical context. SSE is computed using the logarithm of the energy spectra values, rendering it artificially sensitive to deviations in the higher-wavenumber, dissipative range. This can manifest in SSE oftentimes assigning a larger SSE value to dissipative spectra even if it is physically desirable, while not putting enough emphasis on high-wavenumber energy buildup. Therefore, visual inspection of the spectral decay or pile-up is still essential.

5.1. PadeOps

The neural networks are integrated into PadeOps, and after running to statistical stationarity, the spectra are shown below in figures 3a-3c. One can see that training with two filters significantly increases the robustness of the neural network *a posteriori*, as the spectra for neural networks trained with only one filter vary significantly in figures 3a-3b for small perturbations in the neural network architecture. Note that as seen in the *a priori* analysis

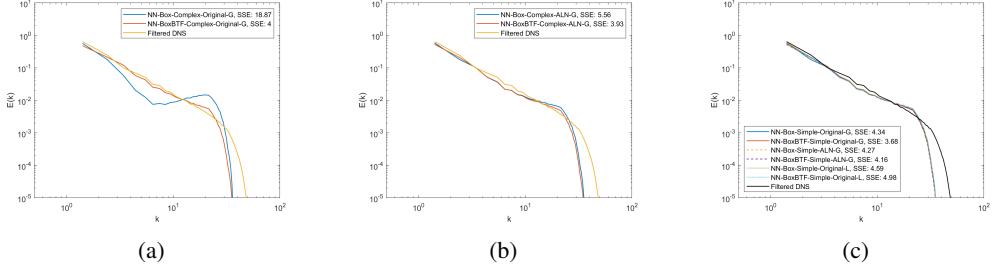


Figure 3: *A posteriori* PadeOps HIT Spectra, SSE is given in the legend (lower the better)

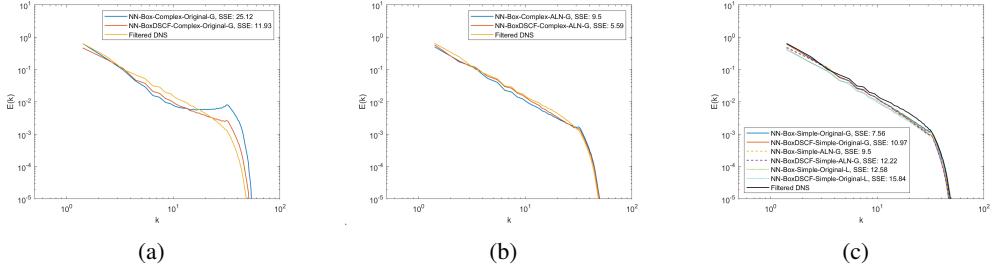


Figure 4: *A posteriori* Padelibs HIT Spectra, SSE is given in the legend (lower the better)

section, all neural networks have similar *a priori* performance, but the *a posteriori* SSE for neural networks trained with one filter can vary by a factor of 4. When considering fig. 3c, using a simpler set of inputs also significantly helps reduce the *a posteriori* variability across various neural network architectures and normalizations, which is also seen in the SSE. Note that in situations where the one filter approach performs well, the two filter approach also performs well, while if the one filter approach performs poorly, oftentimes the two filter approach performs better.

5.2. PadeLibs

The neural network *a posteriori* results for PadeLibs, after running to statistical stationarity, are shown in figures 4a-4c to investigate if the results hold for a different flow solver with different numerics. From figure 4a and figure 4b, one can see the same trend: training with two filters significantly increases the robustness of the neural network *a posteriori*, as training with one filter has very large variance for slight perturbations of neural network architectures, and has large SSE values even though *a priori* performance of the neural networks are similar. The same trend holds for figure 4c. Using the simpler set of inputs leads to more consistent *a posteriori* performance across different neural network normalizations and architectures. Models trained with one filter that perform well also show good results when trained on two filters, with *a posteriori* SSE values more similar to one another, consistent with *a priori* results. This suggests that using the simpler input set results in less distribution shift. The robustness seen likely stems from the simpler input set having less numerical error amplification. Numerical error and aliasing often manifest as high-wavenumber perturbations, which are often amplified when taking higher powers of the velocity gradient tensor. Thus, the “complex” input set suffers from a larger distribution shift. Even when the normalization is changed or the neural network architecture changes slightly, the simpler input set retains a reasonable spectra shape, with consistent results between one filter versus two filter cases. In general, when models trained with one filter

perform poorly, their two filter counterparts perform better, while if models trained on one filter perform well, their two filter counterparts also perform well.

