Progressive, Extrapolative Machine Learning for Turbulence Modeling

Jiaqi Li, Xiang Yang

Collaborators
George Huang @ Wright State University
Yuanwei Bin @ Penn State
Lihua Chen @ Zhejiang University
Acknowledgment
Philippe Spalart @ Boeing

Background: classical turbulence modeling

We still rely on cost-efficient tools like RANS.
Scale-resolving simulations are costly at high Reynolds numbers.
RANS requires turbulence modeling.

RANS equations $$\rho\frac{\partial \bar{u}_i}{\partial t} + \rho\bar{u}_j\frac{\partial \bar{u}_i}{\partial x_j} = \rho\bar{f}_i + \frac{\partial}{\partial x_j}\left[-\bar{p}\delta_{ij} + 2\mu\bar{S}_{ij} - \rho\overline{u_i' u_j'}\right]$$

Reynolds stress

Boussinesq hypothesis $$\tau_{ij}^R + \frac{2}{3}\rho k\delta_{ij} = 2\mu_T S_{ij}$$

Eddy viscosity

Conventional models are results of progressive modeling to account for different physics

Example: the Spalart-Allmaras model
The standard model, SA92: for free shear, wake, and flat plate boundary layer flows
Added corrections for system rotation [Dacles-Mariani et al. 95, 99]
Added corrections for wall curvature [Shur et al. 00]
Added corrections for compressibility [Spalart 00]
etc.

The more complex models respect the less complex models and protect the learned empiricism, e.g., the law of the wall.

Background: data driven turbulence modeling

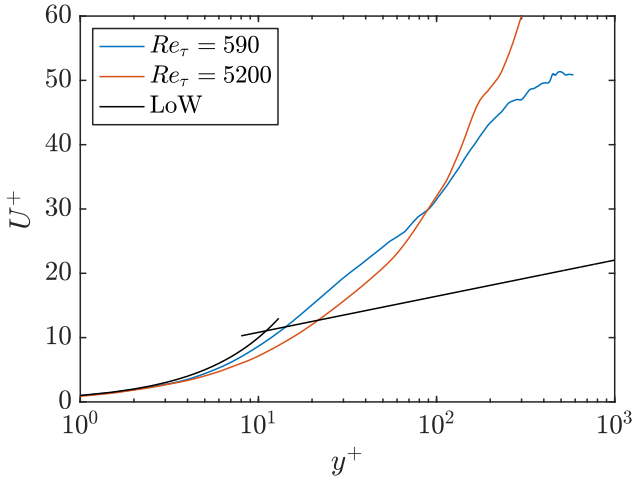
Data-enabled models are results of one-stop modeling

Example: Tensor basis neural network (TBNN) [Ling et al. 16]

Training data:
Duct, channel, jet in cross flow, flow around a square cylinder, converging diverging channel → Target:
Reynolds stress

Training is one-stop. No effort is made to protect the known empiricisms.

If we apply the TBNN to predict the mean velocity profiles in a channel:
Training: channel flow at friction Re = 590
Predicting: channel flow at friction Re = 5200
Does not protect the law of the wall



Data-enabled augmentations do not respect the known empiricism, e.g., the law of the wall. The models do not generalize well to unseen high Reynolds numbers.

[Rumsey, Coleman, Wang 2022]

Objectives

- **Apply progressive modeling, which is human thinking, in machine learning**
- Requirements
  - The model is open to compatible corrections to a baseline
  - The more complex model respects the less complex model
  - The model can extrapolate to high Reynolds numbers

- From Dr. Spalart, “The Mission and Requirements of a Turbulence Model”
  - Proposing a machine-learned model “for a specific flow” (e.g., periodic hill) is instructive, but is not “a product”
  - Ideally, a new model will come with corrections for curvature, compressibility, or anisotropy

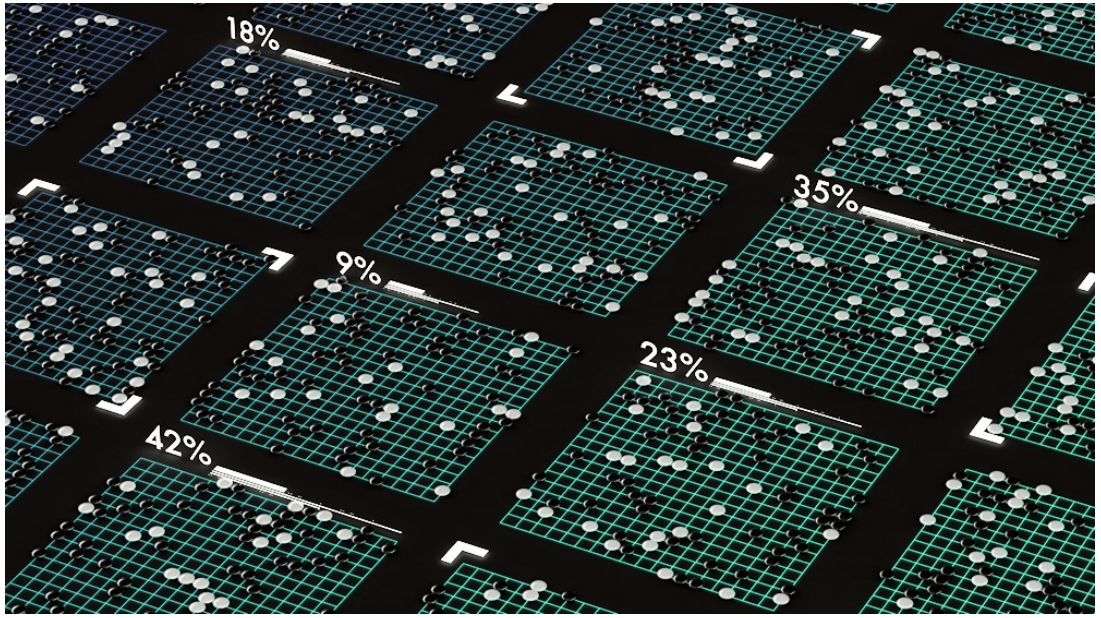
Motivation

Why do we want an AI that thinks like a human?

Let us consider a field where AI has seen success: the game Go.

Zen: consider all possible moves, machine-like thinking. ⟶ AI will never beat human intelligence.

AlphaGo: consider moves that will increase the probability of winning, human-like thinking.



AlphaGo thinks like a human. Through incrementing and iterating, it was able to transcend human intelligence.

Theoretical foundation

Universal approximation theorem:
A feedforward network with a single layer is sufficient to represent any function.

We will consider single-layer feedforward neural nets, without loss of generality.

Extrapolation theorem:
For a bias-unit-free non-trivial neural network that maps from R1 to R1, we have

net(∞) = a constant

if we employ the sigmoid transfer function for all neurons, and

net(x) ∼x

for sufficiently large x if we employ the rectified linear unit (ReLU) as the transfer function for all neurons.

The extrapolative behavior of an FNN is determined by the activation function.

Neutral neural network theorem:
0 input guarantees 0 output if we remove all bias units in a fully connected single-layer feedforward neural network.

It is possible to add neutral compatible corrections to an existing machine learning model.

Progressive machine learning

The basic idea

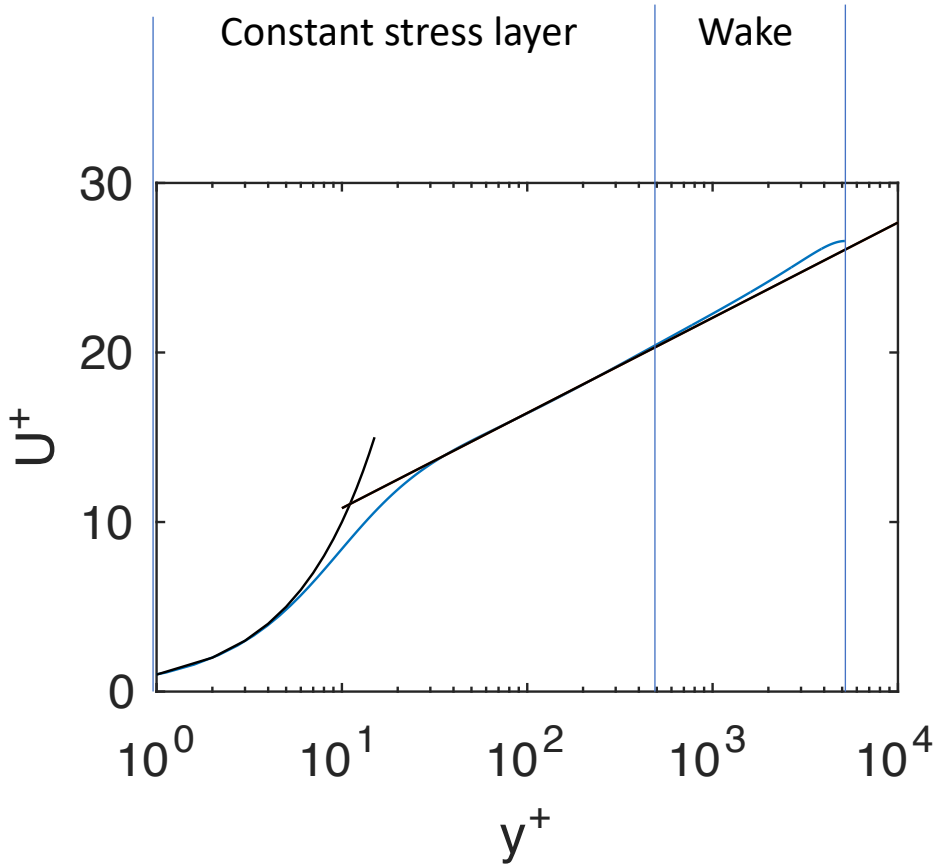
$$\mathrm{MODEL}^*(\mathbf{x}, \mathbf{X}) = \mathrm{MODEL}(\mathbf{x}) + \mathrm{CORRECTION}(||\mathbf{X}||_2 \cdot \mathbf{x}, \mathbf{X})$$

MODEL: an existing model
x: parameterization of some basic physics, e.g., S and d.
X: parameterization of new physics, e.g., Omg.
CORRECTION: a data-enabled correction.
MODEL*: an augmented model.

CORRECTION:
Neutral: CORRECTION = 0, when **X** = 0 (neutral neural network theorem).
Extrapolative: design output to preserve known empiricisms at unseen Reynolds numbers (extrapolation theorem).

Example

Turbulent boundary layer



Equations

$$-\langle u'v' \rangle + \nu \frac{dU}{dy} = \tau_w/\rho$$

$$-\langle u'v' \rangle = \boxed{\nu_t} \left| \frac{dU}{dy} \right|,$$

Target

Modeling the eddy viscosity

Constant stress layer (baseline, less complex model)

Constant stress layer + wake layer (more complex model)

Constant stress layer + wake layer + system rotation (even more complex model)

Progressive

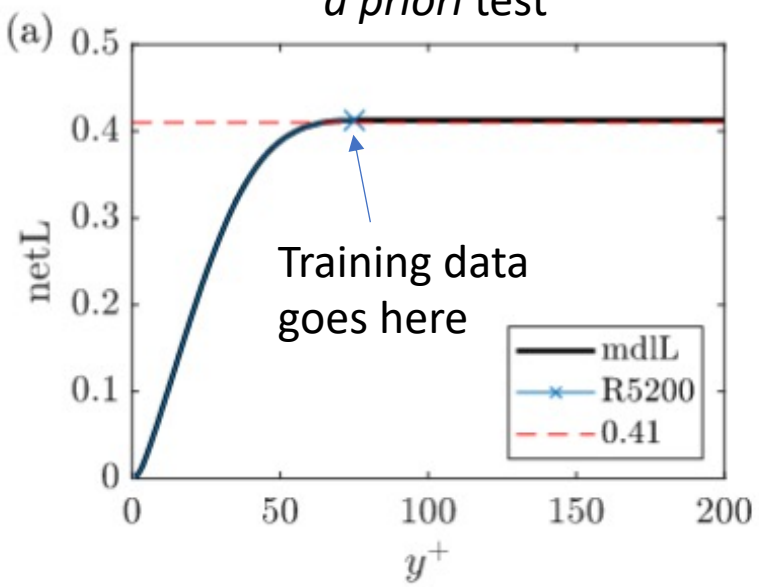
Examples of progressive machine learning

Train for eddy viscosity in the constant stress layer

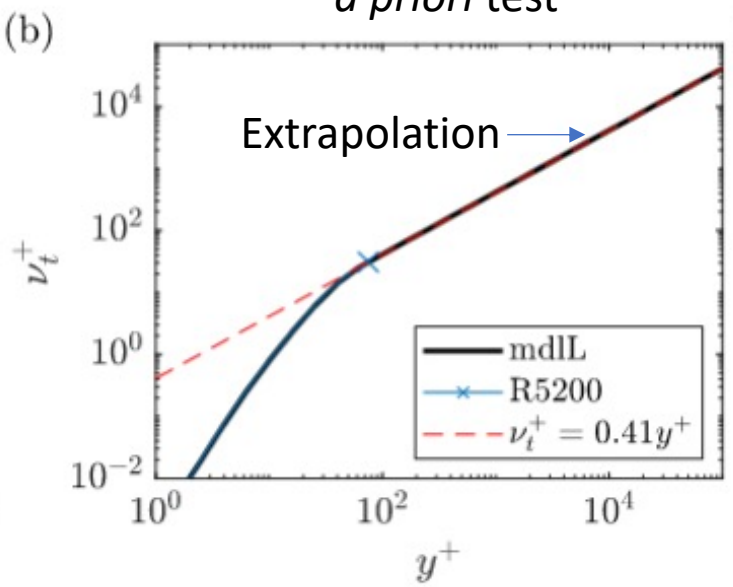
Network: sigmoid activated single-layer fully-connected feed-forward neural network, denote as netL
Output: v_t^+/y^+ --- extrapolation theorem guarantees the law of the wall
Training data: channel flow data up to $y/h = 0.015$.

