mathematics

MDPI

Article

Prediction of Turbulent Boundary Layer Flow Dynamics with Transformers

Rakesh Sarma [1,*], Fabian Hübenthal [2], Eray Inanc [1] and Andreas Lintermann [1]

[1] Forschungszentrum Jülich GmbH, Jülich Supercomputing Centre, Wilhelm-Johnen-Straße, 52425 Jülich, Germany; e.inanc@fz-juelich.de (E.I.); a.lintermann@fz-juelich.de (A.L.)
[2] Institute of Aerodynamics and Chair of Fluid Mechanics, RWTH Aachen University, Wüllnerstraße 5a, 52062 Aachen, Germany; f.huebenthal@aia.rwth-aachen.de
* Correspondence: r.sarma@fz-juelich.de

Abstract: Time-marching of turbulent flow fields is computationally expensive using traditional Computational Fluid Dynamics (CFD) solvers. Machine Learning (ML) techniques can be used as an acceleration strategy to offload a few time-marching steps of a CFD solver. In this study, the Transformer (TR) architecture, which has been widely used in the Natural Language Processing (NLP) community for prediction and generative tasks, is utilized to predict future velocity flow fields in an actuated Turbulent Boundary Layer (TBL) flow. A unique data pre-processing step is proposed to reduce the dimensionality of the velocity fields, allowing the processing of full velocity fields of the actuated TBL flow while taking advantage of distributed training in a High Performance Computing (HPC) environment. The trained model is tested at various prediction times using the Dynamic Mode Decomposition (DMD) method. It is found that under five future prediction time steps with the TR, the model is able to achieve a relative Frobenius norm error of less than 5%, compared to fields predicted with a Large Eddy Simulation (LES). Finally, a computational study shows that the TR achieves a significant speed-up, offering computational savings approximately 53 times greater than those of the baseline LES solver. This study demonstrates one of the first applications of TRs on actuated TBL flow intended towards reducing the computational effort of time-marching. The application of this model is envisioned in a coupled manner with the LES solver to provide few time-marching steps, which will accelerate the overall computational process.

Keywords: turbulent boundary layer; transformer; machinelearning; dynamicmodedecomposition; flow dynamics;time-marching; activedragreduction; spanwise traveling transversal surface waves

MSC: 68T07; 76F40

Citation: Sarma, R.; Hübenthal, F.; Inanc, E.; Lintermann, A. Prediction of Turbulent Boundary Layer Flow Dynamics with Transformers. *Mathematics* **2024**, *12*, 2998. https://doi.org/10.3390/math12192998

Academic Editor: Efstratios Tzirtzilakis

Received: 31 July 2024
Revised: 17 September 2024
Accepted: 18 September 2024
Published: 26 September 2024

1. Introduction

Turbulent flows are encountered in many scientific and engineering problems, such as in aerospace, weather sciences, and biophysical systems. Computational Fluid Dynamics (CFD) simulations to study the turbulent dynamics, need to address multiscale flow features due to the interactions of strong and chaotic fluctuations over a wide range of scales. As the flow REYNOLDS number increases, even smaller scales are developed in the energy cascade as the inertial force overwhelms the viscous force, leading to large scale separation in both space and time. Therefore, fine mesh resolutions are required to obtain accurate solutions. Evidently, time-marching of such turbulent flows with high-fidelity numerical techniques, such as Direct Numerical Simulation (DNS) and wall-resolved Large Eddy Simulation (LES), are prohibitively expensive for many practical use cases. Alternatively, Machine Learning (ML)-based methodologies can potentially offer significant speed-ups for the estimation of turbulent fields [1,2]. Although fully replacing the CFD solver for all time integration steps with ML or a data-driven model is challenging and could lead to the accumulation of errors, ML can still offer computational efficiency through offloading

of a *few* steps. In such a workflow, the idea is to use ML for acceleration rather than as a replacement for the underlying CFD solver. ML techniques have already been widely explored in CFD for various such applications, for instance in model-order reduction [3], super-resolution [4], and also for temporal predictions [5], among many other efforts [6].

For unsteady problems, the Recurrent Neural Network (RNN) architecture is well suited to capture and model temporal dynamics. An RNN consists of hidden layers with a feedback loop, where each layer has an internal state vector, which is combined with the input vector to obtain the output. This output from a hidden layer is fed into the next layer, leading to the *recurrent* architecture, which allows the processing of time signals. However, RNNs suffer from *vanishing* and *exploding* gradients, as described in Bengio et al. [7]. Initial solutions proposed to the gradient problems in RNNs involved techniques such as gradient clipping, and with *truncated backpropagation through time* [8] by truncating the length of backpropagation. In terms of other architectures to improve RNNs, the Long Short-Term Memory (LSTM) architecture [9], which features the so-called *memory cells*, contributed to overcome the vanishing gradient issue, and led to many developments in temporal predictions. Similar to the development of LSTM, the gating mechanism in the Gated Recurrent Unit (GRU) [10] also tackled the vanishing gradient issue. In CFD, both RNNs and LSTMs have been used for temporal prediction, including for turbulent flows [11,12]. Although the vanishing and exploding gradient issues are addressed by these developments, training such networks is usually slow as the network relies on sequential computation, making it difficult to benefit from parallel systems for acceleration. This is a major drawback since the current ML applications are compute- and data-intensive, where parallel architectures, especially with accelerators such as Graphical Processing Units (GPUs), need to be exploited. To solve this issue, the Transformer (TR) architecture introduced in Vaswani et al. [13] employs the attention mechanism to entirely avoid the recurrence relationship to deduce global dependencies. This model relies entirely on a self-attention mechanism (explained in further detail in Section 3), inherently allowing parallel training. TRs have already been widely exploited in the Natural Language Processing (NLP) community [14,15], demonstrating excellent generative and predictive potential. Owing to their large uptake in the NLP community and also in the computer vision [16], there have been developments in the scientific domain as well [17,18].

In CFD, the use of TRs is still at a relatively early stage compared to other ML architectures. In one of the first applications [19], a TR is coupled to a Generative Adversarial Network (GAN) to generate turbulent inflow conditions for Turbulent Boundary Layer (TBL) simulations. For the prediction of temporal dynamics in a Reduced Order Modelling (ROM)-based framework, TRs have been used to time-march compressed representations of the flow field. In Hemmasian and Barati Farimani [20], an AutoEncoder (AE)-based network is used for compression, while, more recently, a β-variational autoencoder network is used with a TR to predict the encoded fields [21], where also the superiority of TR over LSTM is demonstrated. However, compressed representations tend to underestimate the high frequency components during reconstruction and require careful treatment of the hyperparameters that define the network [22]. Nonetheless, these investigations show that TRs can outperform other prediction methods for CFD applications. Also for long temporal sequences, TR has shown excellent prediction ability [19], whereas LSTMs have been shown to reconstruct long-term dependencies only when separately predicting modes corresponding to different frequency ranges [23].