5.3. Combined *A Posteriori* Analysis

Overall, combining data augmentation and simpler inputs yield robust *a posteriori* performance across solvers with different numerical methods. Training with two filters allows the neural network to see more diverse input and output distributions, leading to reduced distribution shift *a posteriori*. This allows the neural network, even though it is trained on generic filters not exactly mimicking the LES implicit transfer function, to perform more accurately to unseen implicit filters as it is not overfitting to the kernel artifacts associated with one specific filter (e.g. Gibbs Oscillations). Meanwhile, the “simple” training set outperforms the more complex training set because it is able to reduce the input distribution shift, as higher order powers of the velocity gradient tensor suffer more from numerical artifacts such as aliasing, especially at the grid cutoff resolution. By relying on the lower-order terms, the “simple” inputs allow the neural network to be less exposed to and not overfit to high-wavenumber numerical artifacts, increasing the *a posteriori* robustness of the neural network model.

6. Conclusion

Two methods to increase the *a posteriori* robustness of neural networks are proposed, data augmentation and decreasing the complexity of neural network inputs. Data augmentation involves training with multiple filters in the training data, and two filters have been proposed. BTF accounts for two-thirds dealiasing as an additional filter, whereas DSCF is non-oscillatory in physical space while approximating a sharp spectral cutoff filter in spectral space. *A priori*, neural networks suffer no performance degradation when trained on one filter as compared to two filters, suggesting that neural network models are able to distinguish between various filters. *A posteriori*, neural network models trained with two filters are significantly more robust than neural networks trained with one filter, and this trend is seen across two different LES flow solvers. Furthermore, neural network models with less complex inputs perform better, significantly reducing the distribution shift *a posteriori*. Training with two different filters does not reduce the performance of a neural network subgrid stress model either *a priori* or *a posteriori*. These trends hold across small neural network architecture perturbations and input normalizations. Training with data filtered with two different filters and also using less complex inputs to neural networks significantly increases the robustness of neural networks *a posteriori*.

Acknowledgements. This work used CPU and GPU resources via Bridges2 at the Pittsburgh Supercomputing Center through allocation PHY230025 from the Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support (ACCESS) program, which is supported by National Science Foundation grants #2138259, #2138286, #2138307, #2137603, and #2138296. DNS data is provided by the Johns Hopkins Turbulence Database.

Funding. Andy Wu is partially supported by NASA Cooperative Agreement number 80NSSC22M0108 and Northrop Grumman, as well as the NDSEG Fellowship.

Declaration of interests. Declaration of Interests. The authors report no conflict of interest.

REFERENCES

- ABEKAWA, A, MINAMOTO, Y, OSAWA, K, SHIMAMOTO, H & TANAHASHI, M 2023 Exploration of robust machine learning strategy for subgrid scale stress modeling. *Physics of Fluids* **35** (1).