The resulting model: mdlL

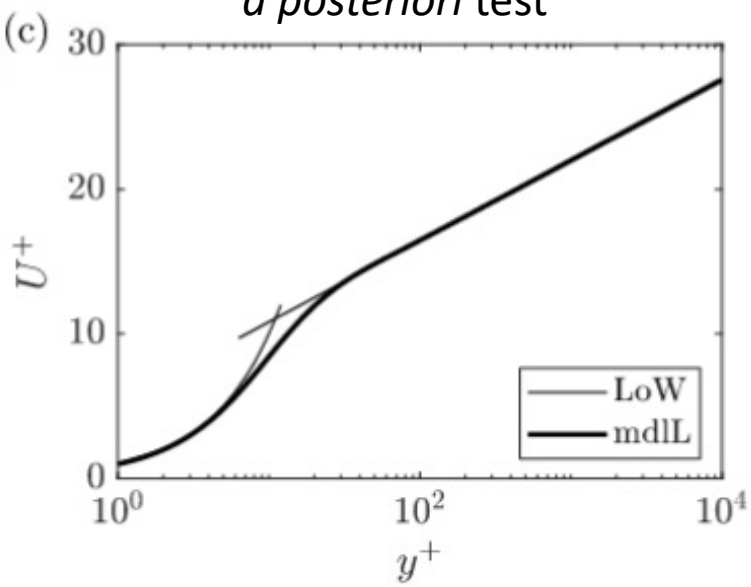
a priori test



a priori test



a posteriori test



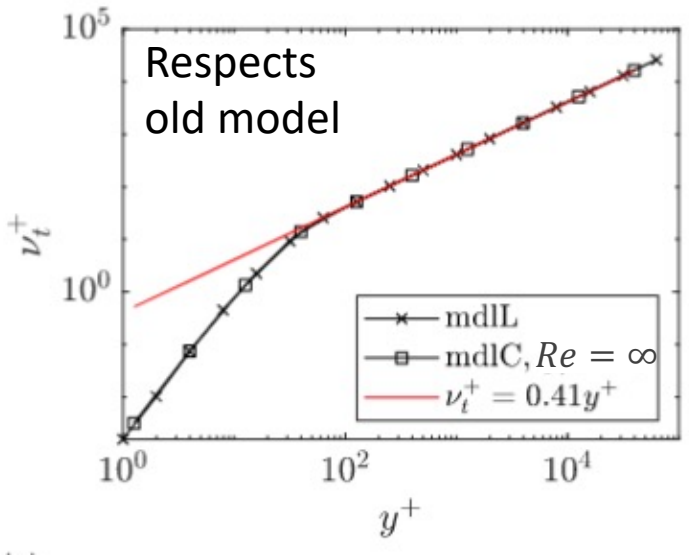
Learns the log layer physics and extrapolates to high Re.

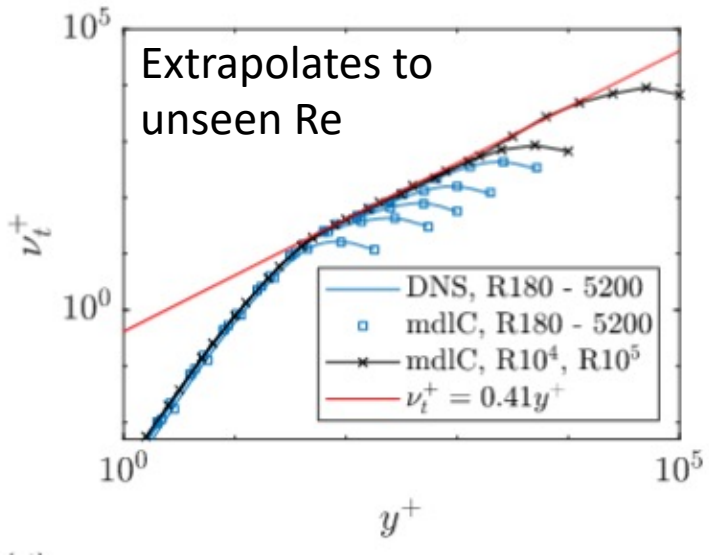
Examples of progressive machine learning

Train for eddy viscosity in the entire channel

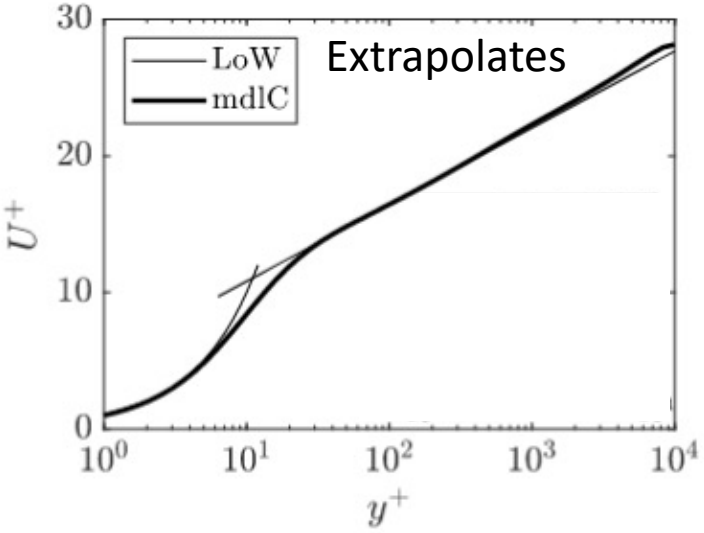
$$\nu_t^+/y^+ = \mathrm{netL}(y^+) + \mathrm{netC}(y^+(y/h), y/h).$$

a priori test





a posteriori test



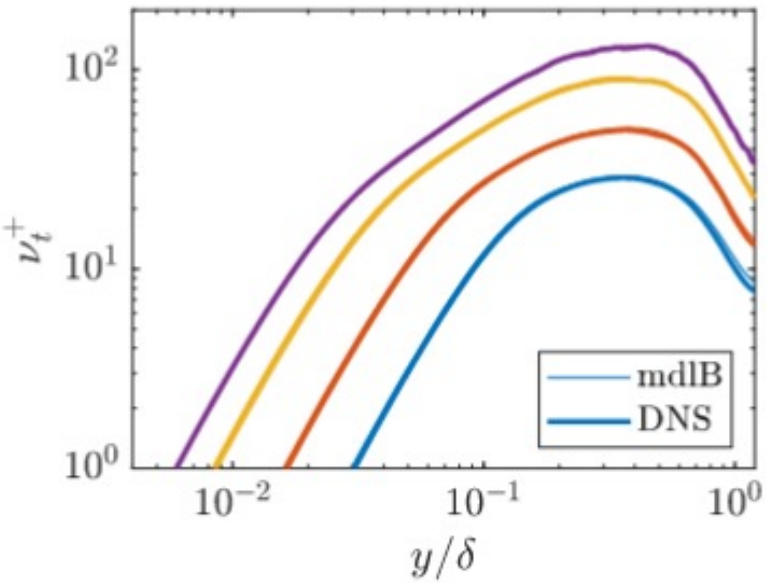
Learns the wake layer physics and protect the law of the wall in channel flow.

Examples of progressive machine learning

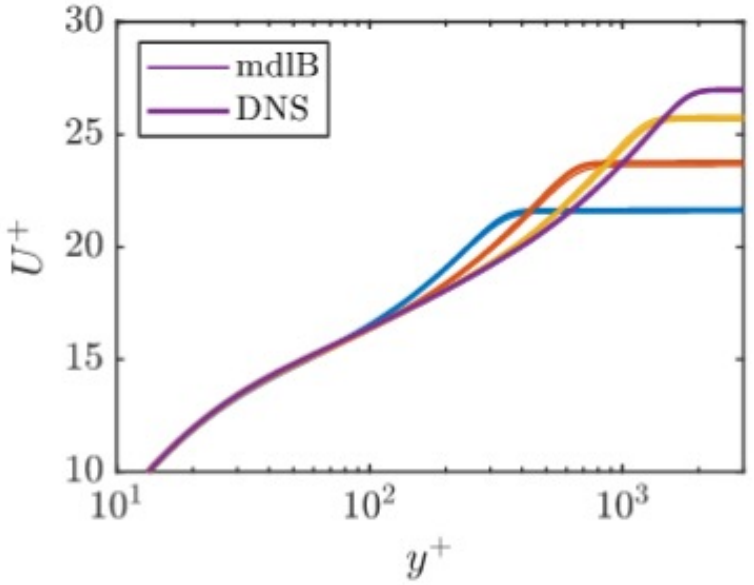
Train for eddy viscosity in boundary layer

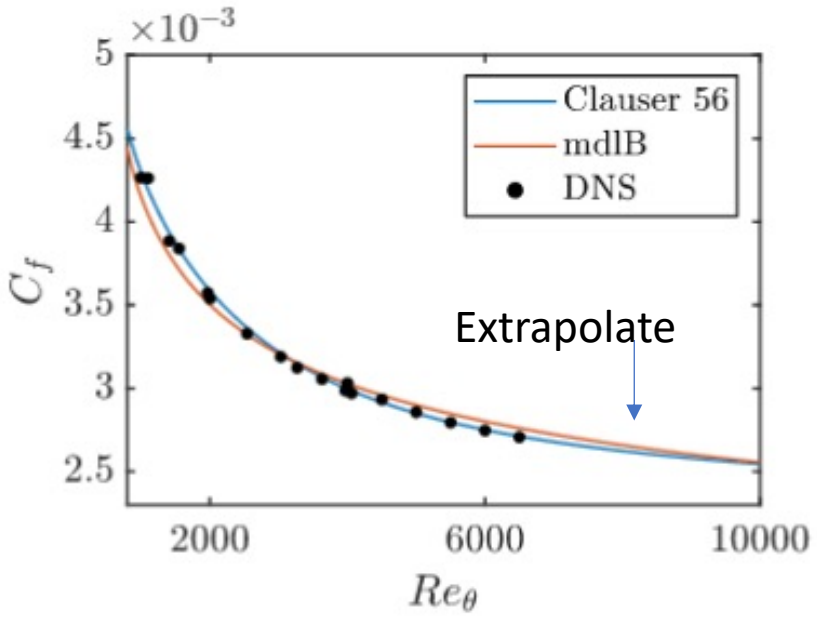
$$\nu_t = \mathrm{netL}(y^+)y^+ + \mathrm{netB}(y^+(y/\delta_\theta), y/\delta_\theta),$$

a priori test



a posteriori test





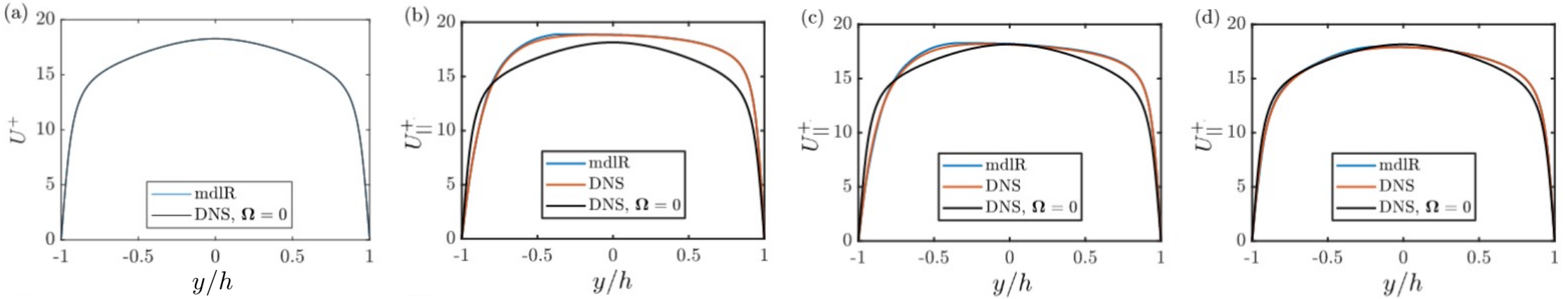
Learns the wake layer physics and protect the law of the wall in boundary layer.

Examples of progressive machine learning

Train for eddy viscosity in a rotating channel with small system rotation

$$v_t^+ = \mathrm{netL}(y^+)y^+ + \mathrm{netC}(y^+(y/h), y/h\,)y^+ + \mathrm{netR}(y^+(y/h)|\boldsymbol{\Omega}|^+, y/h|\boldsymbol{\Omega}|^+, \Omega_x^+, \Omega_y^+, \Omega_z^+)$$

Training data: 105 DNS from Huang & Yang 2021
Test data: 44 DNS from Huang & Yang 2021



Respects mdlC

unseen conditions

Learns rotation physics and protect previous learned physics.

Conclusions

- A new paradigm for machine learning, namely, progressive machine learning, is proposed and validated.
- In this framework, data-enabled turbulence models can be augmented like empirical models according to the neutral neural network theorem.
- We can control a network's behavior when extrapolating according to the extrapolation theorem.
- We apply progressive machine learning to progressively model the flow in the constant stress layer, the entire channel, and rotating channel. We show that the more complex models do not "forget" what it already "knows".