Inspired by these developments surrounding the use and superiority of TRs in estimating CFD fields, this study analyzes the capability of TRs to predict the full velocity field in an actuated TBL problem. For this purpose, an encoder–decoder configuration of a TR architecture is employed. To limit the number of features that the TR model needs to predict, thus reducing the complexity of the self-attention mechanism, the inputs are reshaped into smaller cubic sub-domains. This also allows handling of non-uniform input shapes encountered in CFD simulations where the computational domain changes. The developed model is envisaged to be integrated into a coupled setup including the baseline

LES solver. The idea is to offload a *few* time-marching steps of the LES solver, which allows to achieve significant computational speed-up. The number of time steps over which the TR model is applicable for time-marching offload is based on the accuracy of the model over different prediction time steps. This is analyzed in this study with the Dynamic Mode Decomposition (DMD) method [24–26].

This manuscript demonstrates for the first time, the use of TR to achieve speed-up in time-marching actuated TBL fields. With the use of TR in an envisioned hybrid workflow coupled to a CFD solver, this manuscript provides a methodology to speed-up the computational time of time-marching turbulent fields with an ML-assisted solution, while retaining accuracy similar to the baseline solver. The contribution of this study is the development of a methodology using a TR architecture to offload time-marching steps of a CFD solver, which leads to massive savings in computational resources. To the best knowledge of the authors, this is the first application of TRs to time-marching of an actuated TBL flow problem.

The manuscript is structured as follows. Section 2 provides an overview of the computational setup, details on the numerical solver, and the associated data that is employed for training the TR model. The TR architecture and methodology of the training are discussed in Section 3. The performance of the TR model is shown and analyzed in Section 4. Finally, Section 5 concludes the manuscript with a summary of the findings and provides directions for future research.

2. TBL Problem Formulation

Since the aviation sector accounts for a significant share of energy demand and associated greenhouse gas emissions, and as political goals and rising energy costs pose environmental and economic challenges for aircraft, aerodynamic improvements are needed. A promising technique to actively and therefore adaptively reduce the aerodynamic viscous drag are spanwise traveling transversal surface waves to manipulate the near-wall turbulent boundary layer [27].

As a first step approximation to more realistic and computationally more expensive application scenarios such as aircraft wings, a CFD model based on a validated zero-pressure gradient flat plate configuration plate is selected to study the underlying physics and potential of this active drag reduction technique. For that purpose, wall-resolved LES is performed using the in-house CFD solver m-AIA (https://git.rwth-aachen.de/aia/m-AIA/m-AIA, accessed on 25 September 2024) [28–30], depicted in Figure 1 for the actuated case. The physical domain of the flat plate model is shown for the actuated case, where the dimensions in the Cartesian directions are L_x, L_y, and L_z. The actuation parameters of the spanwise traveling wave are the wavelength λ, the time period T and the amplitude A. At the inflow of the domain, the Reformulated Synthetic Turbulence Generation (RSTG) method is used to initiate a TBL flow [31]. The onset of the surface actuation, analyzed in Albers et al. [28], Fernex et al. [29], is located at x_0, where a fully developed TBL is established. The surface area A_{surf} for the integration of the wall-shear stress τ_w is shaded in gray. Periodic Boundary Conditions (BC) are used in the spanwise direction z, characteristic outflow conditions [32] are applied on the downstream and upper boundaries, and the no-slip condition is imposed on the wall [28].

For the solver, the unsteady compressible Navier–Stokes equations in the Arbitrary Lagrangian-Eulerian (ALE) formulation for time-dependent body-fitted deformable meshes are solved with the structured part of m-AIA. A second-order accurate Finite Volume (FV) approximation of the governing equations is used in which the inviscid fluxes are computed by the Advection Upstream Splitting Method (AUSM) using a Monotonic Upstream-Centered Scheme for Conservation Laws (MUSCL) to reconstruct the cell-center values. The viscous fluxes are discretized by a modified cell-vertex scheme at second-order accuracy. The time integration is performed via a five-stage Runge–Kutta scheme with second order accuracy. Additional volume fluxes are determined to satisfy the Geometry Conservation Law (GCL). According to the Monotonically Integrated Large Eddy Simulation (MILES)

approach, the subgrid dissipative scales of the LES are implicitly modeled by the numerical dissipation of the AUSM scheme [28].

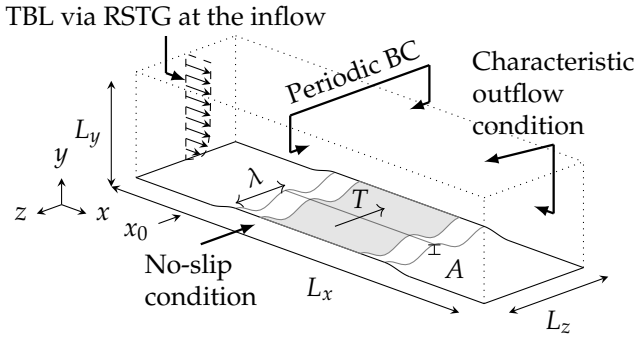


Figure 1. Sketch of the CFD domain with the three-dimensional actuated TBL flow. The actuation of the wall is shown with a spanwise wave, where a no-slip boundary condition is imposed. The gray area indicates the region of interest where the TBL data is extracted.

For the three-dimensional turbulent flow, the wave parameters are non-dimensionalized and given in inner units $(\bullet)^+$ based on the friction velocity u_τ and the kinematic viscosity ν, both at x_0 averaged in the spanwise direction. The actuation is characterized by the non-dimensional actuation parameters, wavelength $\lambda^+ = \lambda u_\tau/\nu$, amplitude $A^+ = A u_\tau/\nu$ and time period $T^+ = T u_\tau^2/\nu$, such that the wall normal coordinate of the spanwise traveling transversal surface wave is described by Equation (1). The piecewise-defined function $g(x^+)$ ensures a smooth spatial transition in the streamwise direction from the non-actuated to the actuated surface area and vise versa, which is given by:

$$y^+_{\text{wall}}\left(x^+, z^+, t^+ | \lambda^+, T^+, A^+\right) = g(x^+)\, A^+ \cos\left(\frac{2\pi}{\lambda^+} z^+ + \frac{2\pi}{T^+} t^+\right). \tag{1}$$

In previous studies, 80 LESs, i.e., one reference/non-actuated and 79 actuated cases, were performed for grid-like distributed actuation parameter combinations within the bounds $[200, 20, 10]^\mathrm{T} \leq [\lambda^+, T^+, A^+]^\mathrm{T} \leq [3000, 120, 78]^\mathrm{T}$ [28,29].

