Synthetic turbulent inflow generator using machine learning

Kai Fukami,* Yusuke Nabae, Ken Kawai, and Koji Fukagata†
Department of Mechanical Engineering, Keio University, Yokohama 223-8522, Japan

(Received 13 July 2018; published 4 June 2019)

We propose a methodology for generating time-dependent turbulent inflow data with the aid of machine learning (ML), which has the possibility to replace conventional driver simulations or synthetic turbulent inflow generators. As for the ML model, we use an autoencoder-type convolutional neural network with a multilayer perceptron. For the test case, we study a fully developed turbulent channel flow at the friction Reynolds number of $\mathrm{Re}_\tau = 180$ for easiness of assessment. The ML models are trained using a time series of instantaneous velocity fields in a single cross section obtained by direct numerical simulation (DNS) so as to output the cross-sectional velocity field at a specified future time instant. From the *a priori* test in which the output from the trained ML model are recycled to the input, the spatiotemporal evolution of cross-sectional structure is found to be reasonably well reproduced by the proposed method. The turbulence statistics obtained in the *a priori* test are also, in general, in reasonable agreement with the DNS data, although some deviation in the flow rate was found. It is also found that the present machine-learned inflow generator is free from the spurious periodicity, unlike the conventional driver DNS in a periodic domain. As an *a posteriori* test, we perform DNS of inflow-outflow turbulent channel flow with the trained ML model used as a machine-learned turbulent inflow generator (MLTG) at the inlet. It is shown that the present MLTG can maintain the turbulent channel flow for a long time period sufficient to accumulate turbulent statistics, with much lower computational cost than the corresponding driver simulation. It is also demonstrated that we can obtain accurate turbulent statistics by properly correcting the deviation in the flow rate.

DOI: 10.1103/PhysRevFluids.4.064603

I. INTRODUCTION

To date, various types of inflow generators have been proposed for inflow-outflow simulations of turbulence. Physically speaking, the most straightforward method is to simulate the natural transition, starting from the laminar velocity profile with superimposed random fluctuations, as was examined by Rai and Moin [1] and also used in a recent direct numerical simulation (DNS) of a turbulent boundary layer by Wu and Moin [2]. Although this method is ideal, it requires high computational cost because of the necessity of a sufficiently long computational domain for laminar-turbulent transition. Adding relevant fluctuations to the mean velocity profile of already turbulent flow [3] is another option, often called a synthetic turbulent inflow generator. As discussed by Keating *et al.* [4], the synthesized velocity fluctuations should have spectral contents similar to those of actual turbulent flows; otherwise the added fluctuations dissipate quickly. For this purpose, several attempts have been made to generate random fluctuations having proper spatiotemporal

*Present Address: Department of Mechanical and Aerospace Engineering, University of California, Los Angeles, California 90095, USA.
†fukagata@mech.keio.ac.jp

correlations. Druault *et al.* [5] and Perret *et al.* [6] reconstructed inflow turbulence from measured experimental data using proper orthogonal decomposition and linear stochastic estimation. Klein *et al.* [7], di Mare *et al.* [8], and Hœpffner *et al.* [9] used digital filtering techniques to generate a correlated field out of random noise. Yet another, but seemingly most popular, method nowadays is to use an auxiliary (i.e., driver) turbulence simulation with a periodic computational domain. In order to take into account the spatial development in the periodic driver domain, Lund *et al.* [10] proposed a rescaling and recycling of velocity profiles and velocity fluctuations based on the law of the wall, which can be considered as a modified Spalart method [11]. As a result, they could successfully reproduce the development of turbulent boundary layer. Although such a driver-type inflow generator is more straightforward than the sophisticated synthetic turbulence generators introduced above, one of its major drawbacks is its additional computational cost. Another, and more crucial, drawback is the spurious periodicity issue arising from the streamwise periodicity in the driver simulation, as extensively discussed by Wu [12].

In recent years, machine learning has gathered increasing attention as a part of the boom in big data and artificial intelligence. Application of machine learning to fluid mechanics problems has a relatively long history. For instance, Lee *et al.* [13] devised a single-layer perceptron, which is a simplest type of neural network (NN), to learn the control input of opposition control [14] for turbulent friction drag reduction. Milano and Koumoutsakos [15] used a multilayer perceptron (MLP) [16] to estimate the flow field above the wall from the information on the wall; it is a surprising fact that they treated more than 26 000 inputs for a NN already about 20 years ago. Owing to the recent active development of machine learning libraries such as TensorFlow and Chainer, machine learning has now become a handier tool for fluid mechanics as well. Recently, Gamahara and Hattori [17] attempted regression of the subgrid scale (SGS) stress in large-eddy simulation using a three-layer perceptron and succeeded in reproducing SGS stresses similarly to those of the conventional Smagorinsky model [18,19]. Ling *et al.* [20] performed regression of the anisotropy tensor in Reynolds-averaged Navier-Stokes (RANS) simulations using a specially designed NN with an additional tensor input layer so as to account for the Galilean invariance and demonstrated a better prediction performance than a simple MLP. Huang *et al.* [21] attempted to predict using NN the difference between Reynolds stresses computed by DNS and RANS in a high Mach number flow.

Among different NN architectures, a convolutional neural network (CNN) [22] has been widely used in the field of image recognition. One of the features of a CNN is that it can naturally take into account the spatial structure of input data, in contrast to the traditional MLP having a fully connected NN architecture. This feature of a CNN is also advantageous when fluid mechanics problems are considered. Guo *et al.* [23] proposed a CNN implementation for real-time prediction of a nonuniform steady laminar flow. Although the result shows lower fidelity than the traditional computational fluid dynamics (CFD), a CNN is shown to be able to predict the velocity field faster than the CFD solver. Yilmaz and German [24] applied a CNN for prediction of the pressure coefficient on airfoils (note that this is originally a regression problem, but they converted it into a classification problem by making groups of pressure coefficients) and achieved more than 80% test accuracy. In addition, they mentioned that use of hybrid experimental and computational data as the training data sets has a possibility for better regression performance. Zhang *et al.* [25] proposed multiple CNN structures to predict the lift coefficient of airfoils with different shapes at different Mach numbers, Reynolds numbers, and angles of attack. One of the useful suggestions from their study is that using an artificial image as the input, which is a colored image corresponding to the input parameters, also improves the test accuracy.