A Progressive Re-calibration of the Standard SA Model

Jiaqi Li, Xiang Yang

Collaborators
George Huang @ Wright State University
Yuanwei Bin @ Penn State
Lihua Chen @ Zhejiang University
Philippe Spalart @ Boeing

The standard SA model

The standard SA model is a result of progressive modeling

- Baseline: flow convection

$$\frac{D\nu_t}{Dt} = 0$$

- Free shear: turbulence production + diffusion

$$\frac{D\nu_t}{Dt} = c_{b1}S\nu_t + \frac{1}{\sigma}\frac{\partial}{\partial x_i}\left(\nu_t\frac{\partial \nu_t}{\partial x_i}\right) + \frac{c_{b2}}{\sigma}\left(\frac{\partial \nu_t}{\partial x_i}\right)^2$$

  - Physics + Data
  - ML: Bayesian optimization

- Log layer: destruction

$$\frac{D\nu_t}{Dt} = c_{b1}S\nu_t + \frac{1}{\sigma}\left[\frac{\partial}{\partial x_i}\left(\nu_t\frac{\partial \nu_t}{\partial x_i}\right) + c_{b2}\left(\frac{\partial \nu_t}{\partial x_i}\right)^2\right] - c_{w1}f_w\left(\frac{\nu_t}{d}\right)^2$$

  - Physics + Data
  - Data: high Re

- Viscosity

$$\frac{D\tilde{\nu}_t}{Dt} = c_{b1}\tilde{S}\tilde{\nu}_t + \frac{1}{\sigma}\left[\frac{\partial}{\partial x_i}\left((\nu + \tilde{\nu}_t)\frac{\partial \tilde{\nu}_t}{\partial x_i}\right) + c_{b2}\left(\frac{\partial \tilde{\nu}_t}{\partial x_i}\right)^2\right] - c_{w1}f_w\left(\frac{\tilde{\nu}_t}{d}\right)^2$$

  - Physics + Data
  - Data: high Re

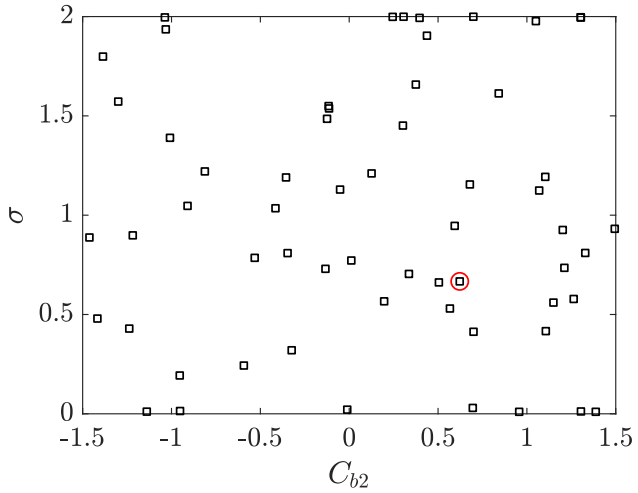
Progressive modeling

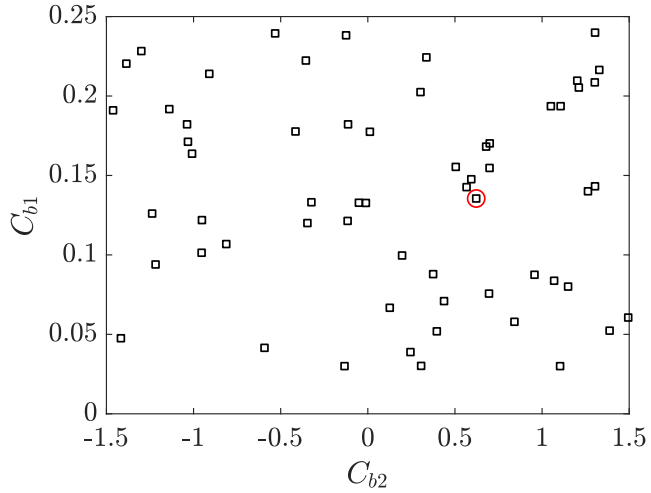
- The framework is progressive modeling.
- We follow these steps but rely on machine learning tools.
- We will limit the training data to mixing layer, wake, and channel (to test generalizability).

Recalibration of c_{b1}, c_{b2}, σ

Progressive modeling

Bayesian optimization: c_{b1}, c_{b2}, σ a 3D parameter space
Training data: mixing layer and jet (plane jet and axis-symmetric jet).
Projection of sampled points





Red: least-worse point

		σ	c_{b1}	c_{b2}
Standard SA	SA	*2/3*	*0.1355*	*0.622*
Data-enabled SA	SAM	*2/3*	*0.1355*	*0.622*

We expect the ML model to behave like the standard SA in free shear flows—
if the augmentation in the next steps do not destroy its behavior.

Recalibration of c_{b1}, c_{b2}, σ

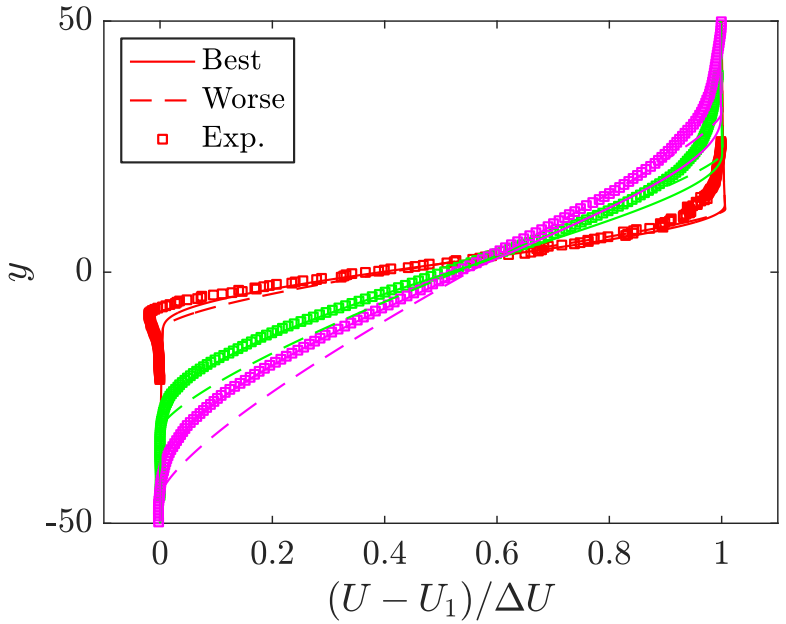
Cost function

- $\epsilon = \sum_{i=1}^{3} w_i \epsilon_i$
- $\epsilon_i = \left(\frac{\int (f_i - f_{i,t})^2 dx}{\int (f_{i,t} - \bar{f}_{i,t})^2 dx} \right)^{1/2}, w_i = 1$
- $f_{i,t}$: true value

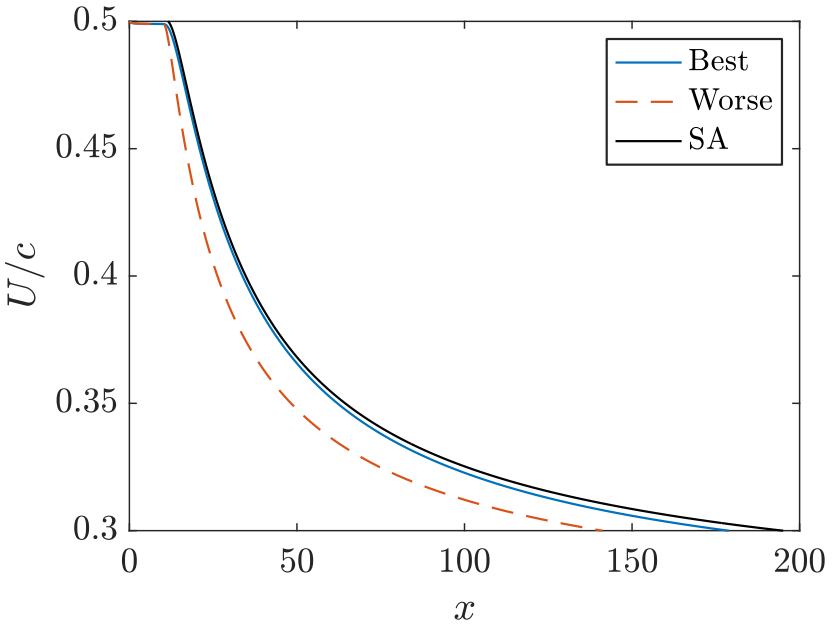
Example of a non-optimal combination

- $\sigma = 1.1548, c_{b1} = 0.1682, c_{b2} = 0.6791$

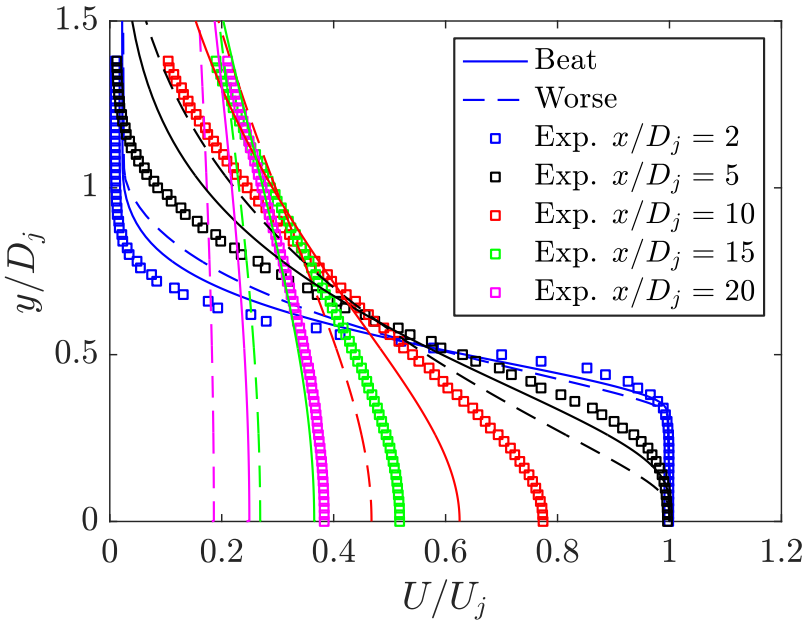
Mixing layer



Plane jet



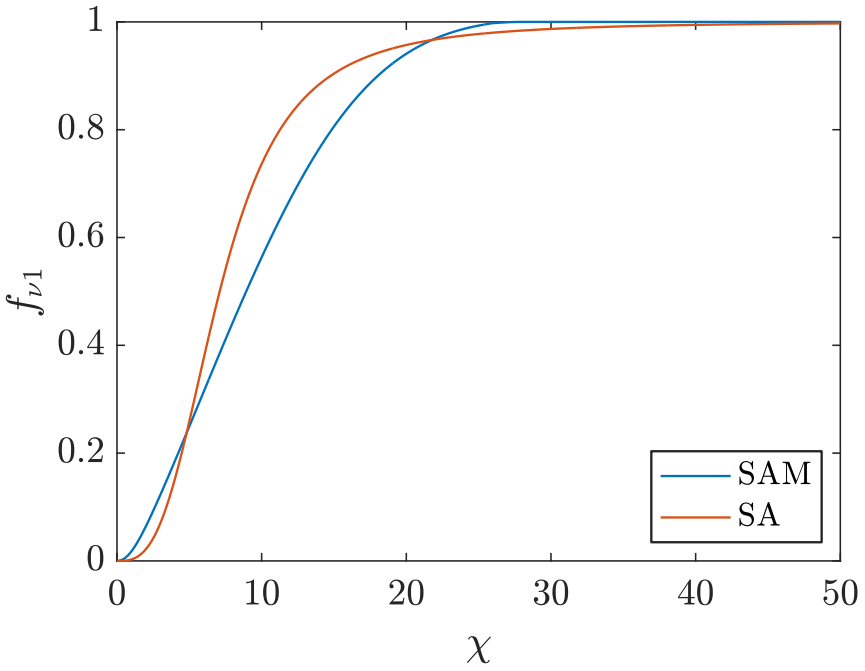
Axisymmetric jet

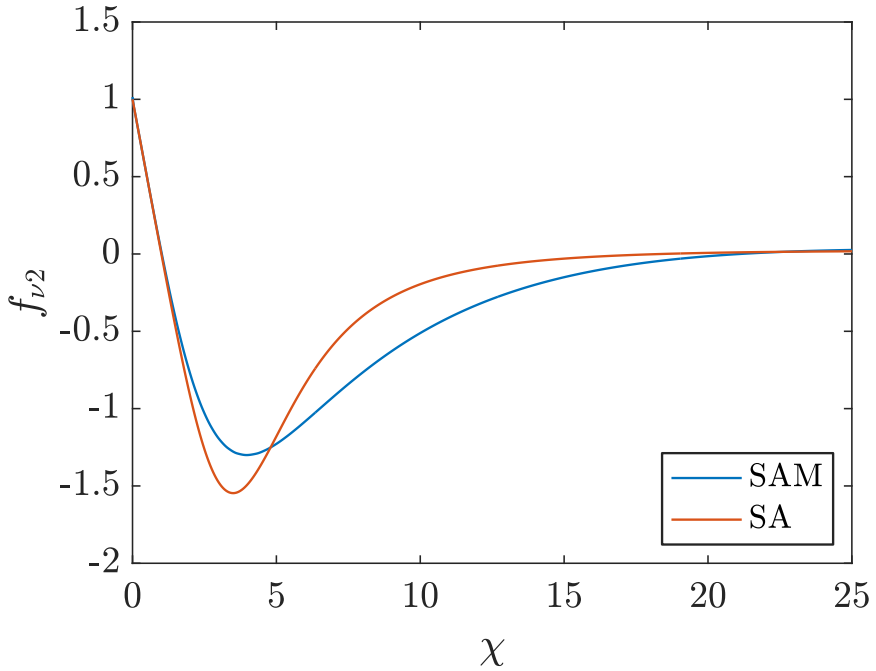


Recalibration of f_{v1}, f_{v2}

Progressive modeling

- Viscosity
- Training data: Channel flow at friction Re 5200.