The flow conditions are predefined by the momentum thickness-based REYNOLDS number $\mathrm{Re}_\theta = u_\infty \theta/\nu = 1{,}000$ at x_0 and the MACH number $\mathrm{M} = u_\infty/a = 0.1$ using the free stream velocity u_∞, the momentum-based boundary layer thickness θ, the kinematic viscosity ν, and the ideal gas speed of sound a of air. The mesh resolution is $\Delta x^+ = 12$ in the streamwise, $\Delta y^+_{\text{wall}} = 1$ in the wall-normal direction gradually coarsening off the wall up to $\Delta y^+_{\text{edge}} = 16$ at the boundary layer edge, and $\Delta z^+ = 4$ in the spanwise direction. This yields a DNS-like resolution near the wall, rendering these simulations wall-resolved LESs. Further details on the numerical method, the computational setup, validation of the LES, BC and simulation data points including mesh independence studies and flow field statistics can be found in Albers et al. [28]. For the purpose of this investigation with the TR model, a high-sampling version of the actuated dataset is generated and used for training, where snapshots in every 24 solver time-steps are stored exemplarily for the actuation parameter combination of $\lambda^+ = 1{,}000$, $A^+ = 40$ and $T^+ = 40$.

3. TR Model Architecture

The TR model is required to provide temporal predictions of velocity fields of the TBL flow. As mentioned in Section 1, this is achieved through an encoder–decoder configuration of a TR model that has been adapted from Wu et al. [33]. The architecture is shown in Figure 2, where the number of encoder and decoder layers, and attention heads are chosen after manually tuning these hyperparameters, such that the loss is minimized. Also the input layer of the network is configured such that the non-uniform velocity field tensors can be read as input to the transformer. The encoder consists of an input layer defined by a fully-connected network and a stack of six encoding layers. Positional encoding

is defined by sinusoidal functions. The six layers consist of a self-attention and a fully connected feed-forward layer, each followed by a normalization layer. The self-attention mechanism allows to capture dependencies between tokens, which are the velocity field tensors in the temporal direction in this context. The TR model obtains this dependency by representing each token in the form of query (Q), key (K), and value (V) vectors. These vectors are used to compute attention scores, which represents the relevance of a token with respect to other tokens, thus enabling the capture of not only short-term, but also long-term dependencies. Further details on this can be found in Vaswani et al. [13]. The decoder has a fully connected input layer, again followed by six decoder layers, and a linear mapping to the target sequence at the output layer. In the decoder layers, there is an additional layer that applies self-attention to the encoder outputs. To ensure that the decoder only sees information from the previous time steps, look-ahead masks are applied. The loss function for training the model consists of a Mean-Squared-Error (MSE) term, L_{MSE}, and an additional gradient loss term, L_{grad}, defined by the first-order gradients of the velocity field. For example, if $\mathbf{u}_{t_n}$ and $\tilde{\mathbf{u}}_{t_n}$ are the target and TR-predicted velocity tensors at time instance t_n, and the velocity gradients are $\partial\mathbf{u}_{t_n}/\partial x$, $\partial\mathbf{u}_{t_n}/\partial y$, and $\partial\mathbf{u}_{t_n}/\partial z$ in three directions x, y, and z, the loss L_{total} is defined by the following:

$$L_{\text{total}} = \alpha \cdot \frac{1}{d_u}\sum_{i=1}^{d_u}(\mathbf{u}_{t_n} - \tilde{\mathbf{u}}_{t_n})^2 + \beta \cdot \frac{1}{d_u}\sum_{i=1}^{d_u}\left(\left(\frac{\partial \mathbf{u}_{t_n}}{\partial x} - \frac{\partial \tilde{\mathbf{u}}_{t_n}}{\partial x}\right)^2 + \left(\frac{\partial \mathbf{u}_{t_n}}{\partial y} - \frac{\partial \tilde{\mathbf{u}}_{t_n}}{\partial y}\right)^2 + \left(\frac{\partial \mathbf{u}_{t_n}}{\partial z} - \frac{\partial \tilde{\mathbf{u}}_{t_n}}{\partial z}\right)^2\right), \quad (2)$$

where d_u is the dimension of the velocity tensor summed over all the directions, and α and β are the weights assigned to L_{MSE} and L_{grad}, and $\alpha = 1.0$ and $\beta = 0.06$ [34]. To avoid L_{grad} dominating L_{MSE}, it is scaled in the first 100 epochs such that $L_{\text{grad}}^{\text{scaled}} = L_{\text{grad}}/10^{\lceil log(L_{\text{grad}}/L_{\text{MSE}})\rceil}$.

Assuming that the training dataset consists of velocity field time instances, a subset sequence of $\mathbf{u}_{t_1}, \mathbf{u}_{t_2}, \ldots, \mathbf{u}_{t_m}$ serves as the encoder input, where m is the encoder sequence length. This is shown in Figure 2. In this case, the decoder input consists of velocities at time instances $\mathbf{u}_{t_m}, \ldots, \mathbf{u}_{t_{n-1}}$, and the decoder outputs the velocities at time-instances $\mathbf{u}_{t_{m+1}}, \ldots, \mathbf{u}_{t_n}$, where $n - m$ is the target sequence length. For the investigated TBL problem, these time sequences of the velocity field tensors vary in shape. Specifically, the width of the computational domains used as training data for the TR varies in the spanwise (z) direction. To resolve the non-uniform samples, the velocity field tensors are reshaped to smaller cubic subdomains. In this case, these sub-domains have a dimension of eight computational cells in each direction. This sub-domain size is a hyperparameter, which could influence the accuracy of the developed model. In the present investigation, this hyperparameter is tuned manually. Other cubic sub-domain sizes of four and 16 are also tested, but these lead to higher errors. Sub-domain sizes of more than 16 are not considered, as training costs increase and also the complexity of self-attention increases. Importantly, this reshaping operation allows to limit the number of features that the TR model needs to predict, which significantly reduces the computational complexity of the self-attention mechanism. It is also observed that the TR performance in terms of accuracy is significantly worse when the full velocity field is predicted.