From these contexts, it would be natural to consider utilizing machine learning to develop a turbulent inflow generator that may replace conventional driver simulations or synthetic turbulent inflow generators. In the present study, we propose such a turbulent inflow generator based on machine learning.

We propose an autoencoder [26] -type CNN combined with a MLP as a present machine-learning (ML) model. As schematically shown in Fig. 1, the CNN part of the present model works to compress the high-dimensional data of the cross-sectional velocity and pressure field into a

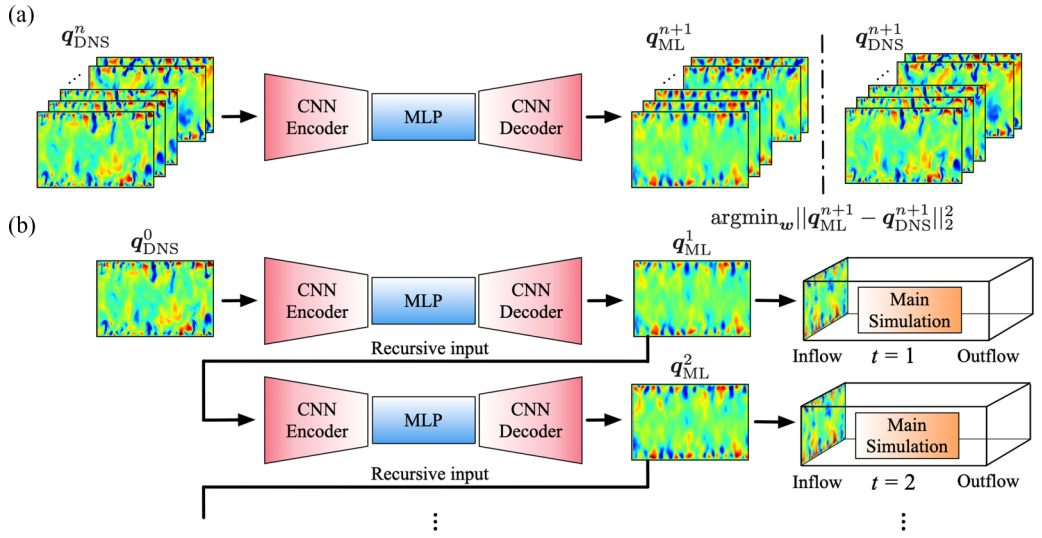

FIG. 1. Schematic diagram of the present machine learning and its use as an inflow generator. (a) Training stage; (b) *a priori* and *a posteriori* tests.

lower-dimensional latent space so that important spatial features of the flow are extracted, while the MLP part is used to regress their temporal relationship. In Sec. II the present ML model will be explained in greater detail.

The main idea of the present work is illustrated in Fig. 1(a). First, the ML model is trained using the velocity and pressure data in a cross section obtained by direct numerical simulation (DNS) so that the mean-squared error (MSE) between the cross-sectional velocity and pressure field at the next time step (i.e., output, $\boldsymbol{q}^{n+1}_{\mathrm{ML}}$), which is obtained as a response to that at the present time step (i.e., input, $\boldsymbol{q}^{n}_{\mathrm{DNS}}$), and that of DNS (i.e., the answer, $\boldsymbol{q}^{n+1}_{\mathrm{DNS}}$) is minimized. After the ML model is trained, we perform an *a priori* test by recycling the output of the ML model to the input, as indicated by the black arrows with a label "Recursive input" in Fig. 1(b), to investigate whether the spatiotemporal structure similar to turbulence is properly maintained within the ML model. Note that, at this stage, the DNS data are fed into the ML model only once for the initialization [i.e., $\boldsymbol{q}^{0}_{\mathrm{DNS}}$ in Fig. 1(b)]. Finally, an *a posteriori* test is conducted by inflow-outflow DNS with time-dependent inflow conditions computed using this machine-learned turbulent inflow generator (MLTG).

II. PROBLEM FORMULATION AND TRAINING METHODS

A. Training procedure

The training datasets are generated using DNS. For ease of assessment, we consider a fully developed incompressible turbulent channel flow as the test case to examine the feasibility of the present approach, although an efficient inflow generator may be appreciated more in simulations of external flows such as spatially developing boundary layers and flows around a body.

The governing equations are the incompressible Navier-Stokes equations,

$$\nabla \cdot \boldsymbol{u} = 0, \tag{1}$$

$$\frac{\partial \boldsymbol{u}}{\partial t} = -\nabla \cdot (\boldsymbol{u}\boldsymbol{u}) - \nabla p + \frac{1}{\mathrm{Re}_\tau}\nabla^2 \boldsymbol{u}, \tag{2}$$

where $\boldsymbol{u} = [u\, v\, w]^{\mathrm{T}}$ represents the velocity with u, v, and w being the streamwise (x), wall-normal (y), and spanwise (z) components; p is the pressure, t is the time, and $\mathrm{Re}_\tau = u_\tau \delta/\nu$ is the friction

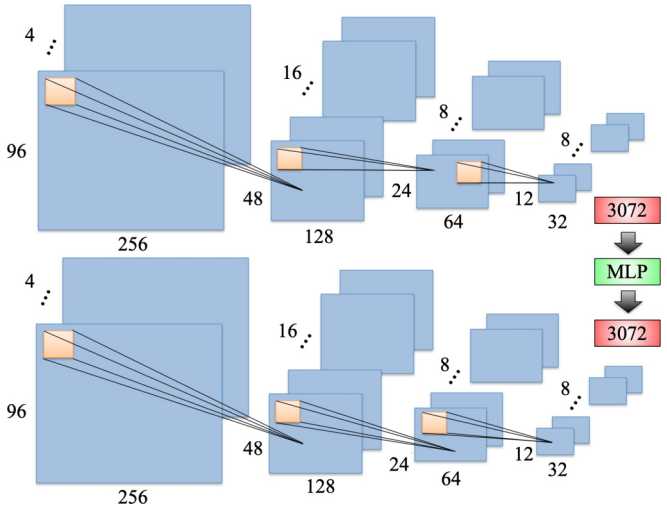

FIG. 2. Schematic structure of the ML model using the autoencoder-type convolutional neural network with multilayer perceptron (e.g., Case 1).

Reynolds number. The quantities are made dimensionless using the channel half-width δ and the friction velocity u_τ.