$\chi = \tilde{\nu}_t/\nu$

Red: standard SA

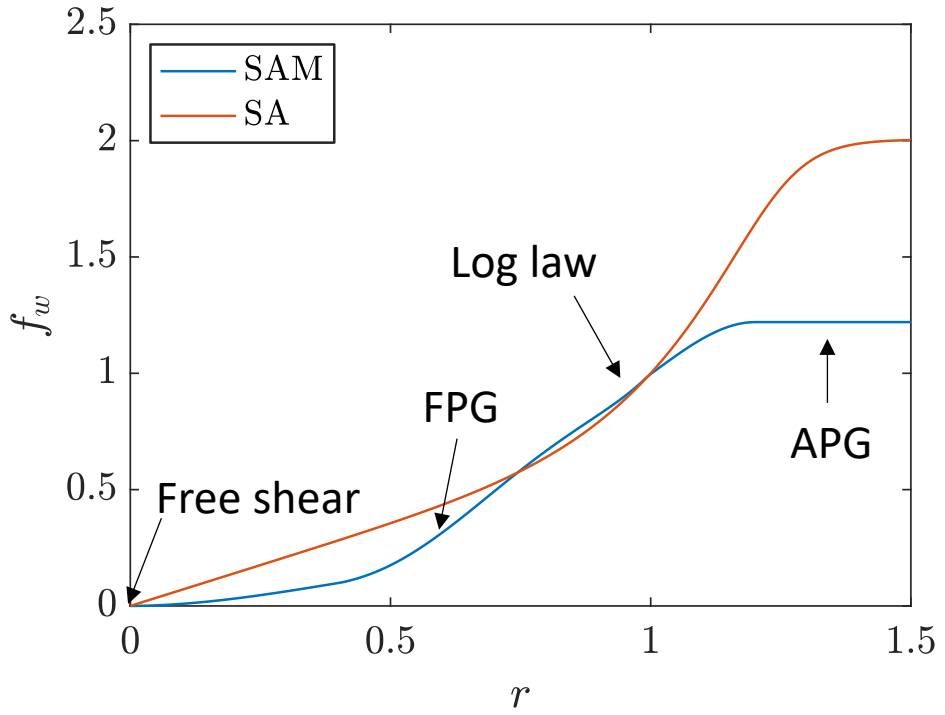
Blue: neural network prediction

We expect the ML model to handle the viscous layer more competently than the standard SA—if the augmentation in the next steps do not destroy its behavior.

Recalibration of f_w

Progressive modeling

- Near-wall behavior
- Training data: Channel (FPG) and Couette-Poiseuille (APG & FPG) DNS data.



$$r = \frac{\tilde{\nu}_t}{\tilde{S}\kappa^2 d^2}$$

Red: standard SA
Blue: Prediction of a neural net

We expect the ML model to do better than the standard SA in wall-bounded flows.

Recalibration for f_{v1}, f_{v2}, f_w

Blue: data can be obtained from DNS

Channel flow

- $f_{v1} \equiv \nu_t/\tilde{\nu}_t$, where $\nu_t = \frac{-\overline{uv}}{\partial U/\partial y}$, $\tilde{\nu}_t = u_\tau \kappa y$
- Transport equation of $\tilde{\nu}_t$ can be written as

$$\frac{D\tilde{\nu}_t}{Dt} = c_{b1} S \tilde{\nu}_t + \frac{\tilde{\nu}_t^2}{\kappa^2 d^2} f_{v2} + \frac{1}{\sigma}(\nu + \tilde{\nu}_t)\frac{\partial^2 \tilde{\nu}_t}{\partial x_i^2} + \frac{1 + c_{b2}}{\sigma}\left(\frac{\partial \tilde{\nu}_t}{\partial x_i}\right)^2 - \left(\frac{c_{b1}}{\kappa^2} + \frac{1 + c_{b2}}{\sigma}\right) f_w \left(\frac{\tilde{\nu}_t}{d}\right)^2$$

- For channel flow, near wall region, $\tilde{\nu}_t = u_\tau \kappa y, f_w = 1$

$$0 = c_{b1} S u_\tau \kappa y + u_\tau^2 f_{v2} + \frac{1 + c_{b2}}{\sigma}(u_\tau \kappa)^2 - \left(\frac{c_{b1}}{\kappa^2} + \frac{1 + c_{b2}}{\sigma}\right)(u_\tau \kappa)^2$$

$$f_{v2} = 1 - S\frac{\kappa y}{u_\tau}$$

- For channel flow, extend f_{v2} to log-law region
  - $\tilde{\nu}_t = u_\tau \kappa y$ still exists

$$f_w = 1$$

- For channel flow, extend f_{v2} to wake region
  - $\tilde{\nu}_t = \nu_t$

$$f_w = \frac{c_{b1} S \tilde{\nu}_t + \frac{\tilde{\nu}_t^2}{\kappa^2 y^2} f_{v2} + \frac{1}{\sigma}(\nu + \tilde{\nu}_t)\frac{\partial^2 \tilde{\nu}_t}{\partial x_i^2} + \frac{1 + c_{b2}}{\sigma}\left(\frac{\partial \tilde{\nu}_t}{\partial x_i}\right)^2}{\left(\frac{c_{b1}}{\kappa^2} + \frac{1 + c_{b2}}{\sigma}\right)\left(\frac{\tilde{\nu}_t}{d}\right)^2}$$

Neural network detail

Use neural network to train f_{v1}, f_{v2}, f_w with constrain

- $f_{v1}(0) = 0$
- $f_{v2}(0) = 1$
- $f_w\ (0) = 0$
- $f_w\ (1) = 1$

- To ensure $f(0) = 0$, we can train an intermediate odd function $g(x)$ and then $f(x)$ can be calculated as: $f(x) = g(x) - g(-x)$.
- One point in f will map to two points in g.

- To ensure $f(1) = 1$ and $f(0) = 0$, we formulate $f(x) = x(1-x)g(x) + x$ and train for $g(x)$.

Validation

Test cases from NASA TMR website with same grid

- 2D Airfoil-Near Wake
- 2D Mixing Layer
- 2D Coflowing jet
- Axisymmetric Subsonic Jet
- 2D Fully developed channel
- 2D Zero Pressure Gradient Flat Plate
- Axisymmetric Transonic Bump
- NACA 0012 airfoil with different AOA
- 2D WALL-Mounted Hump
- Back-Facing Step

Is the machine learning modeling better than the SA model?

Review

Baseline

In Search of Data-Driven Improvements to RANS Models Applied to Separated Flows

C. L. Rumsey*, G. N. Coleman*, and L. Wang*

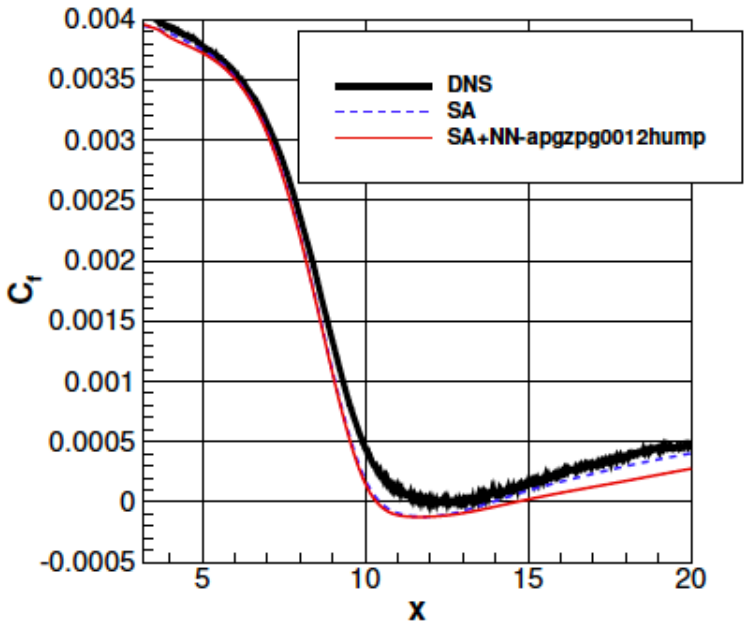
NASA Langley Research Center, Hampton, VA 23681

The goal of this work is to improve the capability of Reynolds-averaged Navier-Stokes turbulence models for separated flows using data-driven enhancements. The resulting model should be "universal" in the sense that it can be used by anyone and applied to as many flows as possible without concern for unusual or detrimental behavior. At worst, the data-driven corrections should not degrade the accuracy of the baseline model (in this case the Spalart-Allmaras one-equation model), while preserving the Galilean invariance and similar theoretical qualities of the original model. In the literature, most current data-driven improvements to turbulence models are only applicable to very similar types of cases as those used to train the model for a specific class of flows. In this work, the impact of using a wide array of cases in the machine-learning training is described. Unwanted behaviors from trained neural networks are examined, and possible mitigation strategies are proposed. However, to date, consistent and broadly applicable data-driven improvements for separated flows have not been achieved.

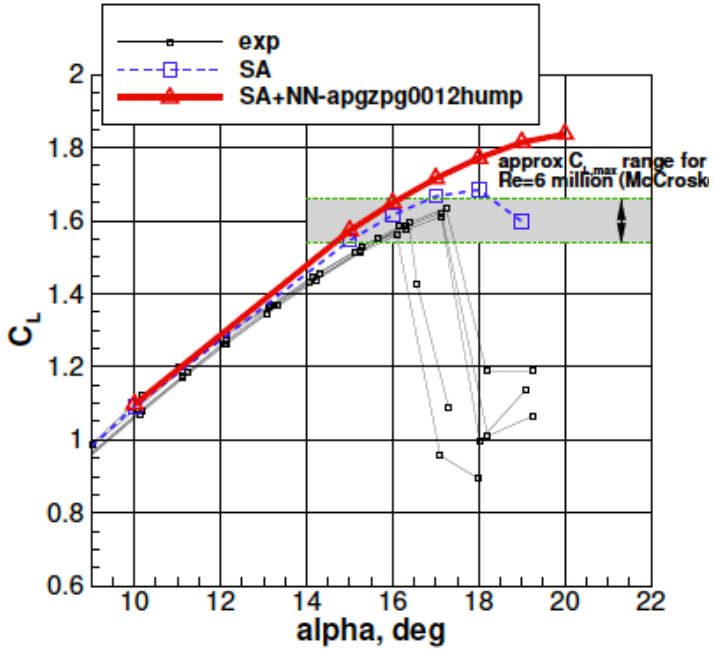
Review

Baseline

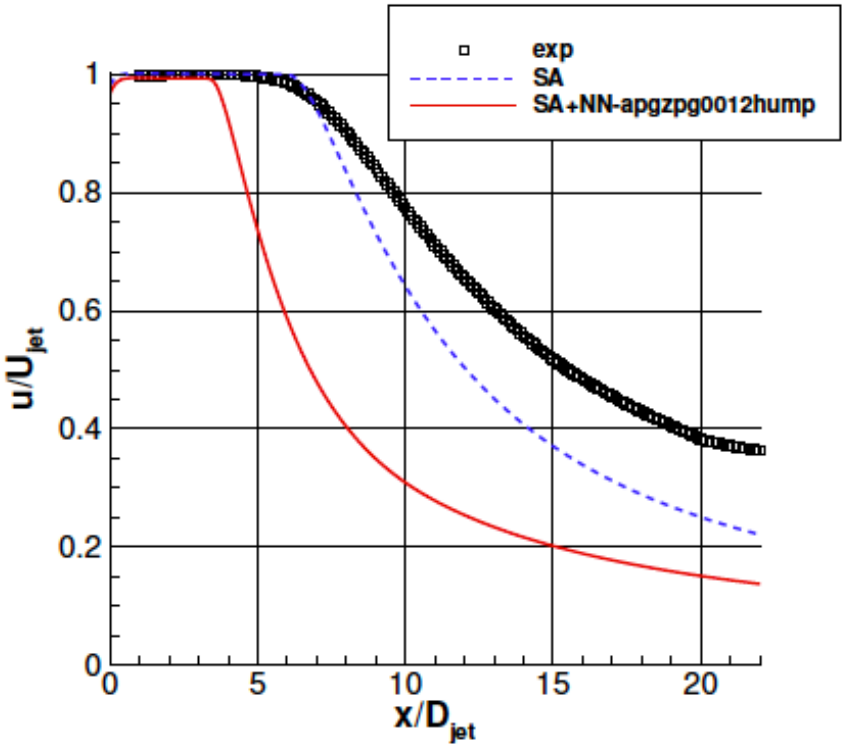
- Extrapolation and universality



(a) ZPG-APG plate at lower Re=80000.



(c) NACA 0012 near $C_{L,max}$.