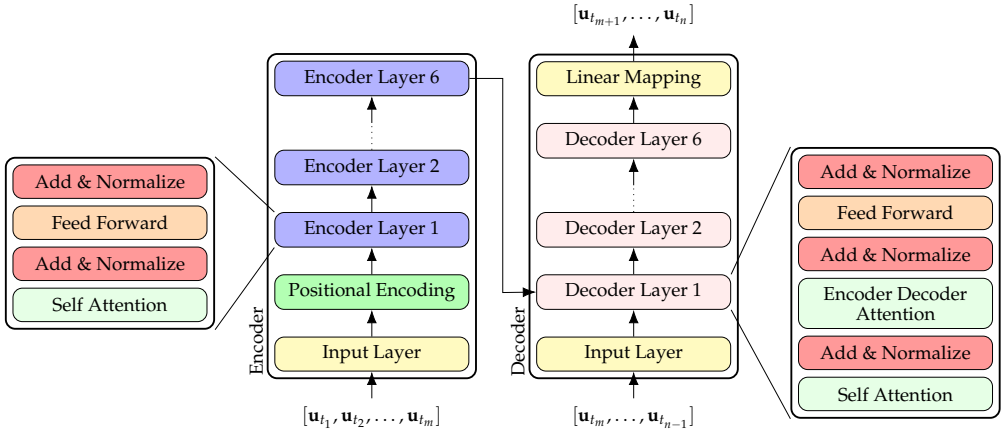


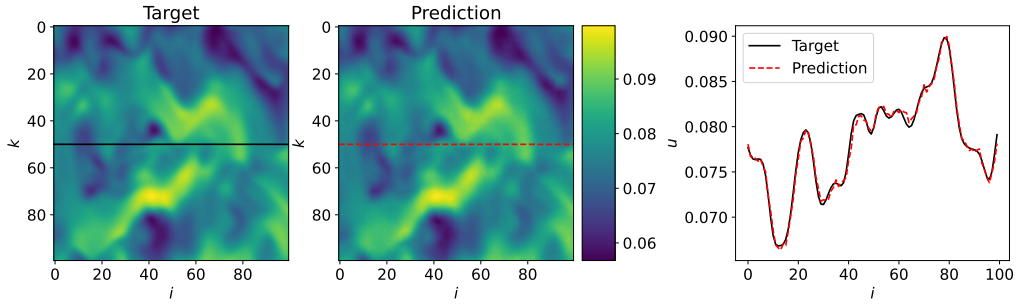
Figure 2. Architecture of the TR model that is used for temporal predictions (adapted from Wu et al. [33]). An encoder–decoder configuration of a transformer model is used, where the input to the encoder are the three-dimensional velocity field tensors, and the decoder outputs the time-marched velocity fields.

Furthermore, 16 attention heads are employed in the network architecture. For regularization to each of the encoder and decoder layers, a dropout value of 0.3 is used. The Adam optimizer [35] is used for training for 1,700 epochs. The learning rate is scaled linearly with the number of workers (in this case, GPUs) that are used for the training. This is done to preserve the accuracy of the model when dealing with large batch sizes encountered in large-scale distributed training, which is a known issue in ML [36,37]. The distributed trainings are conducted with the DeepSpeed framework (https://github.com/microsoft/DeepSpeed, accessed on 25 September 2024), which is provided by the open-source library AI4HPC (https://ai4hpc.readthedocs.io/en/latest/, accessed on 25 September 2024) for scaling ML workloads on High Performance Computing (HPC) systems [38]. The library includes parsing options, which allows configuring the training parameters of the network. For training the TR model, the size of the cubic sub-domains is an important hyperparameter, see discussion above. Another important parameter to consider is the seeding of the network, which is also used to allow deterministic runs. This can easily be specified in AI4HPC with the nseed argument. For exploiting the HPC environment used for training the model, also multiple workers with nworker and prefetching of data with prefetch argument are used. All of these options can be seamlessly specified as input parser arguments to the library [38]. The TR model (https://gitlab.jsc.fz-juelich.de/CoE-RAISE/FZJ/ai4hpc/ai4hpc/-/blob/master/Cases/DS_ATBL_TR.py, accessed on 25 September 2024) and the training data [39] are available open-source.

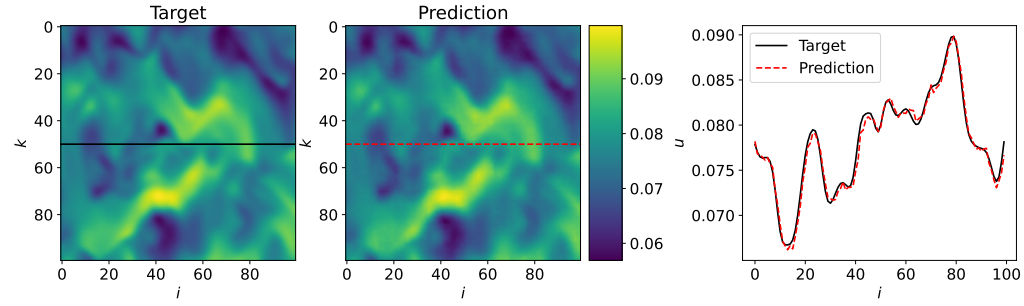
4. Results and Discussion

This section highlights the results obtained with the TR model and the TBL dataset described above. The time-marching capabilities of the TR model are evaluated by increasing the predicted future time steps. For analyzing the performance of the TR model, 10% of the entire dataset is used for test purposes, and the results shown in this section refer to this test dataset. It is to be noted that the convective time for the physical CFD solver time step is $\Delta t_{\mathrm{CFD}} u_\infty / \theta = 1/300$, where Δt_{CFD} is the physical time covered by one CFD solver time step, u_∞ is the far-field velocity outside the boundary layer, and θ is the momentum-based boundary layer thickness at the onset of the actuation at x_0. For training the TR model, a dataset with a high sampling rate is generated, where the velocity fields are stored every 24 physical CFD solver time steps. Hence, during inference, the convective TR time step in terms of the convective CFD solver time step is $\Delta t_{\mathrm{TR}} u_\infty / \theta_0 = 24 \times \Delta t_{\mathrm{CFD}} u_\infty / \theta_0 = 24/300 = 0.08$. As mentioned in Section 3, the TR has a target sequence length of $n - m$ during training, where each of the steps of the target length corresponds to Δt_{TR}.

During inference, the TR can be tested for longer time sequences, as the inference step runs the model iteratively, such that $\tilde{\mathbf{u}}_{t_{n+\Delta t_{\mathrm{TR}}}} = \mathcal{T}(\tilde{\mathbf{u}}_{t_n})$, where $\mathcal{T}$ is the TR model operator. The implementation of the inference step is available in the AI4HPC repository (https://gitlab.jsc.fz-juelich.de/CoE-RAISE/FZJ/ai4hpc/ai4hpc/-/blob/master/Cases/src/networks.py, accessed on 25 September 2024). Exemplary reconstructions provided by the TR model for the streamwise velocity (u) are shown in Figure 3. Qualitatively, it can be observed that the velocity fields predicted by the TR are in close agreement with the LES fields for $\Delta t_{\mathrm{TR}} \leq 2$. The reconstructions above $\Delta t_{\mathrm{TR}} > 10$ are worse compared to $\Delta t_{\mathrm{TR}} = 10$. The larger flow features are clearly reconstructed by the TR model. However, for $\Delta t_{\mathrm{TR}} \geq 5$, the discrepancies in the TR-predicted fields are visible, which can be better observed in the line plots along the right panel. In particular, the values at locations with sharp gradients are poorly estimated. Since the sharpest gradients generally correspond to points in the velocity field with extreme values, such behavior with ML-based models can be expected as ML models with good generalizability capture the mean flow characteristics better compared to the outliers. However, up to $\Delta t_{\mathrm{TR}} = 2$, the line plots are in close agreement. The coefficient of determination (R^2) is also computed, and an average $R^2 \approx 0.99$ is found for $\Delta t_{\mathrm{TR}} \leq 2$, after which it starts to drop. The drop in L_{MSE}, given by the first term in Equation (2), with increasing Δt_{TR} is shown in Figure 4, where the R^2 scores are also shown. As expected, a trend of increase in MSE and decrease in R^2 is observed with increasing Δt_{TR}. Given the reconstruction quality observed in Figure 3c, it can be concluded that the TR predictions are poor for $\Delta t_{\mathrm{TR}} > 5$. However, the time-marching of the TBL flow up to $\Delta t_{\mathrm{TR}} < 5$ already provides high computational speed-up (shown in Section 4.2) with high accuracy. In terms of a measure to quantify the temporal evolution of the flow that the TR model is able to provide, $\Delta t_{\mathrm{TR}} = 5$ would correspond to a convective time of about 0.40, which means about 40% of the boundary layer thickness.