The DNS is performed using the finite difference code of Fukagata *et al.* [27], which has been validated by comparison with spectral DNS data of Moser *et al.* [28]. The size of the computational domain and the number of grid points are $(L_x, L_y, L_z) = (4\pi\delta, 2\delta, 2\pi\delta)$ and $(N_x, N_y, N_z) = (256, 96, 256)$, respectively. The grid is uniform in the x and z directions, while nonuniform in the y direction. A no-slip boundary condition is imposed on the walls, and the periodic boundary condition is applied in the x and z directions. The DNS is performed under a constant pressure condition at $\mathrm{Re}_\tau = 180$.

Time series of velocity and pressure field in a single y-z cross section computed by this DNS are used as the training data for the ML model. Since the raw streamwise velocity, u, has a strongly skewed distribution due to its mean velocity component, and such an ill-formed distribution of training data is known to deteriorate prediction using a neural network [29], we use the fluctuations, $\boldsymbol{u}' = [u'\ v'\ w']^{\mathrm{T}}$ and p' as the input and output vector, $\boldsymbol{q} = [u'\ v'\ w'\ p']$. The regression using the ML model can be expressed as

$$\boldsymbol{q}_{\mathrm{ML}}^{n+1} = \mathcal{F}\left(\boldsymbol{q}_{\mathrm{DNS}}^{n}; \boldsymbol{W}\right) \approx \boldsymbol{q}_{\mathrm{DNS}}^{n+1}, \tag{3}$$

where $\boldsymbol{q}_{\mathrm{ML}}^{n+1}$ denotes the cross-sectional field at the next time step computed by the ML model, $\boldsymbol{q}_{\mathrm{DNS}}^{n}$ represents the DNS data at the present time step that are fed as the input to the ML model, and $\boldsymbol{q}_{\mathrm{DNS}}^{n+1}$ is the answer that should be reproduced by the ML model. The nonlinear mapping by the ML model is denoted by $\mathcal{F}(\cdot)$, and $\boldsymbol{W}$ denotes the weights in the ML model that are optimized by learning so as to minimize the loss function, i.e., the MSE between $\boldsymbol{q}_{\mathrm{ML}}^{n+1}$ and $\boldsymbol{q}_{\mathrm{DNS}}^{n+1}$. The time interval between time steps n and $n+1$ in ML is $\Delta t^+ = 1.26$, which corresponds to 10 time steps in the present DNS.

The code for machine learning has been written in-house by utilizing TENSORFLOW 1.2.0 and KERAS 2.0.5 libraries on PYTHON 3.6.

B. Machine-learned turbulence generator

As shown in Fig. 2, the basic network structure of the present ML model is similar to the standard CNN autoencoder used for image recognition, as can be found, e.g., in Keras tutorial [30]. First, instantaneous y-z cross-sectional data, $\boldsymbol{q}_{\mathrm{DNS}}^{n}$, are fed into the network. All input data are standardized

TABLE I. Detailed structure of the machine-learned turbulence generator (e.g., Case 1)

Network	Data size	Activation function
Input	(96,256,4)	-
First Conv2D	(96,256,16)	tanh
Second Conv2D	(96,256,16)	tanh
First AveragePooling 2D	(48,128,16)	-
Third Conv2D	(48,128,8)	tanh
Fourth Conv2D	(48,128,8)	tanh
Second AveragePooling 2D	(24,64,8)	-
Fifth Conv2D	(24,64,8)	tanh
Sixth Conv2D	(24,64,8)	tanh
Third AveragePooling 2D	(12,32,8)	-
First Reshape	(1,3072)	-
First MLP	(3072)	tanh
Second MLP	(3072)	tanh
Second Reshape	(12,32,8)	-
Seventh Conv2D	(12,32,8)	tanh
Eighth Conv2D	(12,32,8)	tanh
First Upsampling 2D	(24,64,8)	-
Ninth Conv2D	(24,64,8)	tanh
10th Conv2D	(24,64,8)	tanh
Second Upsampling 2D	(24,64,8)	-
11th Conv2D	(48,128,16)	tanh
12th Conv2D	(48,128,16)	tanh
Third Upsampling 2D	(96,256,16)	-
Output/13th Conv2D	(96,256,4)	-

so that the mean value is zero and the standard deviation is unity, because it is generally known that if the mean value of input data deviates from zero, the weight update will be affected and the learning speed becomes slower [31]. Since the input data consist of four primitive variables ($\boldsymbol{u}'$ and p') in a single y-z cross section, the total number of inputs in the present study is $96 \times 256 \times 4 = 98\,304$ per instant, i.e., the product of the number of computational points in a y-z section and the number of variables. This high-dimensionalized input data are first compressed by a sequence of convolution layers. Then low-dimensionalized data are passed to fully connected MLP layers, in which the relationship between the low-dimensionalized features at two consecutive time instants are regressed. Finally, the data corresponding to the low-dimensionalized field at the next time step are expanded to their original dimension by the deconvolution layers. The function $\mathcal{F}$ in Eq. (3) is expressed as

$$\mathcal{F}\left(\boldsymbol{q}^n_{\mathrm{DNS}}\right) = \mathcal{F}_{\mathrm{dec}}\left(\mathcal{F}_{\mathrm{MLP}}\left(\mathcal{F}_{\mathrm{enc}}\left(\boldsymbol{q}^n_{\mathrm{DNS}}\right)\right)\right), \tag{4}$$

where $\mathcal{F}_{\mathrm{enc}}$, $\mathcal{F}_{\mathrm{MLP}}$, and $\mathcal{F}_{\mathrm{dec}}$ denote the CNN encoder, MLP layer, and CNN decoder, respectively.

More detailed structure is shown in Table I. Conv2D is a convolution layer, in which the data are convolved with filters. Pooling has the role to compress the data. Two widely used pooling techniques are MaxPooling and AveragePooling: the former selects the maximum value, while the latter selects the average value. We have attempted both pooling models and confirmed that AveragePooling model shows better accuracy than the MaxPooling model in both training and test processes; therefore, we adopt AveragePooling in the present study. Upsampling is an operation for a dimension extension, in which the value is copied to the extended dimensions. In all layers, the filter size is set at 3×3, and the pooling size is 2×2 for both compression and extension. As for the activation function, we have tested three different activation functions: the hyperbolic tangent

TABLE II. Parameters of the ML models, the number of epochs before early stopping, and the resultant MSEs.