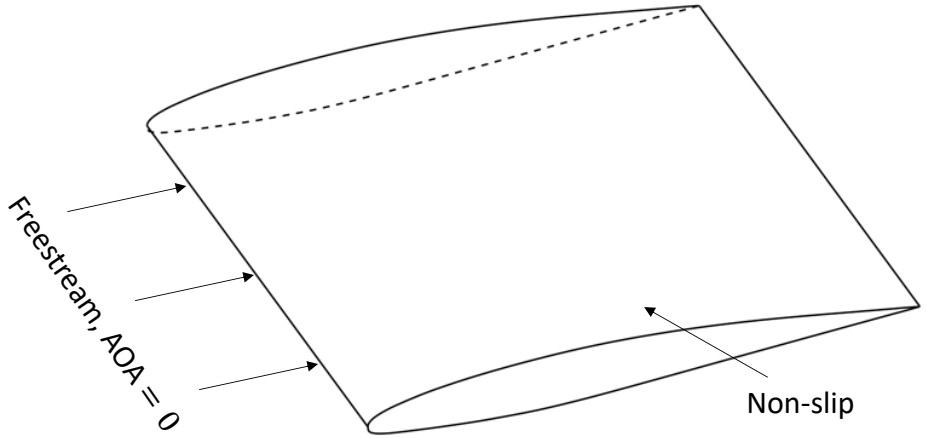
(e) Axisymmetric subsonic jet.

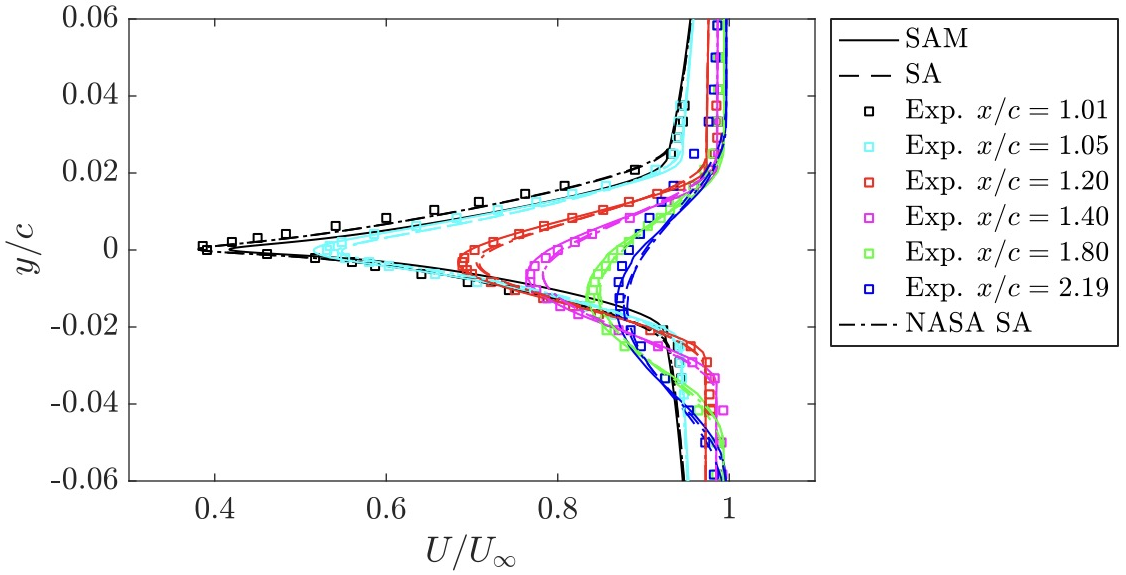
Worse than the standard SA model.

Testing

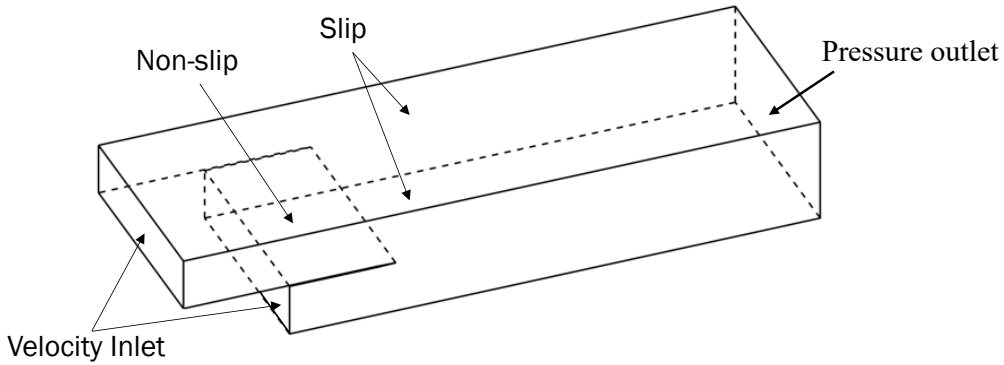
Not worse for free shear flows.

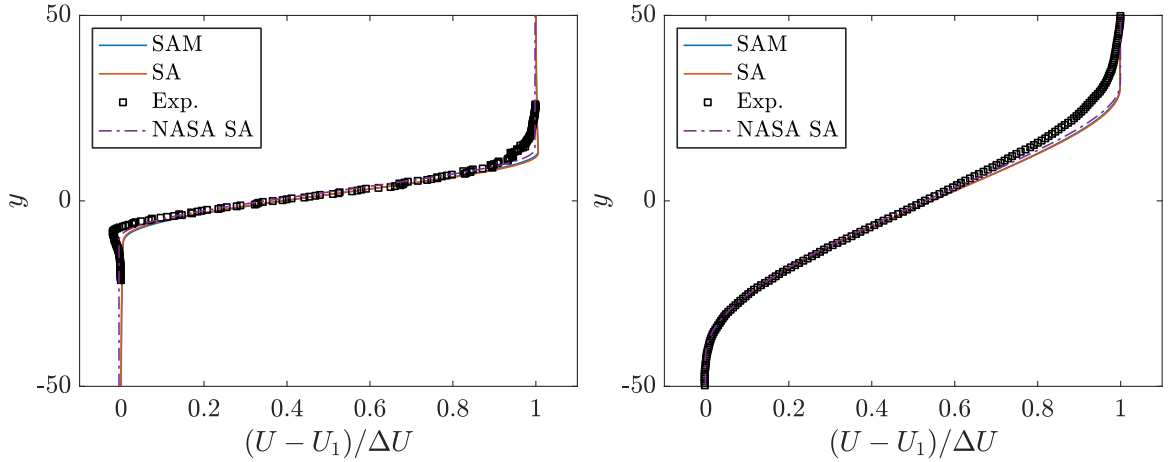
2DANW: 2D Airfoil-Near Wake





2DML: 2D Mixing Layer

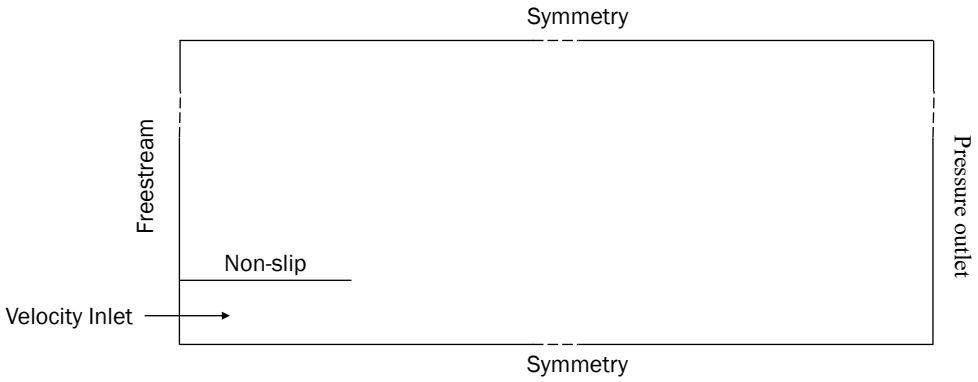


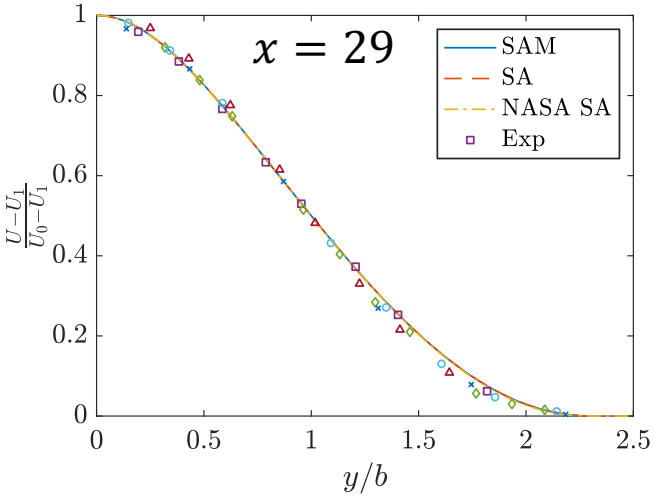


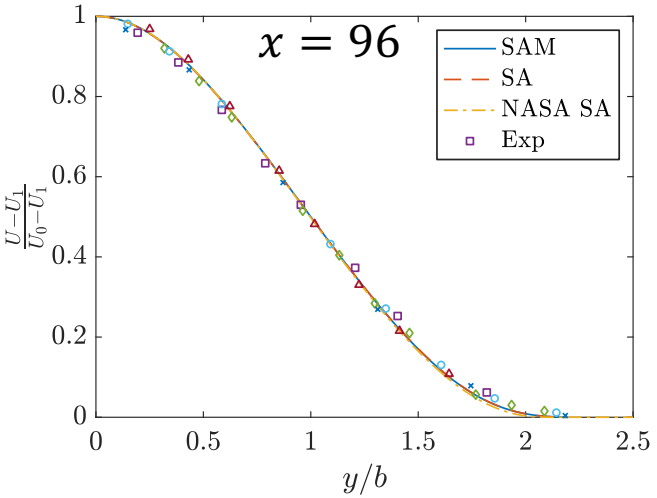
Testing

Not worse for free shear flows.