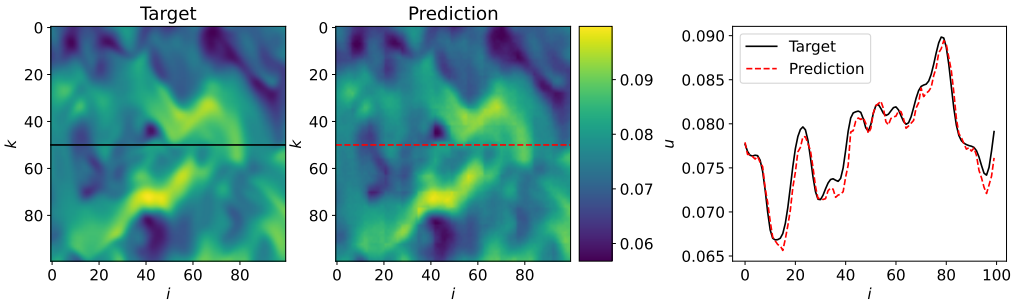


(**a**) $1 \times \Delta t_{\mathrm{TR}}$.

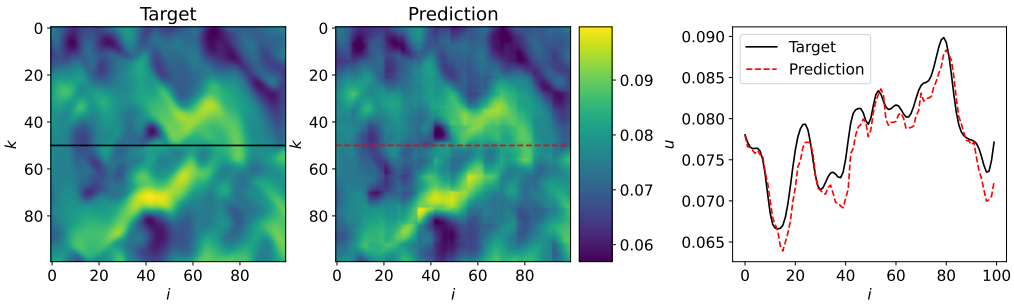


(**b**) $2 \times \Delta t_{\mathrm{TR}}$.

Figure 3. *Cont.*



(**c**) $5 \times \Delta t_{\mathrm{TR}}$.



(**d**) $10 \times \Delta t_{\mathrm{TR}}$.

Figure 3. Cross-section of an exemplary prediction by TR of the streamwise velocity (u) component of the TBL velocity field slice in the wall-normal direction on the i-k plane (Prediction), compared to the original LES field (Target), where i is the streamwise and k is the spanwise direction. The line plot on the right panel shows the velocity at the location shown by the red/black line in the scatter plots. Each subfigure (**a**–**d**) represents increasing TR time-steps Δt_{TR}.

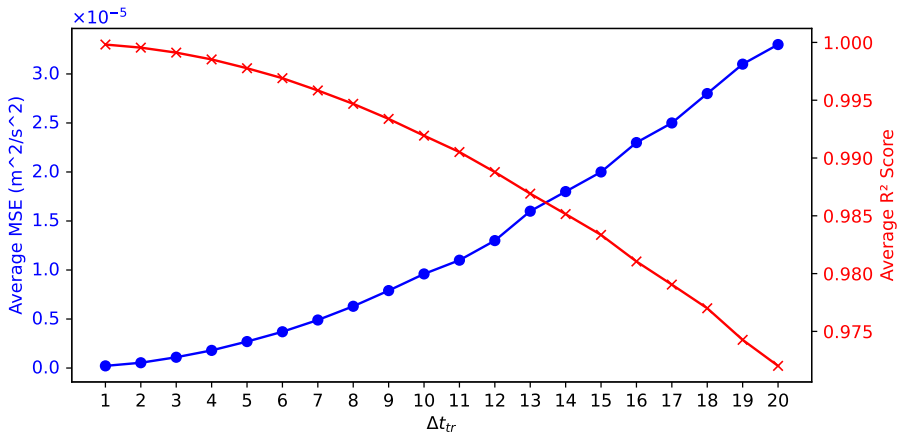


Figure 4. MSE and R^2 score between the LES and TR-predicted velocity fields for different values of Δt_{TR}.

To quantitatively analyze the reconstruction ability of the TR, a modal decomposition of the generated velocity fields is performed in the following Section 4.1. Subsequently, the computational speed-up achieved with the TR for time-marching the velocity fields is compared with the LES execution times in Section 4.2.

4.1. Modal Decomposition Analysis

To further analyze and validate the performance of the TR model for predicting dynamics, the DMD method is used to compare the LES- and TR-predicted fields. DMD is a data-driven method, extensively used for analyzing dynamical systems, especially for high-dimensional data. It has been used widely in the field of CFD and turbulent dynamics to extract coherent structures and understand their temporal evolution [40].

DMD decomposes the behavior of the system into modes and associated frequencies, which allow the interpretation of the evolution of the system in terms of the dominant modes. In particular, for dynamical systems, comparison of the temporal behavior shown by the LES and the TR-based DMD modes provides an assessment if the generated predictions by the TR are stable and remain close to the expected behavior. Here, the modal information extracted from DMD is exploited to measure the accuracy of the TR model with respect to the baseline LES. The open-source Python package PyDMD [41,42] is used to extract the modes and their associated eigenvalues. The DMD modes $\phi_p, p \in \{\text{LES}, \text{TR}\}$ and eigenvalues $\lambda_p, p \in \{\text{LES}, \text{TR}\}$ from the LES and TR fields are compared. Furthermore, the relative Frobenius norm error of the discrepancy between LES and TR fields, given by

$$||\epsilon_{F,rel}|| = \frac{||\phi_{\text{LES}}||_F - ||\phi_{\text{TR}}||_F}{||\phi_{\text{LES}}||_F}, \tag{3}$$

is used to quantify the agreement between the DMD modes. The *mode dynamics* $\alpha_r(t_g)$ for mode r at time t_g is also shown to compare the temporal coefficients that explain the time evolution of each mode.