Case	No. of MLP layers	Structure of MLP layers	No. of epochs	MSE
Case 1	2	3072–3072	148	0.0289
Case 2	2	192–192	104	0.6981
Case 3	3	3072–3072–3072	43	0.6066
Case 4	3	3072–768–3072	73	0.1259

(tanh), the rectified linear unit (ReLU), and the sigmoid function. From this preliminary test, it has turned out that the hyperbolic tangent (tanh) gives the best result in both training and test processes in the present study. The Adam (adaptive moment estimation) optimizer [32] is used to optimize the weighting of the ML model.

In terms of the computational cost, it is preferable to use as small a number of MLP layers and latent data size as possible. Simplifying the MLP layer is preferable also for a physical interpretation of the latent space, if possible. However, oversimplification of the network structure may lead to an insufficient ability to express the essential dynamics. Therefore, we have examined four cases of ML models by varying the numbers of MLP layers and the latent data size fed to the MLP layer, as shown in Table II, to see their influence on the results. Case 1 is the base case with two MLP layers whose latent data size is 3072; in fact, this was the largest size we could handle under our computer environment. In Case 2, a smaller latent data size (i.e., a higher compression ratio) is considered by adding a pair of convolution and deconvolution layers around the MLP layers to the network shown in Fig. 2 and Table I. Case 3 and Case 4 include an additional hidden layer with the size of 3072 and 768, respectively.

In all cases, 10 000 pairs of snapshots spanning 12 600 wall unit time computed by DNS are used. Among them, 70% is used as the training data, and 30% is used as the validation data. Note that the pairs of DNS data at two consecutive time instants are fed into the network in a ramdom sequence. Namely, what the present ML model learns is not the long time series of DNS data themselves but the Navier-Stokes equation distretized with a relatively large time step of $\Delta t^+ \sim 1$. Overfitting, where the error for the training data set is lower than that for the validation data set, is avoided by employing the early stopping criterion [33] in this study. A series of continuous 20 epochs is used for the criterion of early stopping. The number of epochs trained until this early stopping and the resultant MSE values are shown in Table I.

As can be noticed from the learning curves presented in Fig. 3 and the resultant MSE in Table I, the learning is most successful in Case 1. In Case 2 with higher compression ratio, the resultant MSE is much larger than that in Case 1. When an extra hidden layer is added (i.e., Cases 3 and 4), the network seems to suffer from overfitting at smaller number of epochs due to the higher degree of freedom (i.e., many parameters) in the latent space.

III. RESULTS AND DISCUSSION

A. *A priori* test: Recycling within ML model

As an *a priori* test, we recycle the output of the trained ML model to its input for multiple times, with an initial condition taken from a single snapshot of DNS data, namely,

$$\boldsymbol{q}_{\mathrm{ML}}^{n+1} = \mathcal{F}\left(\boldsymbol{q}_{\mathrm{ML}}^{n}; \boldsymbol{W}\right), \tag{5}$$

with the initial condition,

$$\boldsymbol{q}_{\mathrm{ML}}^{0} = \boldsymbol{q}_{\mathrm{DNS}}^{0}. \tag{6}$$

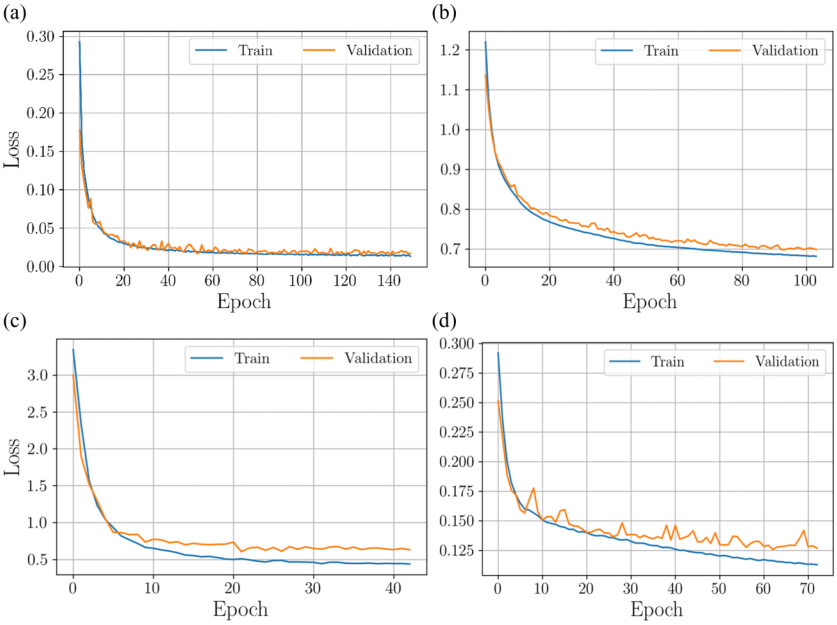

FIG. 3. Learning curve: (a) Case 1, (b) Case 2, (c) Case 3, (d) Case 4.

The statistics presented below are those accumulated for 10 000 time steps (i.e., 12 600 wall unit time) of this recycling. Note that we intend to reduce the computation time as compared to traditional driver-type turbulent inflow generators. Therefore, the primary purpose of this *a priori* test is to see whether the MLTG can generate self-sustaining inflow turbulence without feeding additional DNS data.

Figure 4 shows the "Reynolds stress" components computed using the output "velocity fluctuations," $R_{ij} = \overline{u_i'^+ u_j'^+}$. We express here the "Reynolds stress" and "velocity fluctuations" with quotations because the zero-mean properties of u_i' are lost in the present ML model, as discussed later. Case 1 shows reasonable agreement with the DNS data for all "Reynolds stress" components. Case 2 underestimates the turbulence statistics, especially regarding the components concerning u and w, due to overcompression of the latent vector fed to the MLP layer. These observations suggest that the data size of the latent space is an important parameter to maintain the physical features. In Cases 3 and 4, the computed statistics are qualitatively similar to the reference DNS data; however, the accuracy is poor for R_{11} and R_{33} in Case 3 and R_{22} in Case 4.

Figure 5 shows the mean profiles of "velocity fluctuations," $\overline{u'^+}$, $\overline{v'^+}$, and $\overline{w'^+}$, where the overbar denotes the average in the spanwise direction and in time. Ideally, these quantities should be zero, as indicated by the DNS data shown together. However, the distributions obtained by the ML model show substantial nonzero values due to the lack of zero-mean constraint in the learning process.