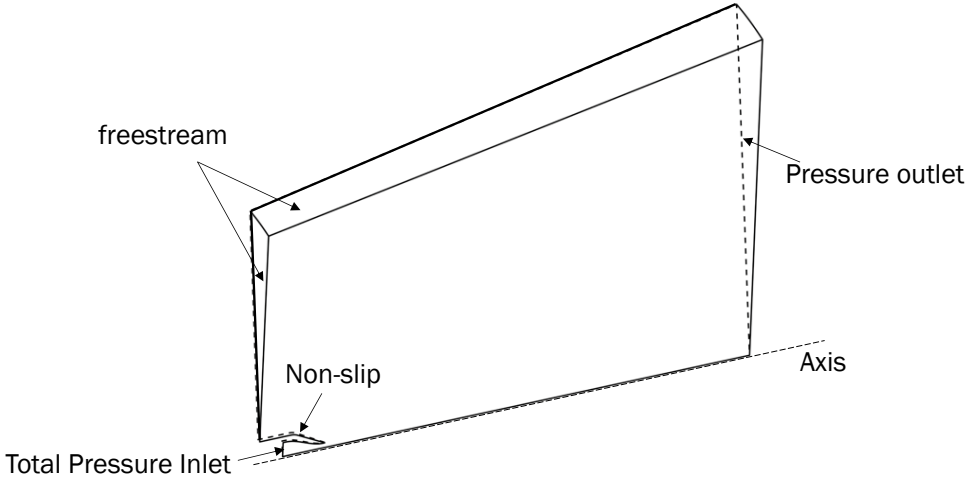
2DCJ: 2D Coflowing jet

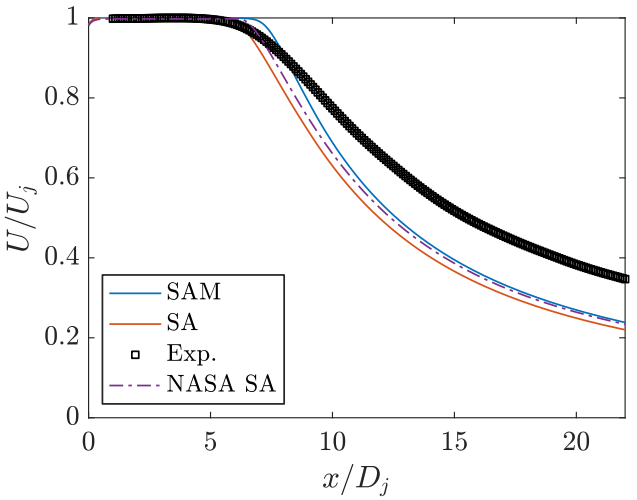


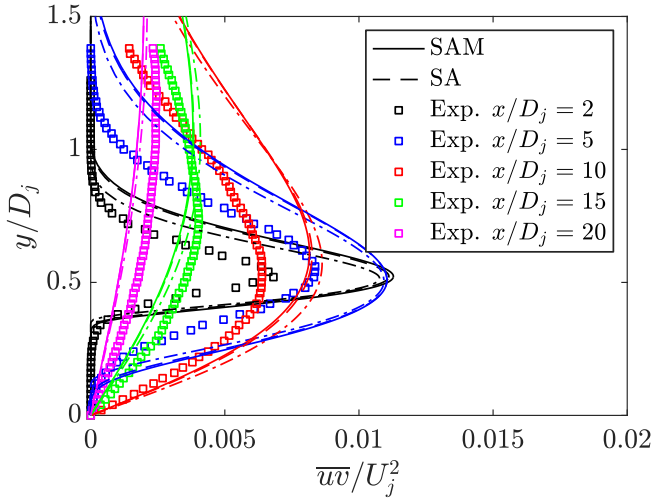




ASJ: Axisymmetric Subsonic Jet



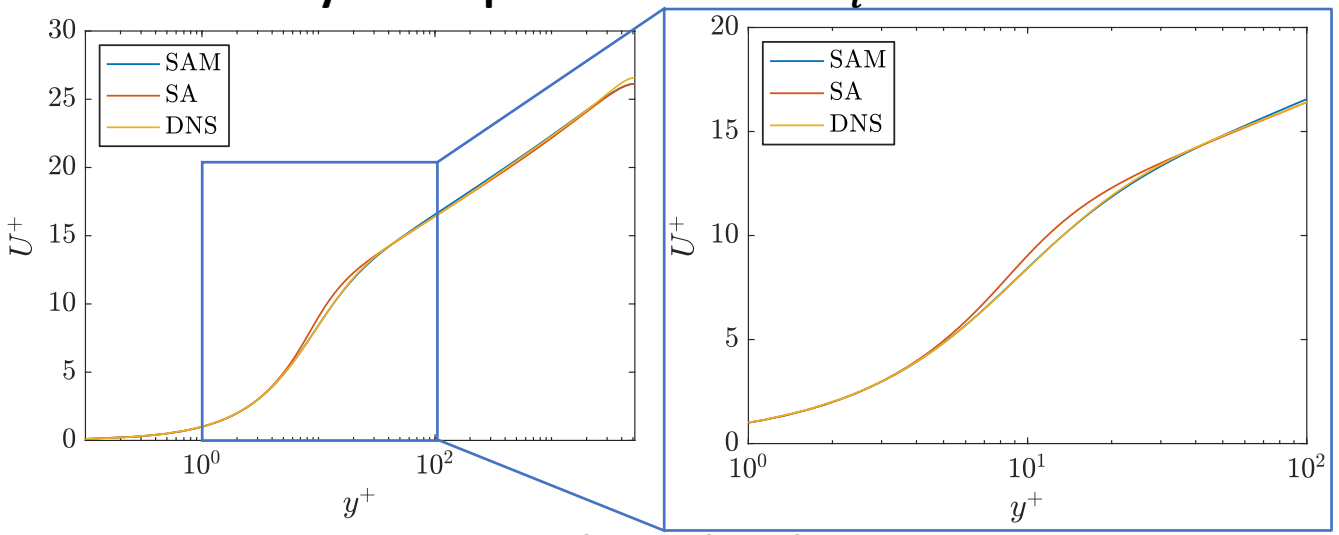




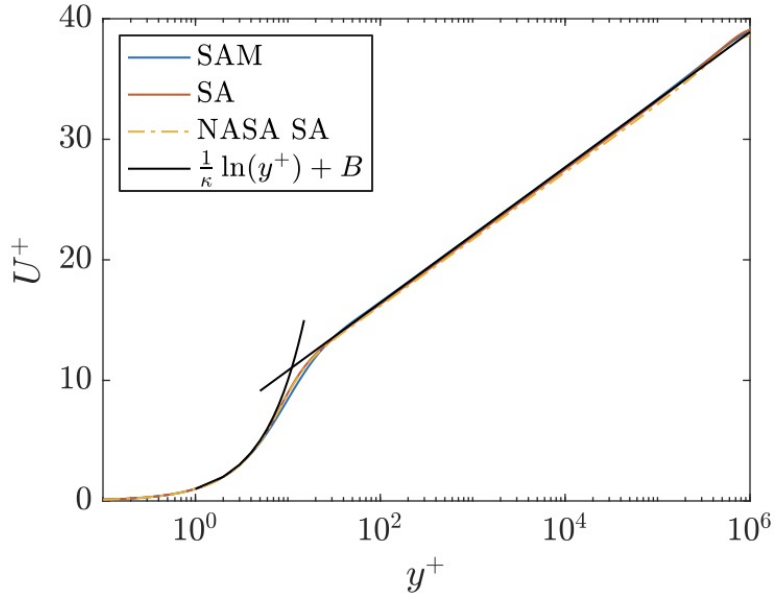
Testing

Protect the law of the wall. Doing better.

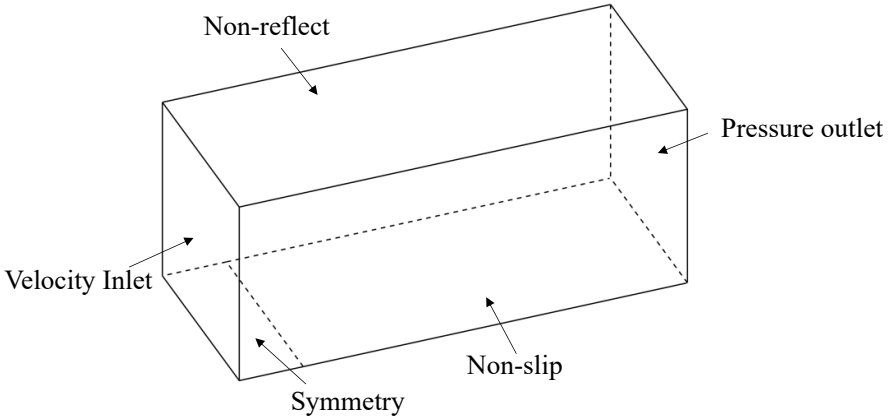
2DFDC: 2D Fully developed channel at $Re_\tau = 5200$