Figure 5 shows a contour plot of the mode shapes derived from the DMD of the LES and TR fields at $\Delta t_{\text{TR}} = 2$. It can be seen that the TR modes are in close qualitative agreement with the LES modes. The TR-predicted contours are less smooth compared to the LES contours, but the discrepancy is minimal, and these probably arise from the smallest flow structures that are not captured by the TR. In order to further analyze the differences, a quantitative analysis is provided in Table 1 for the dominant mode, where $||\epsilon||_{F,rel}$, λ_p, and $\alpha_r(t_g)$ are compared for the LES and TR at different values of Δt_{TR}. It is observed that for $\Delta t_{\text{TR}} < 5$, $||\epsilon||_{F,rel} < 5\%$ is obtained, which shows the excellent reconstruction ability of the TR model. Close agreement between the modes signify that the dominant flow features are accurately captured by the TR, which are responsible for maximum energy content of the system. For $\Delta t_{\text{TR}} \geq 5$, the model starts to show worse performance and $||\epsilon||_{F,rel}$ gives unacceptable values. In terms of λ_p and $\alpha_r(t_g)$, the values are in close agreement until $\Delta t_{\text{TR}} < 5$. The agreement in the mode dynamics suggests that the dynamic evolution of the velocity field predicted by the TR agrees closely with the LES fields, thus validating its use for making temporal predictions.

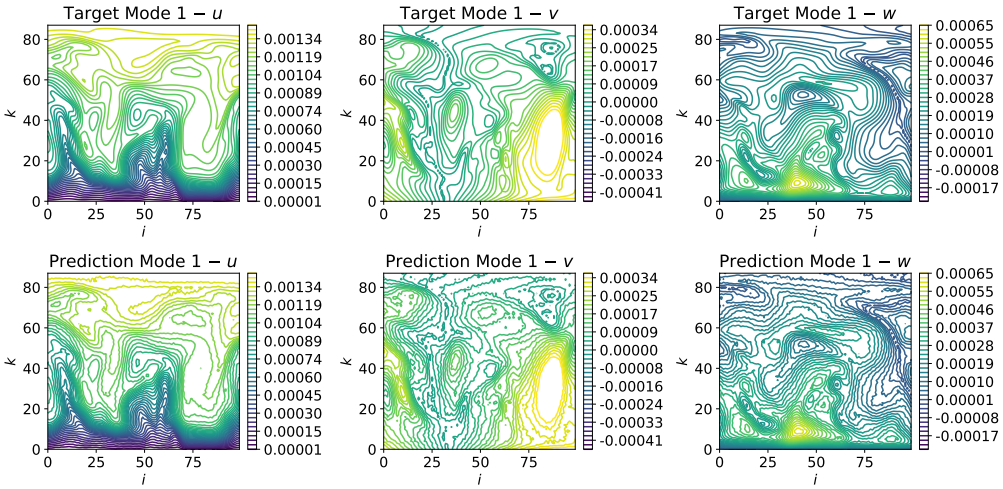


Figure 5. Contour plots of the first mode from DMD of the LES (Target Mode, top panel) and of the TR (Prediction Mode, bottom panel) for the streamwise velocity u (**left**), wall-normal velocity v (**middle**), and the spanwise velocity w (**right**).

Table 1. Eigenvalues, mode dynamics, and relative Frobenius norm between the modes of dominant rank obtained from the DMD of the LES- and TR-predicted velocity fields.

Δt_{TR}	2	3	4	5	8	10	15
$\|\|\epsilon\|\|_{F,rel}$	0.017	0.039	0.030	0.053	0.063	0.082	0.124
λ_{LES}	0.9995	0.9961	0.9966	0.9963	0.9965	0.9965	0.9963
λ_{TR}	0.9981	0.9952	0.9952	0.9952	0.9952	0.9952	0.9952
$\alpha_1(t_1)$ − LES	71.37	−71.47	−71.57	−71.51	−71.69	−71.58	−72.16
$\alpha_1(t_1)$ − TR	71.24	−71.26	−71.38	−71.26	−71.35	−71.20	−71.42
$\alpha_1(t_2)$ − LES	71.33	−71.19	−71.33	−71.25	−71.44	−71.32	−71.89
$\alpha_1(t_2)$ − TR	71.11	−70.92	−71.05	−70.92	−71.01	−70.86	−71.07

4.2. Computational Time per Snapshot and Memory Analysis

In this subsection, a comparison of the computational time and memory usage during inference for the TR and the LES time-marching step with $\Delta t_{\mathrm{TR}} = 1$ is provided. While the training and inference of the TR model is performed on the JURECA system [43], the TR solver uses the HAWK system (https://www.hlrs.de/de/loesungen/systeme/hpe-apollo-hawk, accessed on 25 September 2024). Both the systems feature two 64-core AMD EPYC 7742 Central Processing Units (CPUs) (https://www.amd.com/en/products/cpu/amd-epyc-7742, accessed on 25 September 2024), whereas the HAWK system consists of 4096 compute nodes. The JURECA-DC used in this work contains 192 accelerated nodes, where each node contains four NVIDIA A100 GPUs. The nodes are interconnected via two InfiniBand HDR adapters (https://www.mellanox.com/pdf/whitepapers/IB_Intro_WP_190.pdf, accessed on 25 September 2024). The system peak performances for HAWK and JURECA are 26 Petaflops and 3.54 (CPU) + 14.98 (GPU) Petaflops. The LES solver uses CPUs for the computations, while the TR model uses GPUs. The comparison is made based on the statistics for a single node. The LES computations are performed on 64 nodes, while the TR inference uses one node. Assuming a perfect scaling, the wall-time on a single node and memory usage are reported in Table 2. As can be seen, the inference time of the TR is about 53 times faster than the time-marching step of the LES solver. In terms of memory consumption as well, the TR is found to utilize almost 1100 times less resources. These results show the excellent computational performance of the TR over the LES solver. In practice, the computational gain achieved even with $\Delta t_{\mathrm{TR}} = 1$ is significant for long simulation times.

Table 2. Computational time per snapshot saved every 24 LES solver time step and total memory usage of LES and TR.

	LES	TR
Wall-time (node seconds)	36.48	0.685
Memory (GiB)	466.69	0.422

5. Conclusions

A TR architecture was exploited to time-march velocity fields in a TBL flow problem. The model is based on an encoder–decoder configuration, where a reconstruction strategy was proposed to handle non-uniform inputs and reduce the computational complexity of the self-attention mechanism. The TR-predicted fields were analyzed with the DMD method, and a prediction error of less than 5% was achieved for a horizon of five future TR time steps. Furthermore, a computational performance comparison between the LES and the TR revealed that significant computational savings of up to about 53 times were possible during inference, while consuming 1100 times less memory. This study provides a

new strategy for achieving computational speed-ups for time-marching TBL fields with a TR architecture in a hybrid setup with a traditional CFD solver.