The bulk-mean velocities computed from these profiles, $\langle\overline{u'^+}\rangle$, $\langle\overline{v'^+}\rangle$, and $\langle\overline{w'^+}\rangle$, where the brackets denotes the average in wall-normal direction, presented in Table III reveal that the error amounts to 0.5% of the bulk-mean velocity of the original flow. In order to fix this problem, we define the corrected fluctuations as

$$u_i'' = u_i' - \overline{u_i'} \tag{7}$$

and

$$p'' = p' - \overline{p'}. \tag{8}$$

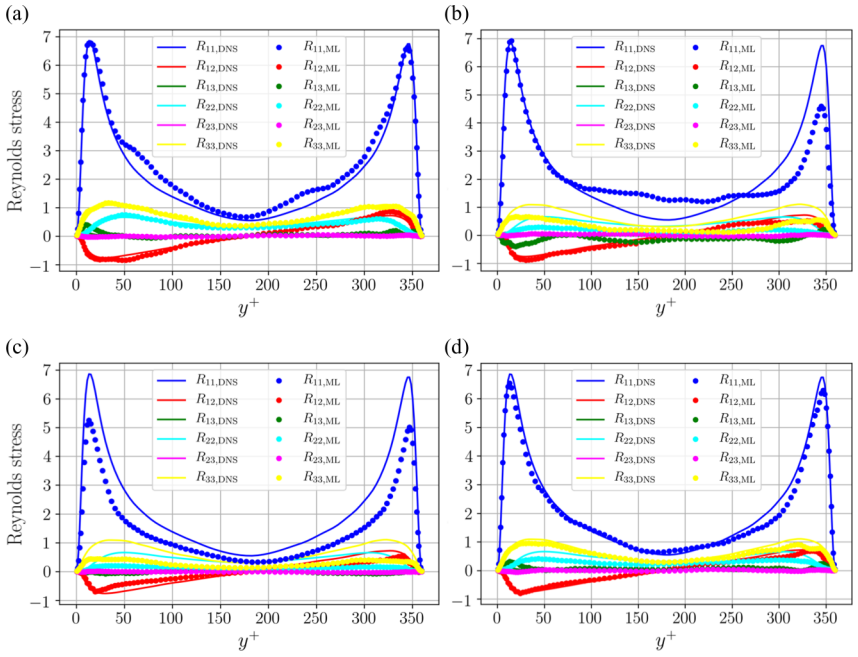

FIG. 4. "Reynolds stresses" computed based on the raw output, $R_{ij} = \overline{u_i'^+ u_j'^+}$: (a) Case 1, (b) Case 2, (c) Case 3, (d) Case 4.

The turbulence statistics based on these corrected velocity fluctuations are shown in Fig. 6. Case 1 shows reasonable agreement with the DNS data, while Cases 2–4 show poorer results, especially for v and w components.

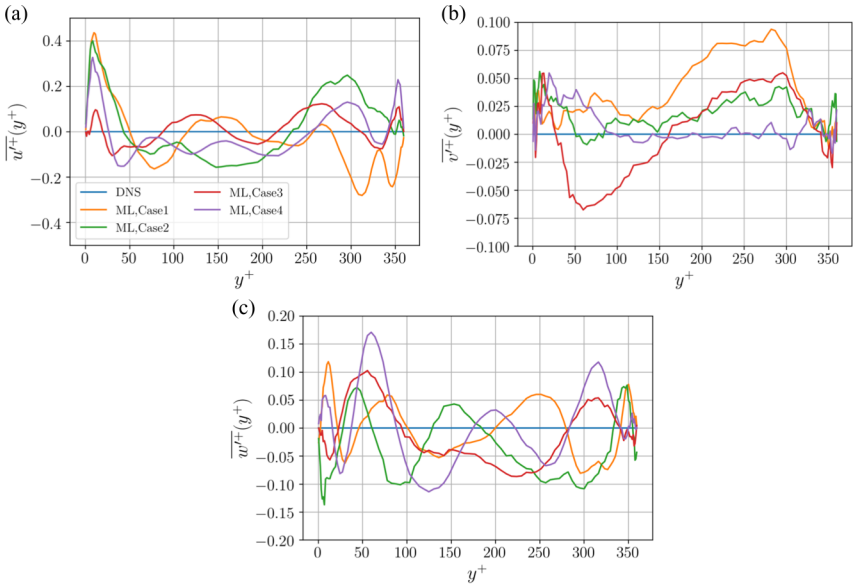

FIG. 5. Mean profiles of "velocity fluctuations": (a) $\overline{u'}$, (b) $\overline{v'}$, (c) $\overline{w'}$.

TABLE III. Bulk-mean value of "velocity fluctuations."

	$\langle\overline{u'^+}\rangle$	$\langle\overline{v'^+}\rangle$	$\langle\overline{w'^+}\rangle$
DNS	-9.67×10^{-17}	3.63×10^{-19}	1.51×10^{-19}
Case 1	-6.59×10^{-3}	3.32×10^{-2}	5.10×10^{-3}
Case 2	5.80×10^{-2}	2.02×10^{-2}	-3.45×10^{-2}
Case 3	1.29×10^{-2}	2.41×10^{-2}	-7.25×10^{-2}
Case 4	2.40×10^{-2}	8.27×10^{-2}	-9.03×10^{-3}

To examine the accuracy in the spatial structure reproduced by the MLTG in greater detail, the spanwise energy spectrum of the streamwise velocity at $y^+ = 13.2$ is compared in Fig. 7. The ML models show reasonable agreement with the DNS data, although some attenuations are observed in higher wave-number range. With a higher compression ratio (i.e., Case 2), the higher wave-number components are damped more, as observed in Fig. 7(b).