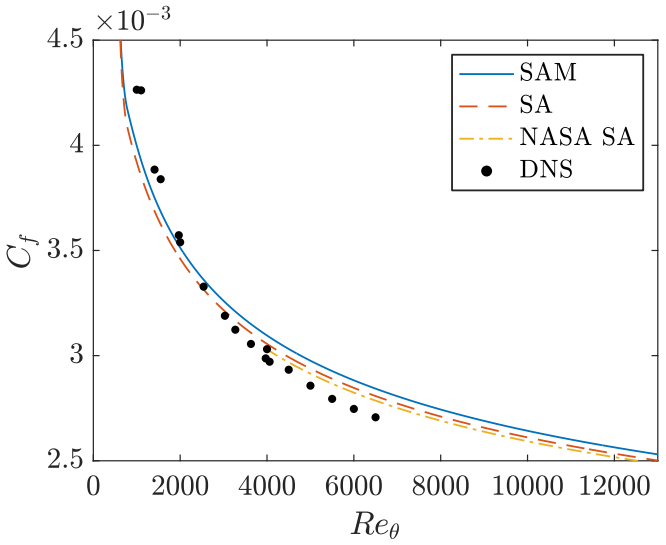


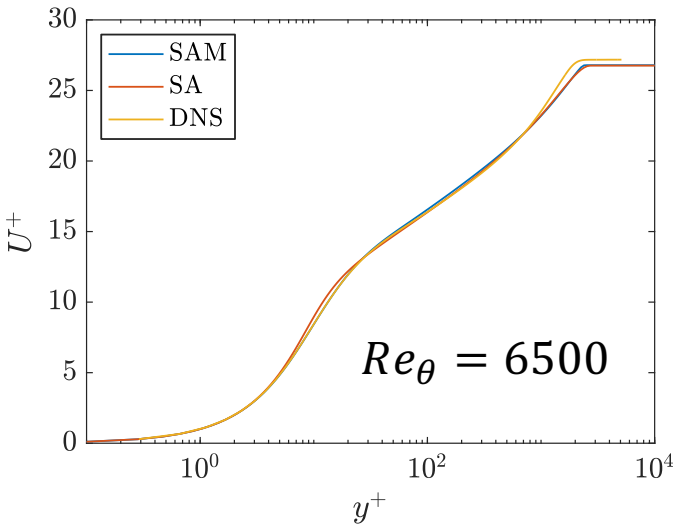
2DFDC: at Re =80,000,000



2DZP: 2D Zero Pressure Gradient Flat Plate

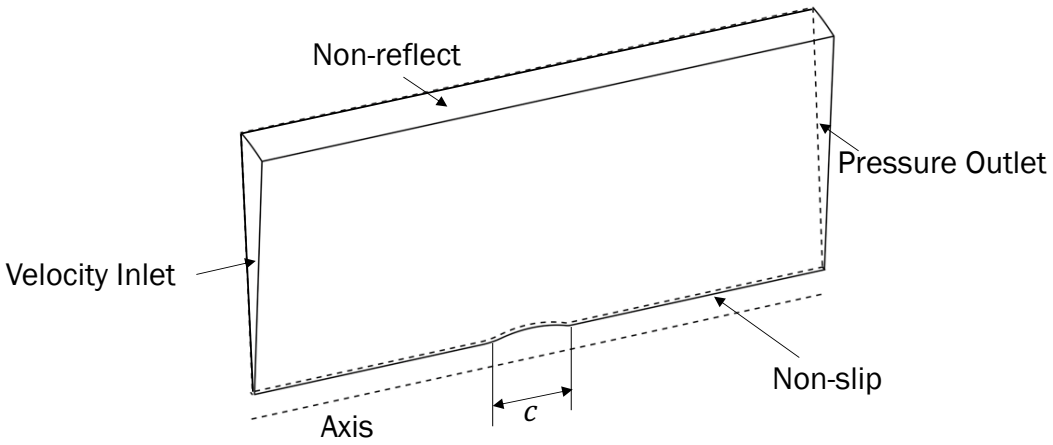


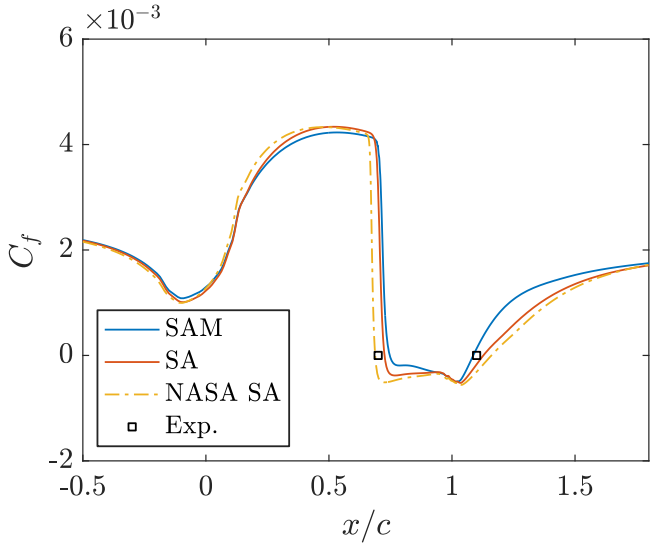


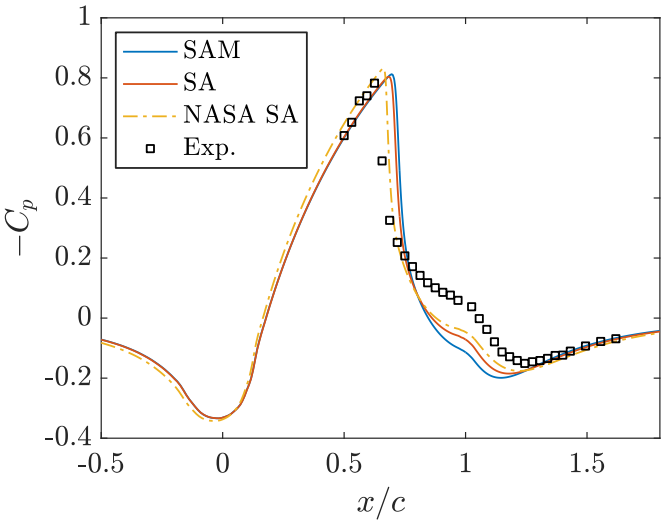


Testing

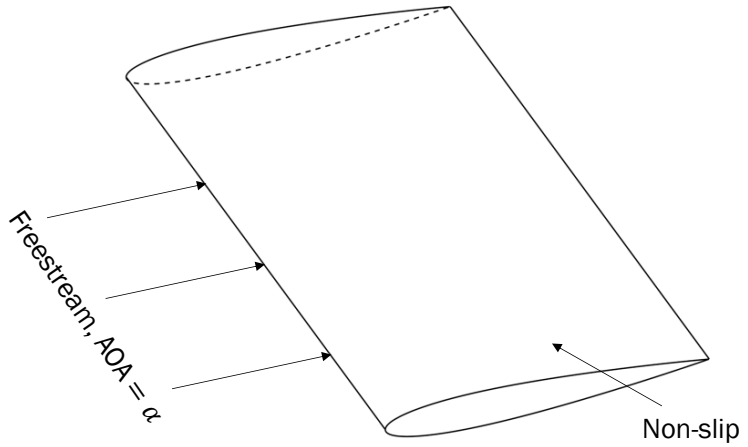
ATB: Axisymmetric Transonic Bump

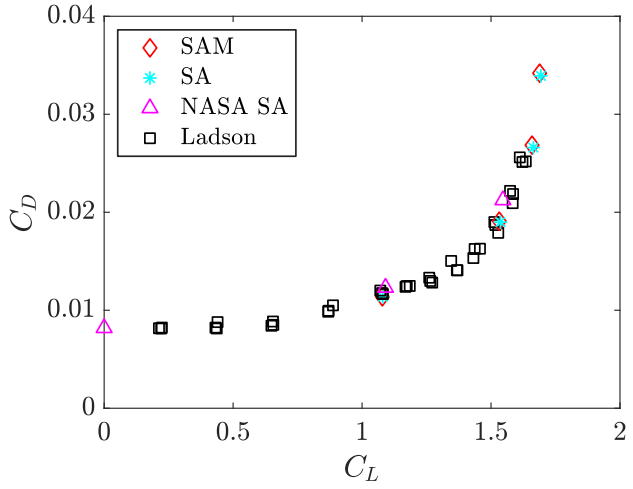


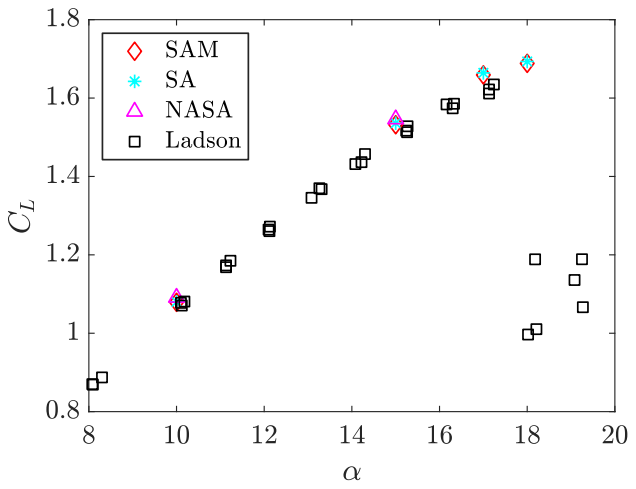




2DN00: NACA 0012 airfoil with different AOA

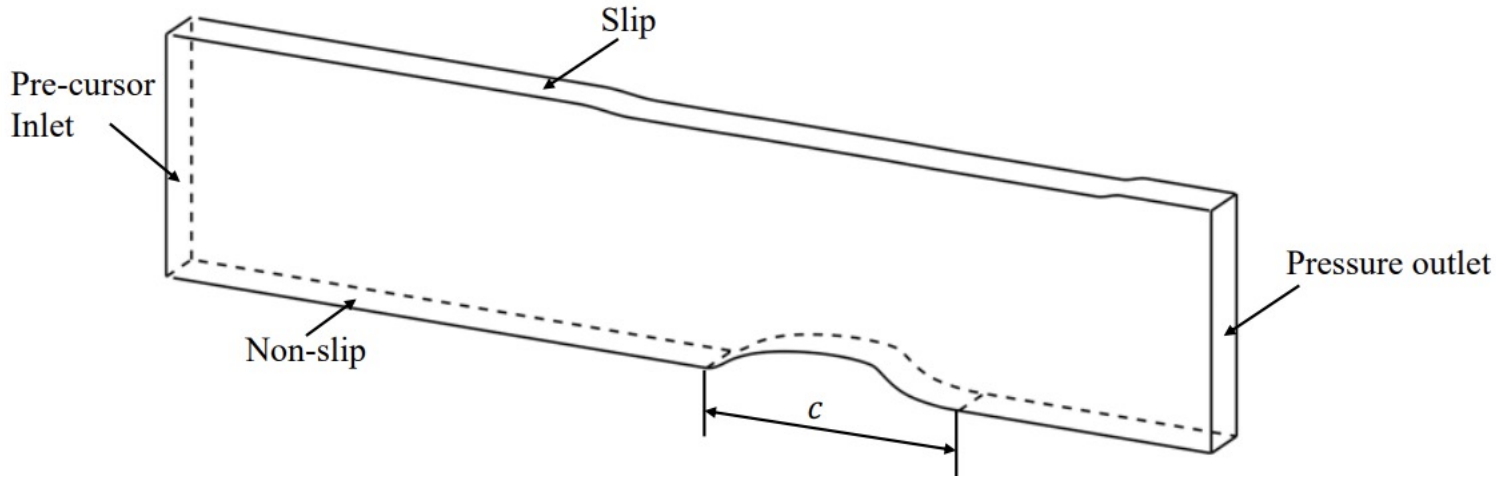


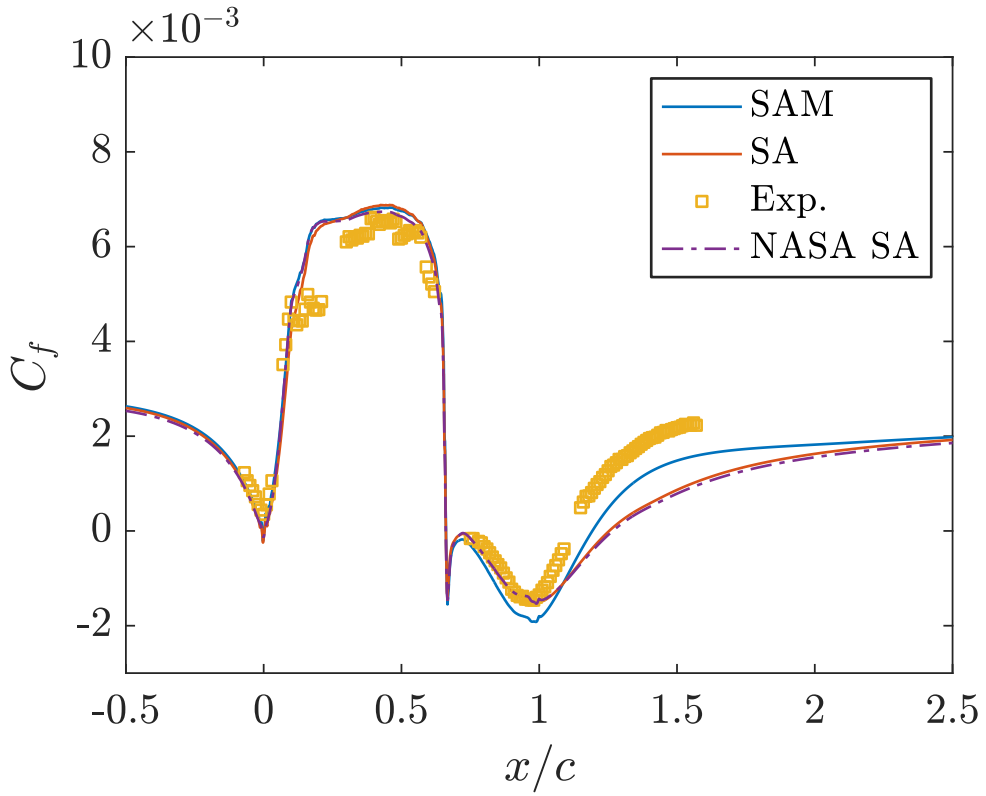


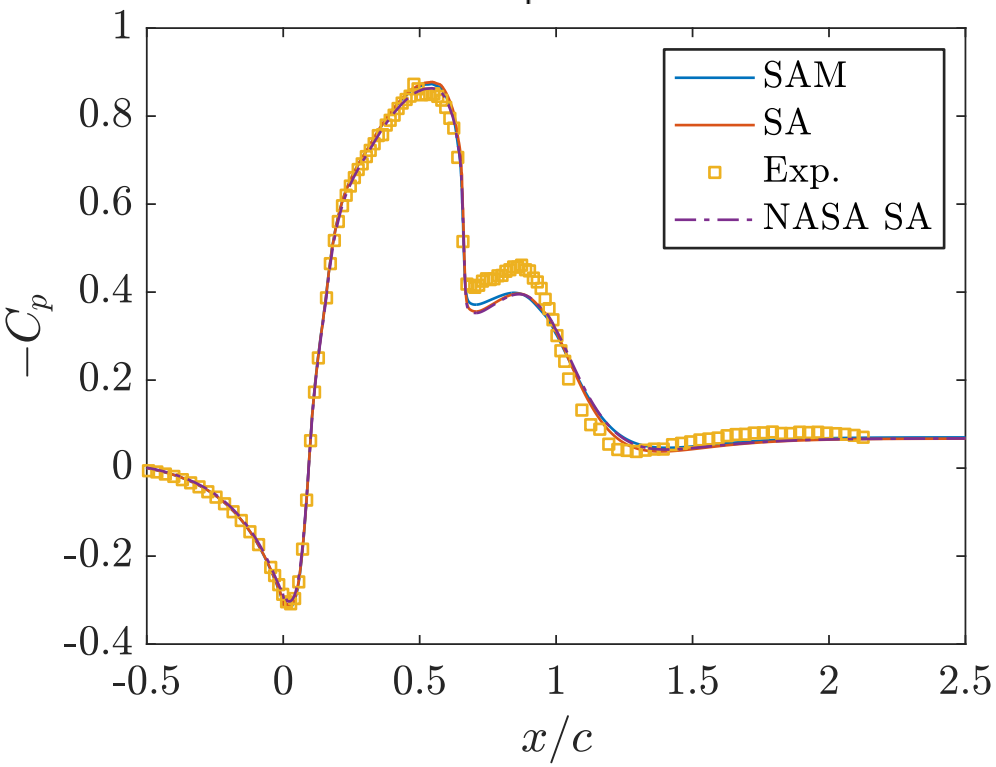


Testing

2DWMH: 2D WALL-Mounted Hump







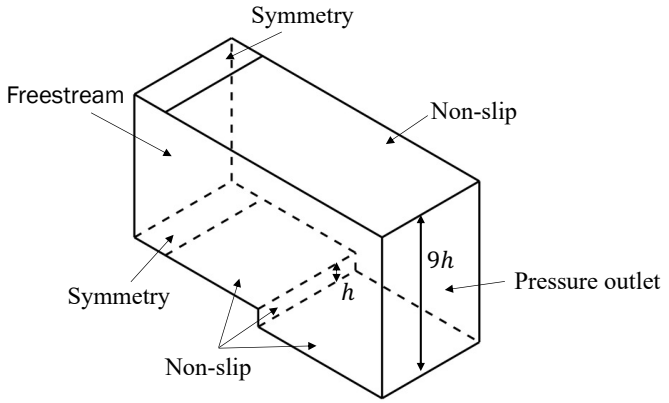
More accurate prediction of reattachment.

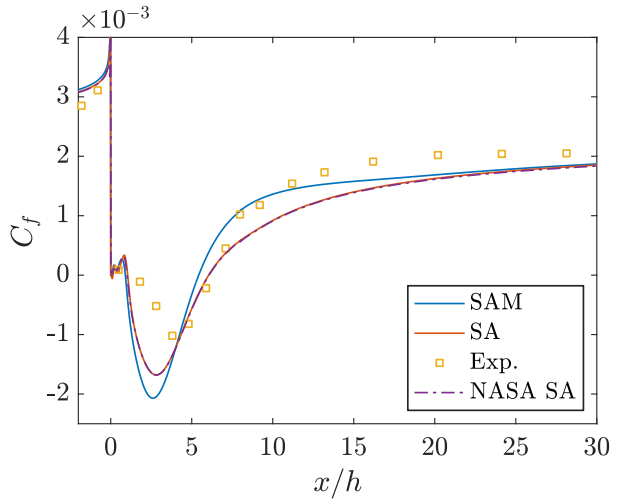
Testing

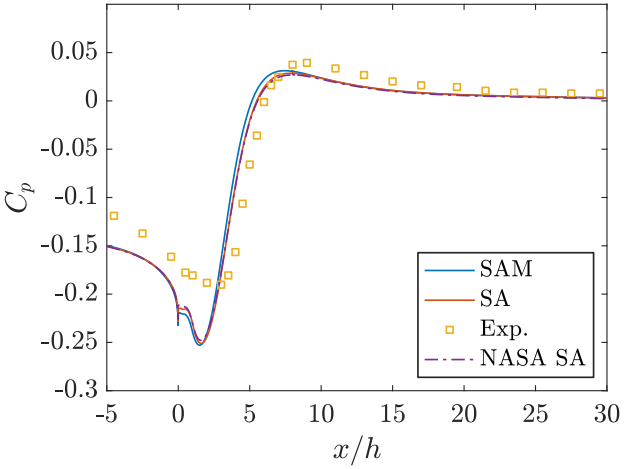
More accurate prediction of reattachment.

BFS: Back-Facing Step

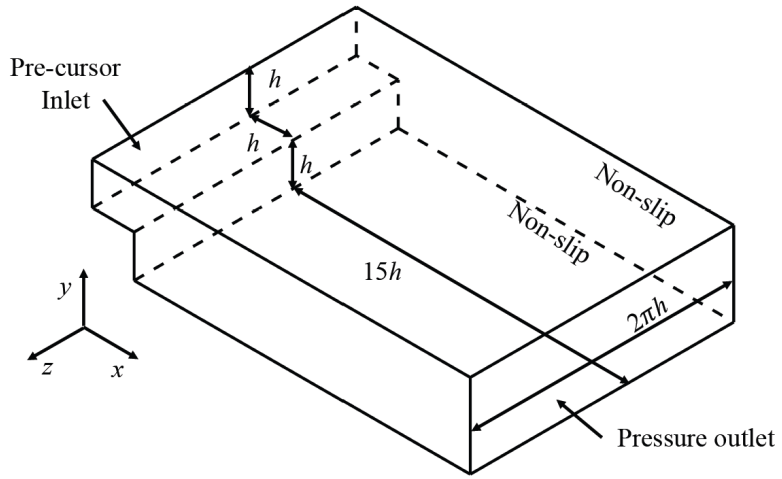
- Velocity recovers faster.