Ongoing work is directed towards coupling the transformer model with the LES solver in m-AIA. This involves checking physical plausibility by tracking possible physical imbalances and also testing the acceleration of time-stepping operations for predicting future velocity fields in a fully coupled scenario. The current work is intended towards the design and optimization of the actuation parameters in the concerned TBL problem. Therefore, future work is focused on examining how the transformer model generalizes to different combinations of actuation parameters. However, if such a model is to be used for a generic flow problem that is applicable across a wider range of Reynolds and Mach numbers (the main factors influencing the flow conditions), future extensions to the TR model will need to check for their generalizability across different flow conditions. Also, more realistic setups such as airfoil simulations have to be considered, where, additionally, the angle of attack as an influential factor will be introduced.

Author Contributions: Conceptualization, R.S., F.H. and E.I.; methodology, R.S. and F.H.; software, R.S. and E.I.; validation, R.S. and F.H.; formal analysis, R.S. and F.H.; writing—original draft preparation, R.S. and F.H.; writing—review and editing, all; supervision, A.L.; project administration, A.L.; funding acquisition, A.L. All authors have read and agreed to the published version of the manuscript.

Funding: The research leading to these results has been conducted in the CoE RAISE project, which receives funding from the European Union's Horizon 2020—Research and Innovation Framework Programme H2020-INFRAEDI-2019-1 under grant agreement no. 951733. The authors gratefully acknowledge the computing time granted by the JARA Vergabegremium and provided on the JARA Partition part of the supercomputer JURECA at Forschungszentrum Jülich, and by the Gauss Centre for Supercomputing e.V. (www.gauss-centre.eu) and provided on the GCS Supercomputer Hazel Hen and Hawk at Höchstleistungsrechenzentrum Stuttgart (www.hlrs.de).

Data Availability Statement: The data and models are available open-source. The models are available in the AI4HPC repository, linked in the manuscript text. The data for training the model have also been released open-source and linked in the manuscript text.

Acknowledgments: The authors acknowledge the contribution of M. Albers and W. Schröder from RWTH Aachen University for providing the LES dataset.

Conflicts of Interest: The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

Abbreviations

The following abbreviations are used in this manuscript:

AE	AutoEncoder
AUSM	Advection Upstream Splitting Method
ALE	Arbitrary Lagrangian–Eulerian
BC	Boundary Conditions
CFD	Computational Fluid Dynamics
CPU	Central Processing Unit
DMD	Dynamic Mode Decomposition
DNS	Direct Numerical Simulation
FV	Finite Volume
GAN	Generative Adversarial Network
GCU	Geometry Conservation Law
GPU	Graphical Processing Unit
GRU	Gated Recurrent Unit
HPC	High-Performance Computing
LES	Large Eddy Simulation
NLP	Natural Language Processing
LSTM	Long Short-Term Memory

MILES	Monotonically Integrated Large Eddy Simulation
ML	Machine Learning
MSE	Mean-Squared-Error
MUSCL	Monotonic Upstream-Centered Scheme for Conservation Laws
ROM	Reduced Order Modeling
RNN	Recurrent Neural Network
RSTG	Reformulated Synthetic Turbulence Generation
TBL	Turbulent Boundary Layer
TR	Transformer

References

1. Duraisamy, K.; Iaccarino, G.; Xiao, H. Turbulence Modeling in the Age of Data. *Annu. Rev. Fluid Mech.* **2019**, *51*, 357–377. [CrossRef]
2. Sandeep Pandey, J.S.; Sreenivasan, K.R. A perspective on machine learning in turbulent flows. *J. Turbul.* **2020**, *21*, 567–584. [CrossRef]
3. Mendez, M.A. Linear and nonlinear dimensionality reduction from fluid mechanics to machine learning. *Meas. Sci. Technol.* **2023**, *34*, 042001. [CrossRef]
4. Fukami, K.; Fukagata, K.; Taira, K. Super-resolution analysis via machine learning: A survey for fluid flows. *Theor. Comput. Fluid Dyn.* **2023**, *37*, 421–444. [CrossRef]
5. Buaria, D.; Sreenivasan, K.R. Forecasting small-scale dynamics of fluid turbulence using deep neural networks. *Proc. Natl. Acad. Sci. USA* **2023**, *120*, e2305765120. [CrossRef]
6. Brunton, S.L.; Noack, B.R.; Koumoutsakos, P. Machine Learning for Fluid Mechanics. *Annu. Rev. Fluid Mech.* **2020**, *52*, 477–508. [CrossRef]
7. Bengio, Y.; Simard, P.; Frasconi, P. Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Netw.* **1994**, *5*, 157–166. [CrossRef]
8. Williams, R.J.; Zipser, D. A Learning Algorithm for Continually Running Fully Recurrent Neural Networks. *Neural Comput.* **1989**, *1*, 270–280. [CrossRef]
9. Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef]
10. Cho, K.; van Merriënboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; Bengio, Y. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar, 25–29 October 2014; Moschitti, A., Pang, B., Daelemans, W., Eds.; pp. 1724–1734. [CrossRef]
11. Srinivasan, P.A.; Guastoni, L.; Azizpour, H.; Schlatter, P.; Vinuesa, R. Predictions of turbulent shear flows using deep neural networks. *Phys. Rev. Fluids* **2019**, *4*, 054603. [CrossRef]
12. Eivazi, H.; Guastoni, L.; Schlatter, P.; Azizpour, H.; Vinuesa, R. Recurrent neural networks and Koopman-based frameworks for temporal predictions in a low-order model of turbulence. *Int. J. Heat Fluid Flow* **2021**, *90*, 108816. [CrossRef]
13. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, L.; Polosukhin, I. Attention Is All You Need. *arXiv* **2017**, arXiv:1706.03762.
14. Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. Language Models are Few-Shot Learners. In Proceedings of the Advances in Neural Information Processing Systems, Virtual, 6–12 December 2020 ; Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., Lin, H., Eds.; Curran Associates, Inc.: New York, NY, USA, 2020; Volume 33, pp. 1877–1901.
15. Huang, Y.; Xu, J.; Jiang, Z.; Lai, J.; Li, Z.; Yao, Y.; Chen, T.; Yang, L.; Xin, Z.; Ma, X. Advancing Transformer Architecture in Long-Context Large Language Models: A Comprehensive Survey. *arXiv* **2023**, arXiv:2311.12351.
16. Khan, S.; Naseer, M.; Hayat, M.; Zamir, S.W.; Khan, F.S.; Shah, M. Transformers in Vision: A Survey. *ACM Comput. Surv.* **2022**, *54*. [CrossRef]
17. Geneva, N.; Zabaras, N. Transformers for modeling physical systems. *Neural Netw.* **2022**, *146*, 272–289. [CrossRef]
18. Hassanian, R.; Myneni, H.; Helgadóttir, A.; Riedel, M. Deciphering the dynamics of distorted turbulent flows: Lagrangian particle tracking and chaos prediction through transformer-based deep learning models. *Phys. Fluids* **2023**, *35*, 075118, [CrossRef]
19. Yousif, M.Z.; Zhang, M.; Yu, L.; Vinuesa, R.; Lim, H. A transformer-based synthetic-inflow generator for spatially developing turbulent boundary layers. *J. Fluid Mech.* **2023**, *957*, A6. [CrossRef]
20. Hemmasian, A.; Barati Farimani, A. Reduced-order modeling of fluid flows with transformers. *Phys. Fluids* **2023**, *35*, 057126. [CrossRef]
21. Solera-Rico, A.; Vila, C.S.; Gómez-López, M.; Wang, Y.; Almashjary, A.; Dawson, S.T.M.; Vinuesa, R. β-Variational autoencoders and transformers for reduced-order modelling of fluid flows. *Nat. Commun.* **2024**, *15*, 1361. [CrossRef]
22. Higgins, I.; Matthey, L.; Pal, A.; Burgess, C.P.; Glorot, X.; Botvinick, M.M.; Mohamed, S.; Lerchner, A. β-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework. In Proceedings of the International Conference on Learning Representations, San Juan, Puerto Rico, 2–6 May 2016.