The cross-sectional structure of the streamwise velocity fluctuations (u') after 200 time steps (i.e., about 250 wall unit time) of recycling within MLTG are shown in Fig. 8. The

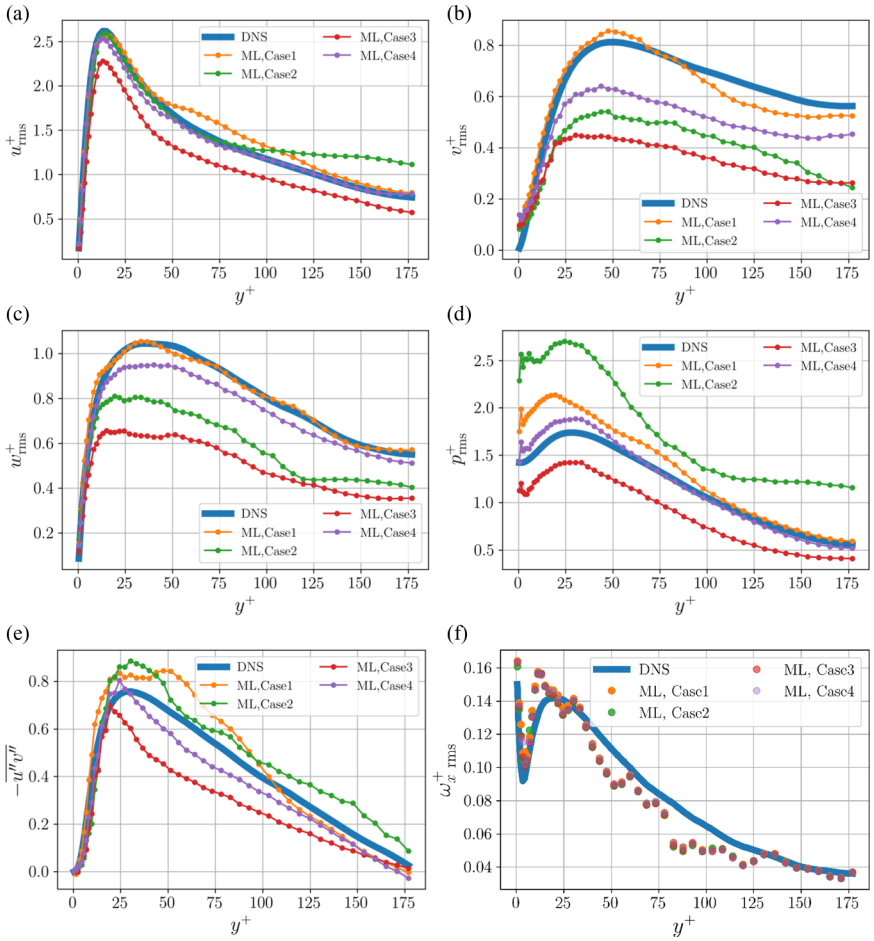

FIG. 6. Statistics based on the corrected fluctuations: (a) $u^+_{\rm rms}$, (b) $v^+_{\rm rms}$, (c) $w^+_{\rm rms}$, (d) $p^+_{\rm rms}$, (e) $-\overline{u''^+v''^+}$, (f) $\omega^+_{x\,\rm rms}$.

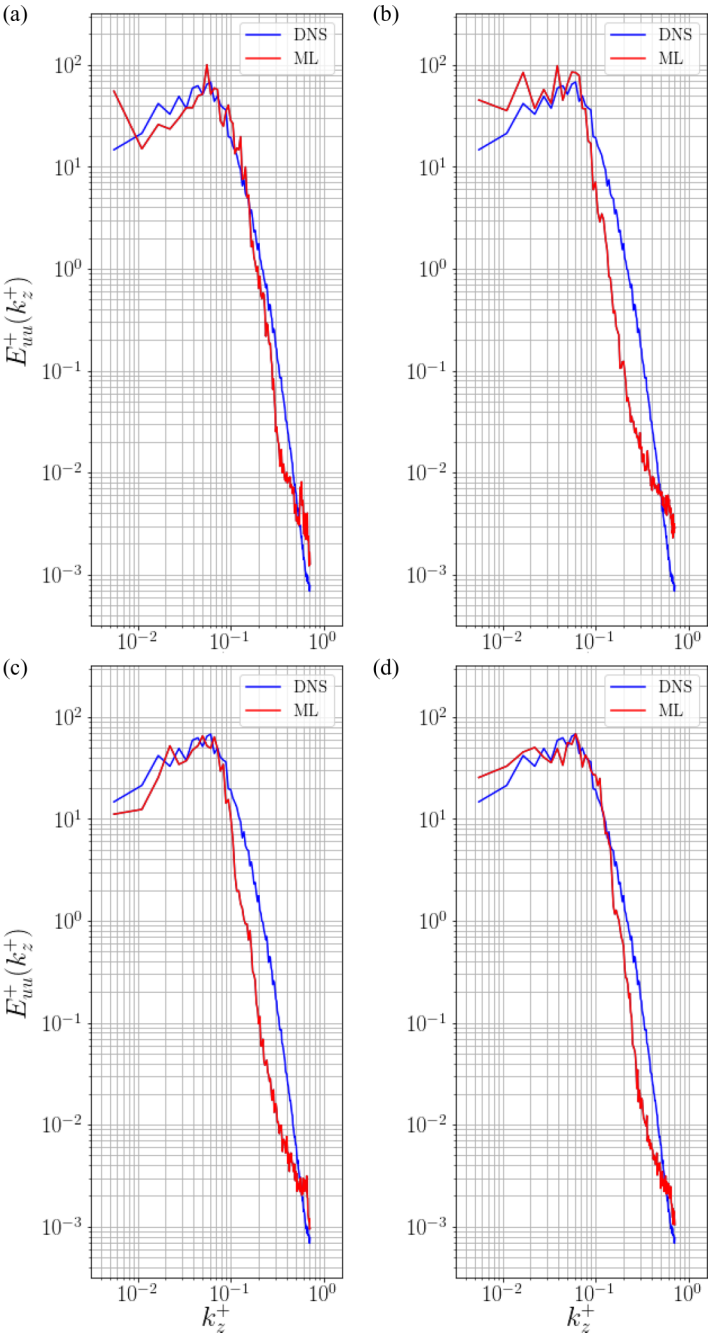

FIG. 7. Spanwise energy spectrum: (a) Case 1, (b) Case 2, (c) Case 3, (d) Case 4.

temporal evolution is best illustrated by an animation [34]. We can confirm the self-sustaining spatiotemporal evolution similar to that of DNS. Consistent with the statistics presented above, the structure in Case 1 is observed to be most similar to that of DNS among the present four cases.