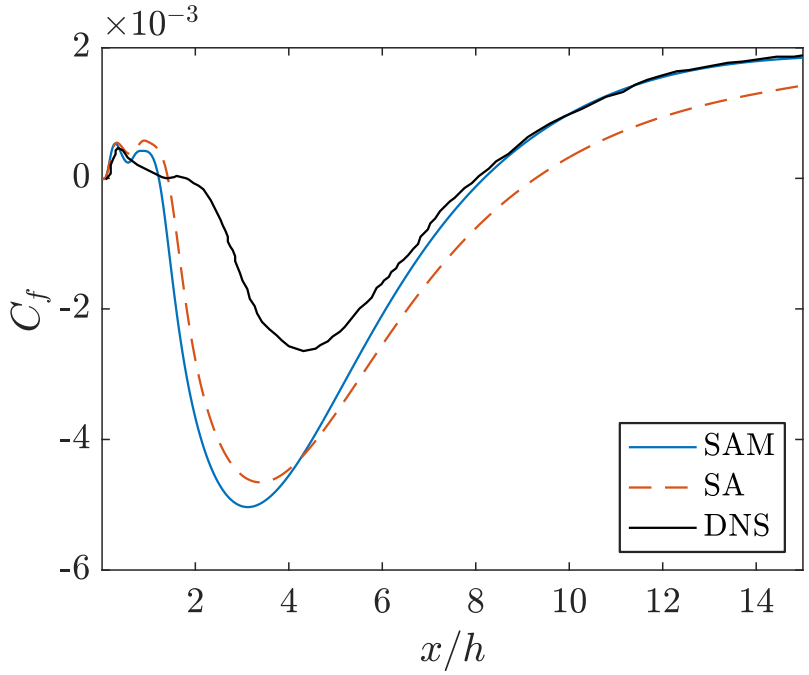






BFS: Back-Facing Step with $2h$





Conclusion

- The development of the SA model is itself progressive.
- ML tools are employed to “re-calibrate” and “re-formulate” the model.
- The progressive reformulation did not suffer from “catastrophic forgetting”.
- The model behaves like the standard SA model in free shear flows.
- The model is more accurate for flow separation.

- We want to continue on this path to account for adverse pressure gradient, surface curvature…
- We want to continue on this path but work on Wilcox k-omg model, FRSM, and the k-omg SST…

Thanks for your attention.

Yang could not make it to the conference.
I will take notes of questions I could not answer.
Yang will get back to you in a few days.

Lookup Table for f_{v1}, f_{v2}, f_w is provided at:
https://github.com/yuanweibin/LookupTableForSAM

χ	0	0.1	0.2	0.3	0.4	0.5	…	59.9	60.0
f_{v1}	0	1.6016e-4	6.6088e-4	0.0015	0.0028	0.0044	…	1	1
f_{v2}	1	0.9000	0.8000	0.7001	0.6004	0.5010	…	0	0

r	0	0.01	0.02	0.03	0.04	0.05	…	1.99	2.00
f_w	0	2.0744e-4	5.8489e-4	0.0011	0.0018	0.0027	…	1.22	1.22

χ																					
f_{v1}	0	0.05	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5	0.55	0.6	0.65	0.7	0.75	0.8	0.85	0.9	0.95	1
	0	3.94E-05	0.00016	0.000372	0.000661	0.001052	0.001527	0.002091	0.002763	0.003522	0.004363	0.005317	0.006368	0.007496	0.008708	0.010031	0.011446	0.012935	0.014486	0.016129	0.017867
		1.05	1.1	1.15	1.2	1.25	1.3	1.35	1.4	1.45	1.5	1.55	1.6	1.65	1.7	1.75	1.8	1.85	1.9	1.95	2
		0.019682	0.021557	0.023484	0.025492	0.027573	0.029714	0.031903	0.034135	0.03643	0.038781	0.04118	0.043616	0.046084	0.048598	0.051156	0.053751	0.056376	0.059024	0.061704	0.064418
		2.05	2.1	2.15	2.2	2.25	2.3	2.35	2.4	2.45	2.5	2.55	2.6	2.65	2.7	2.75	2.8	2.85	2.9	2.95	3
		0.06716	0.069927	0.072712	0.075517	0.078348	0.081201	0.084073	0.086962	0.089864	0.092783	0.095719	0.098672	0.101639	0.104616	0.107604	0.110605	0.113619	0.116645	0.119681	0.122726
											.....										
		27.05	27.1	27.15	27.2	27.25	27.3	27.35	27.4	27.45	27.5	27.55	27.6	27.65	27.7	27.75	27.8	27.85	27.9	27.95	28
		0.999395	0.999454	0.99951	0.999563	0.999614	0.999661	0.999706	0.999748	0.999787	0.999823	0.999856	0.999886	0.999912	0.999935	0.999955	0.999971	0.999983	0.999993	0.999998	1

χ																					
f_{v2}	0	0.05	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5	0.55	0.6	0.65	0.7	0.75	0.8	0.85	0.9	0.95	1
	1	0.947789	0.899999	0.85	0.800018	0.750043	0.700113	0.650226	0.600378	0.550647	0.501044	0.451505	0.40213	0.353031	0.304177	0.255471	0.207032	0.158982	0.111427	0.064191	0.017299
		1.05	1.1	1.15	1.2	1.25	1.3	1.35	1.4	1.45	1.5	1.55	1.6	1.65	1.7	1.75	1.8	1.85	1.9	1.95	2
		-0.02911	-0.0749	-0.12	-0.16464	-0.20875	-0.25218	-0.29478	-0.33651	-0.37764	-0.41808	-0.45769	-0.49633	-0.53393	-0.5708	-0.60689	-0.64207	-0.67619	-0.70916	-0.74126	-0.77255
		2.05	2.1	2.15	2.2	2.25	2.3	2.35	2.4	2.45	2.5	2.55	2.6	2.65	2.7	2.75	2.8	2.85	2.9	2.95	3
		-0.80292	-0.83224	-0.86039	-0.88745	-0.91375	-0.93917	-0.96361	-0.98695	-1.0091	-1.03038	-1.05091	-1.07058	-1.08928	-1.1069	-1.1234	-1.13921	-1.15434	-1.16867	-1.1821	-1.1945
											.....										
		60.05	60.1	60.15	60.2	60.25	60.3	60.35	60.4	60.45	60.5	60.55	60.6	60.65	60.7	60.75	60.8	60.85	60.9	60.95	61
		0.000811	0.000783	0.000755	0.000727	0.000698	0.000668	0.000638	0.000604	0.000568	0.000528	0.000486	0.000442	0.000397	0.000352	0.000307	0.000263	0.00022	0.000179	0.000141	0.000106

r																					
f_w	0	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09	0.1	0.11	0.12	0.13	0.14	0.15	0.16	0.17	0.18	0.19	0.2
	0	0.000207	0.000585	0.001128	0.001833	0.002695	0.003709	0.004873	0.006181	0.007629	0.009213	0.010928	0.012771	0.014737	0.016822	0.019021	0.02133	0.023746	0.026263	0.028878	0.031586
		0.21	0.22	0.23	0.24	0.25	0.26	0.27	0.28	0.29	0.3	0.31	0.32	0.33	0.34	0.35	0.36	0.37	0.38	0.39	0.4
		0.034383	0.037264	0.040226	0.043264	0.046374	0.049552	0.052792	0.056092	0.059447	0.062852	0.066304	0.069797	0.073329	0.076894	0.080487	0.084106	0.087746	0.091402	0.09507	0.098746
		0.41	0.42	0.43	0.44	0.45	0.46	0.47	0.48	0.49	0.5	0.51	0.52	0.53	0.54	0.55	0.56	0.57	0.58	0.59	0.6
		0.102855	0.108202	0.114115	0.120717	0.128018	0.136025	0.144738	0.154155	0.164269	0.175071	0.186545	0.198675	0.21144	0.224815	0.238773	0.253286	0.26832	0.283841	0.299814	0.316199
											.....										
		1.01	1.02	1.03	1.04	1.05	1.06	1.07	1.08	1.09	1.1	1.11	1.12	1.13	1.14	1.15	1.16	1.17	1.18	1.19	1.2
		1.013225	1.029612	1.045892	1.061976	1.077774	1.093198	1.108159	1.122568	1.136335	1.149372	1.16159	1.1729	1.183212	1.192438	1.200489	1.207276	1.212709	1.2167	1.21916	1.22

Lookup Table for f_{v1}, f_{v2}, f_w is provided at:
https://github.com/yuanweibin/LookupTableForSAM

Recalibration

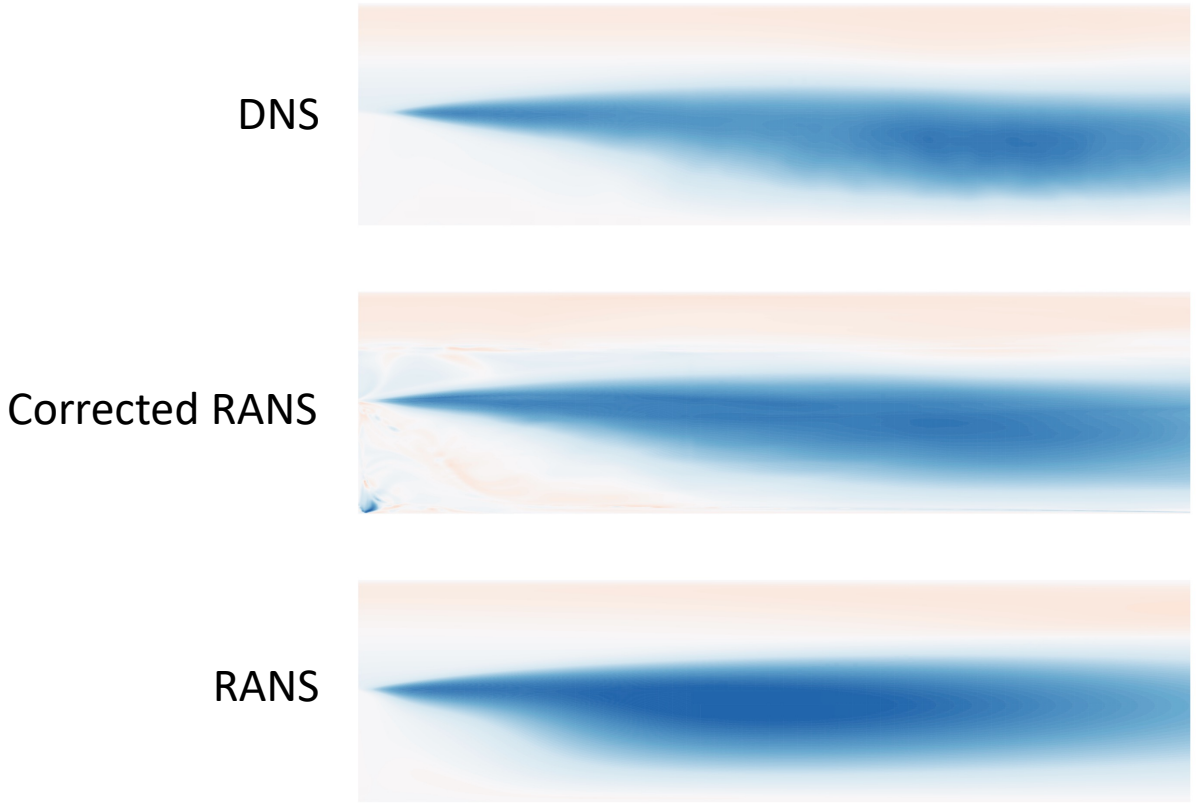
Without considering generalizability

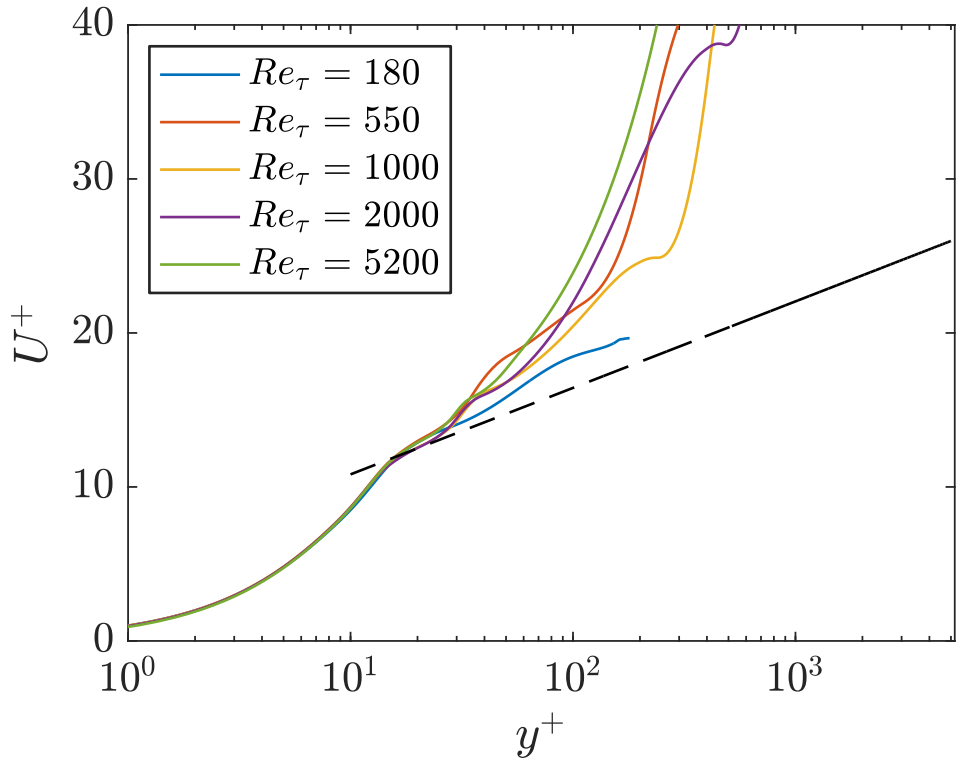
- If we use a BFS case to correct Reynolds stresses
- We will have an extremely good prediction for BFS since the same $\overline{uv}$ as DNS data
- But what will happen if we use this correction on channel flow?

DNS

Corrected RANS

RANS





The ML model must consider generalizability.