23. Borrelli, G.; Guastoni, L.; Eivazi, H.; Schlatter, P.; Vinuesa, R. Predicting the temporal dynamics of turbulent channels through deep learning. *Int. J. Heat Fluid Flow* **2022**, *96*, 109010. [CrossRef]
24. Schmid, P.J. Dynamic mode decomposition of numerical and experimental data. *J. Fluid Mech.* **2010**, *656*, 5–28. [CrossRef]
25. Tu, J.H.; Rowley, C.W.; Luchtenburg, D.M.; Brunton, S.L.; Kutz, J.N. On Dynamic Mode Decomposition: Theory and Applications. *J. Comput. Dyn.* **2014**, *1*, 391–421. [CrossRef]
26. Kutz, J.N.; Brunton, S.L.; Brunton, B.W.; Proctor, J.L. Dynamic Mode Decomposition: Data-Driven Modeling of Complex Systems. In *SIAM Other Titles in Applied Mathematics*; Society for Industrial and Applied Mathematics (SIAM): Philadelphia, PA, USA; 2016.
27. Albers, M.; Schröder, W. Lower drag and higher lift for turbulent airfoil flow by moving surfaces. *Int. J. Heat Fluid Flow* **2021**, *88*, 108770. [CrossRef]
28. Albers, M.; Meysonnat, P.S.; Fernex, D.; Semaan, R.; Noack, B.R.; Schröder, W. Drag Reduction and Energy Saving by Spanwise Traveling Transversal Surface Waves for Flat Plate Flow. *Flow Turbul. Combust.* **2020**, *105*, 125–157. [CrossRef]
29. Fernex, D.; Semaan, R.; Albers, M.; Meysonnat, P.S.; Schröder, W.; Noack, B.R. Actuation response model from sparse data for wall turbulence drag reduction. *Phys. Rev. Fluids* **2020**, *5*, 073901. [CrossRef]
30. Lintermann, A.; Meinke, M.; Schröder, W. Zonal Flow Solver (ZFS): A highly efficient multi-physics simulation framework. *Int. J. Comput. Fluid Dyn.* **2020**, *34*, 458–485. [CrossRef]
31. Roidl, B.; Meinke, M.; Schröder, W. A reformulated synthetic turbulence generation method for a zonal RANS–LES method and its application to zero-pressure gradient boundary layers. *Int. J. Heat Fluid Flow* **2013**, *44*, 28–40. [CrossRef]
32. Whitfield, D. Three-dimensional unsteady Euler equations solution using flux vector splitting. In Proceedings of the 17th Fluid Dynamics, Plasma Dynamics, and Lasers Conference, Snowmass, CO, USA, 25–27 June 1984; p. 1552.
33. Wu, N.; Green, B.; Ben, X.; O'Banion, S. Deep Transformer Models for Time Series Forecasting: The Influenza Prevalence Case. *arXiv* **2020**, arXiv:2001.08317. [CrossRef]
34. Bode, M.; Gauding, M.; Lian, Z.; Denker, D.; Davidovic, M.; Kleinheinz, K.; Jitsev, J.; Pitsch, H. Using physics-informed enhanced super-resolution GANs for subfilter modeling in turbulent reactive flows. *Proc. Combust. Inst.* **2021**, *38*, 2617–2625. [CrossRef]
35. Kingma, D.P.; Ba, J. Adam: A Method for Stochastic Optimization. In Proceedings of the 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, 7–9 May 2015.
36. Keskar, N.S.; Mudigere, D.; Nocedal, J.; Smelyanskiy, M.; Tang, P.T.P. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. In Proceedings of the 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, 24–26 April 2017.
37. Aach, M.; Inanc, E.; Sarma, R.; Riedel, M.; Lintermann, A. Large scale performance analysis of distributed deep learning frameworks for convolutional neural networks. *J. Big Data* **2023**, *10*, 96. [CrossRef]
38. Sarma, R.; Inanc, E.; Aach, M.; Lintermann, A. Parallel and scalable AI in HPC systems for CFD applications and beyond. *Front. High Perform. Comput.* **2024** , *2*, 1444337.
39. Albers, M.; Meysonnat, P.S.; Fernex, D.; Semaan, R.; Noack, B.R.; Schröder, W.; Lintermann, A. CoE RAISE—Data for Actuated Turbulent Boundary Layer Flows. 2023. Available online: https://b2share.fz-juelich.de/records/5dbc8e35f21241d0889906136cf28d26 (accessed on 25 September 2024).
40. Baker, S.; Fang, X.; Shen, L.; Willman, C.; Fernandes, J.; Leach, F.; Davy, M. Dynamic Mode Decomposition for the Comparison of Engine In-Cylinder Flow Fields from Particle Image Velocimetry (PIV) and Reynolds-Averaged Navier–Stokes (RANS) Simulations. *Flow Turbul. Combust.* **2023**, *111*, 115–140. [CrossRef]
41. Demo, N.; Tezzele, M.; Rozza, G. PyDMD: Python Dynamic Mode Decomposition. *J. Open Source Softw.* **2018**, *3*, 530. [CrossRef]
42. Ichinaga, S.M.; Andreuzzi, F.; Demo, N.; Tezzele, M.; Lapo, K.; Rozza, G.; Brunton, S.L.; Kutz, J.N. PyDMD: A Python package for robust dynamic mode decomposition. *arXiv* **2024**, arXiv:2402.07463.
43. Jülich Supercomputing Centre. JURECA: Data Centric and Booster Modules implementing the Modular Supercomputing Architecture at Jülich Supercomputing Centre. *J. Large-Scale Res. Facil.* **2021**, *7*, A182. [CrossRef]