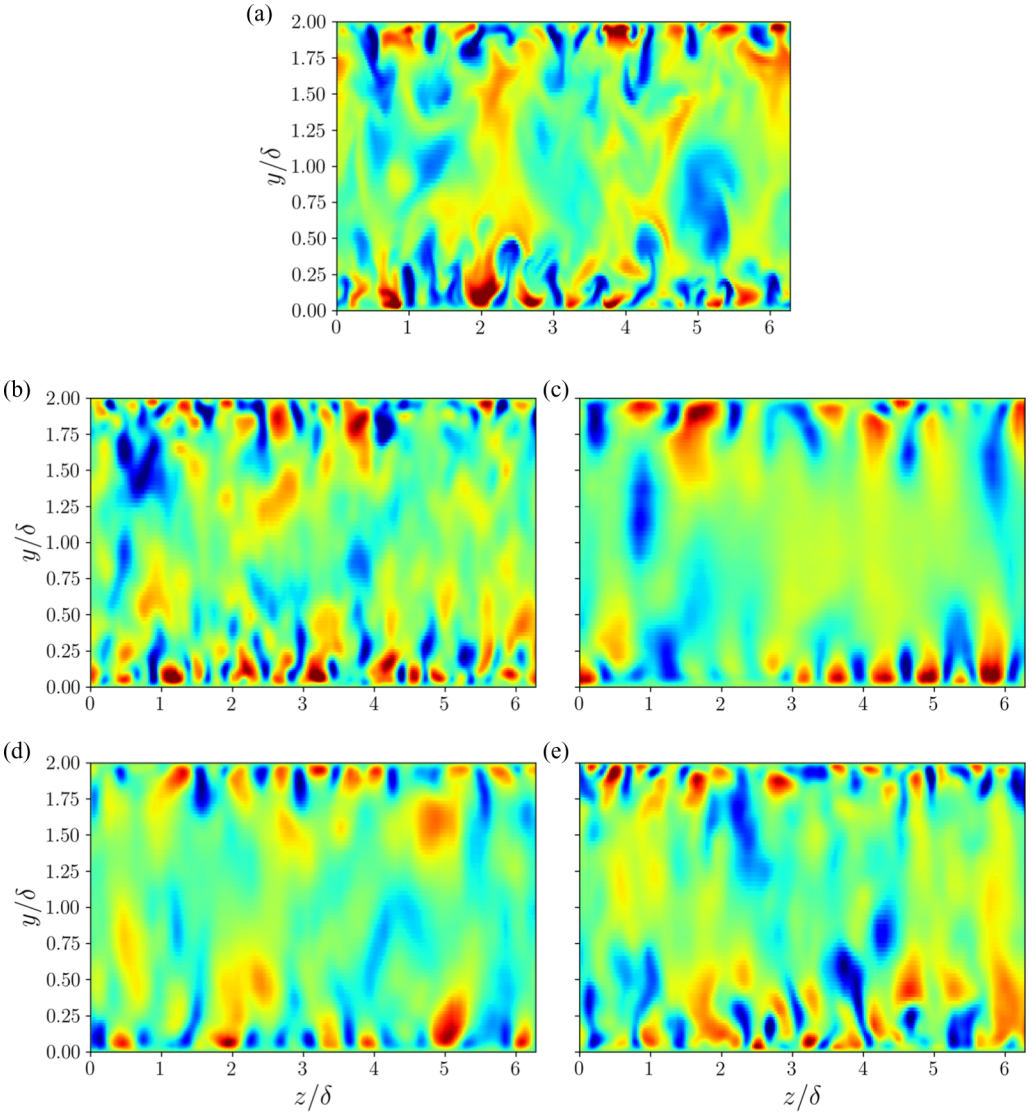

FIG. 8. Cross-sectional contour of streamwise velocity fluctuations u' after recycling for 200 time steps (i.e., about 250 wall unit time): (a) DNS, (b) Case 1, (c) Case 2, (d) Case 3, (e) Case 4. Animations are available at Ref. [34].

To see whether or not the present MLTG also suffers from a spurious periodicity issue, we have computed the temporal spectra and two-point correlations of the streamwise velocity component at two different wall-normal locations: near the wall ($y^+ = 13.2$) and the channel center ($y^+ = 177.0$), as shown in Fig. 9. In the case of the driver DNS with a periodic computational domain, we can clearly observe a spurious periodicity with a period corresponding to the length of the computational domain divided by the advection velocity. In contrast, with the present MLTG, such a spurious periodicity is not observed. This is probably because what the present ML model has learned is not the time sequence of the input data themselves but the *most probable* nonlinear spatiotemporal relationship between two consecutive time instants. In other words, the present MLTG is considered to work as a surrogate for the time-discretized nonlinear Navier–Stokes system.