- BAE, HJ & KOUMOUTSAKOS, P 2022 Scientific multi-agent reinforcement learning for wall-models of turbulent flows. *Nature Communications* **13** (1), 1443.
- BECK, A, FLAD, D & MUNZ, C-D 2019 Deep neural networks for data-driven LES closure models. *Journal of Computational Physics* **398** (108910).
- BECK, A & KURZ, M 2021 A Perspective on Machine Learning Methods in Turbulence Modelling. *Surveys for Applied Mathematics and Mechanics* **44** (1).
- BOGACKI, P & SHAMPINE, LF 1996 An efficient Runge-Kutta (4,5) pair. *Computers & Mathematics with Applications* **32** (6), 15–28.
- CHENG, Y, GIOMETTO, MG, KAUFFMANN, P, LIN, L, CAO, C, ZUPNICK, C, LI, H, LI, Q, HUANG, Y, ABERNATHEY, R & OTHERS 2022 Deep learning for subgrid-scale turbulence modeling in large-eddy simulations of the convective atmospheric boundary layer. *Journal of Advances in Modeling Earth Systems* **14** (5), e2021MS002847.
- CHO, C, PARK, J & CHOI, H 2024 A recursive neural-network-based subgrid-scale model for large eddy simulation: application to homogeneous isotropic turbulence. *Journal of Fluid Mechanics* **1000**.
- GHATE, AS & LELE, SK 2017 Subfilter-scale Enrichment of Planetary Boundary Layer Large Eddy Simulation Using Discrete Fourier-Gabor modes. *Journal of Fluid Mechanics* **819**.
- HU, Z, SUBRAMANIAM, A, KUANG, Z, LIN, J, YU, S, HANNAH, WM, BRENOWITZ, ND, ROMERO, J & PRITCHARD, MS 2025 Stable machine-learning parameterization of subgrid processes in a comprehensive atmospheric model learned from embedded convection-permitting simulations, arXiv: 2407.00124.
- KANG, M, JEON, Y & YOU, D 2023 Neural-network-based mixed subgrid-scale model for turbulent flow. *Journal of Fluid Mechanics* **962**.
- KIM, J, KIM, H, KIM, J & LEE, C 2022 Deep reinforcement learning for large-eddy simulation modeling in wall-bounded turbulence. *Physics of Fluids* **34** (10).
- KURZ, M, OFFENHÄUSER, P & BECK, A 2023 Deep reinforcement learning for turbulence modeling in large eddy simulations. *International journal of heat and fluid flow* **99**, 109094.
- LING, J, KURZAWSKI, A & TEMPLETON, J 2016 Reynolds averaged turbulence modelling using deep neural networks with embedded invariance. *Journal of Fluid Mechanics* **807**, 155–166.
- MAEJIMAM, S & KAWAI, S 2024 Coarse-grid large-eddy simulation by unsupervised-learning-based sub-grid scale modeling. In *AIAA SciTech 2024 Forum*, pp. AIAA 2024–1361.
- NOVAK, R, BAHRI, Y, ABOLAFIA, DA, PENNINGTON, J & SOHL-DICKSTEIN, J 2018 Sensitivity and generalization in neural networks: an empirical study, arXiv: 1802.08760.
- PARK, J & CHOI, H 2021 Toward neural-network-based large eddy simulation: application to turbulent channel flow. *Journal of Fluid Mechanics* **914**.
- POPE, SB 2000 *Turbulent Flows*, 1st edn. Cambridge University.
- SAGAUT, P 2006 *Large Eddy Simulation for Incompressible Flows: An Introduction*, 3rd edn. Springer Science & Business Media.
- SARGHINI, F, DE FELICE, G & SANTINI, S 2003 Neural networks based subgrid scale modeling in large eddy simulations. *Computers & Fluids* **32** (1), 97–108.
- SONG, H, GHATE, AS, MATSUNO, KV, WEST, JR, SUBRAMANIAM, A & LELE, SK 2024 A robust compact finite difference framework for simulations of compressible turbulent flows. *Journal of Computational Physics* **519**, 113419.
- STALLCUP, EW, KSHITIJI, A & DAHM, WJ 2022 Adaptive Scale-Similar Closure for Large Eddy Simulations. Part 1: Subgrid Stress Closure. In *AIAA SciTech*, pp. AIAA 2022–0595. San Diego, CA: AIAA.
- STOFFER, R, LEEUWEN, CM, PODAREANU, D, CODREANU, V, VEERMAN, MA, JANSSENS, M, HARTOGENSIS, OK & HEERWAARDEN, CC 2021 Development of a large-eddy simulation subgrid model based on artificial neural networks: a case study of turbulent channel flow. *Geoscientific Model Development* .
- WU, A & LELE, SK 2025a Subgrid stress modelling with multi-dimensional state space sequence models, arXiv: 2511.10910.
- WU, A & LELE, SK 2025b Two neural network unet architecture for subfilter stress modeling. *Physical Review Fluids* **10** (1), 014601.
- XIE, C, WANG, J & WEINAN, E 2020a Modeling subgrid-scale forces by spatial artificial neural networks in large eddy simulation of turbulence. *Physical Review of Fluids* **5** (5).
- XIE, C, YUAN, Z & WANG, J 2020b Artificial neural network-based nonlinear algebraic models for large eddy simulation of turbulence. *Physical of Fluids* **32** (11).