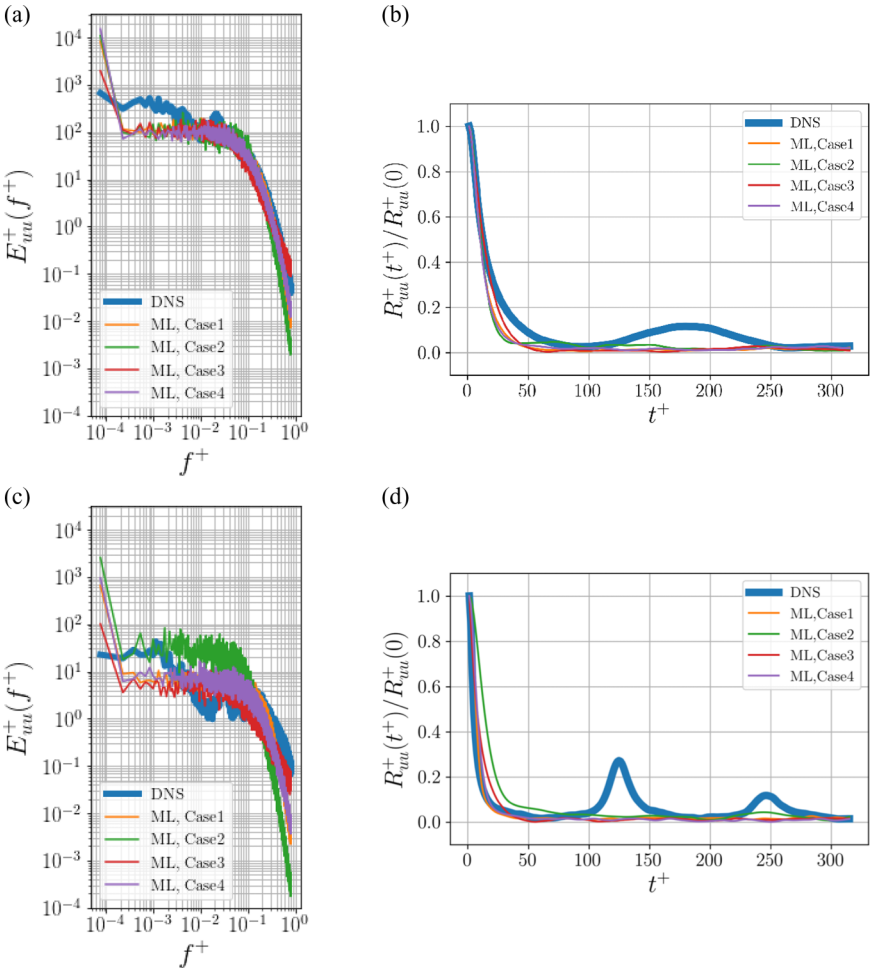

FIG. 9. Temporal statistics in the *a priori* test: (a) temporal spectrum at $y^+ = 13.2$; (b) temporal two-point correlation coefficient at $y^+ = 13.2$; (c) temporal spectrum at $y^+ = 177.0$; (d) temporal two-point correlation coefficient at $y^+ = 177.0$.

The integral timescales, $\mathcal{T}_u^+$, computed from these temporal two-point correlations,

$$\mathcal{T}_u^+ = \int_0^{T^+} \frac{R_{uu}(t^+)}{R_{uu}(0)}\, dt^+, \tag{9}$$

are presented in Table IV. The integration time, T^+, which should be infinity by definition, is $T^+ = 25\,200$ in the present calculation. It can be noticed that the integral timescale in Case 1 is substantially underestimated as compared to that of DNS. This suggests that, although the spatiotemporal structure is qualitatively well reproduced by the present ML models, the network structure and the parameters need further improvement for more quantitative agreement. Note that the value near the wall ($y^+ = 13.2$) in the present DNS, $\mathcal{T}_u^+ \simeq 24$, is also overestimated as compared to that reported in the literature, $\mathcal{T}_u^+ \simeq 20$ at $y^+ = 10$ [35]; this is obviously due to the insufficient steamwise length and the spurious periodicity thereby.

TABLE IV. The integral timescale, $\mathcal{T}_u^+$.

Location	$y^+ = 13.2$	$y^+ = 177.0$
Periodic DNS ($L_x = 4\pi\delta$)	24.1	15.9
MLTG Case 1	16.7	12.3
MLTG Case 2	19.7	23.3
MLTG Case 3	19.0	13.1
MLTG Case 4	17.8	11.8

B. *A posteriori* test: Inflow-outflow DNS using machine-learned inflow generators

As an *a posteriori* test, we assess whether the MLTGs can actually be used in turbulent flow simulations with an inflow condition. Following the results of the *a priori* test, we use the uncorrected data obtained in Case 1, u_i' (termed Case 1), and the corrected data, u_i'' (Case 1′), to provide the time-dependent inflow condition for DNS of turbulent channel flow with an inflow-outflow condition and compare with the results computed using the traditional driver DNS (i.e., an additional periodic DNS as a driver). The streamwise length of the computational domain for the inflow-outflow DNS is $L_x = 4\pi\delta$. The convective outflow condition is imposed at the outlet. The cross-sectional velocity field data by MLTG is generated every 10 time step of DNS, $\Delta t^+ = 1.26$. The inflow data at intermediate time instants are given by using a linear interpolation.

The spatiotemporal development of the peak value in the RMS velocity fluctuations normalized by the value of DNS, $u'_{i,\mathrm{MLTG}}/u'_{i,\mathrm{DNS}}$, and the spatiotemporal development of local friction Reynolds number, Re_τ, computed by DNS with MLTG are shown in Fig. 10. Hereafter, the velocity fluctuations in Case 1′ are also denoted by a single prime for notational simplicity. The horizontal axis is the wall unit time t^+, and the vertical axis represents the streamwise length from the inlet. Note that the peak values of RMS velocities are computed in 16 subsections divided in the streamwise direction. The computations used the present MLTG are continued at $t^+ = 10\,000$, which is considered long enough to accumulate turbulent statistics, and $u'_{i,\mathrm{MLTG}}/u'_{i,\mathrm{DNS}}$ and Re_τ are maintained nearly constant. Although the velocity fluctuations are slightly damped near the inlet due the errors of the MLTG, the flow recovers to have the correct statistics after $x/\delta \simeq 2$.

Turbulence statistics computed in the inflow-outflow DNS with the MLTG are shown in Fig. 11. The turbulence statistics here are accumulated in the entire computational domain of $L_x = 4\pi\delta$ and normalized in the wall unit of the original flow, $\mathrm{Re}_\tau = 180$. The statistics obtained in the DNS in a periodic domain of $L_x = 4\pi\delta$ are also shown for comparison. Mean velocity profile, RMS velocity fluctuations, RMS vorticity fluctuations, Reynolds shear stress, and streamwise and spanwise spectra of streamwise velocity are all in reasonable agreement with the periodic DNS, which confirms that the present ML model properly works as the inflow generator. In particular, Case 1′ based on the corrected fluctuations outperforms Case 1. The deviation in Case 1 is attributed to the increased flow rate due to the nonzero mean component as observed in the *a priori* test.

The present result demonstrates that the turbulence statistics can be reproduced in an inflow-outflow DNS with an MLTG as far as the MLTG reproduces the spatiotemporal characteristics of inflow turbulence reasonably well, although not perfect. Even if a small amount of error is contained at the inlet, the flow recovers in the DNS domain to have the correct statistics. More extensive work, however, is needed to clarify how much and what kinds of errors are allowed at the inlet.

At last, the computational time required for generating the turbulent inflow data for one time step is compared shown in Table V. When we use a periodic DNS as a driver simulation, at least $L_x = 2\pi\delta$ should be required to obtain reasonable statistics. Therefore, comparison is made between the present model and a periodic DNS with $L_x = 2\pi\delta$. Although the concrete value of computational time highly depends on the environment such as the machine, complier, and library used and the way of coding, the MLTG is apparently faster than the driver-type turbulence generator. Under our environment, the computational speed of the ML model for generating one cross-sectional velocity

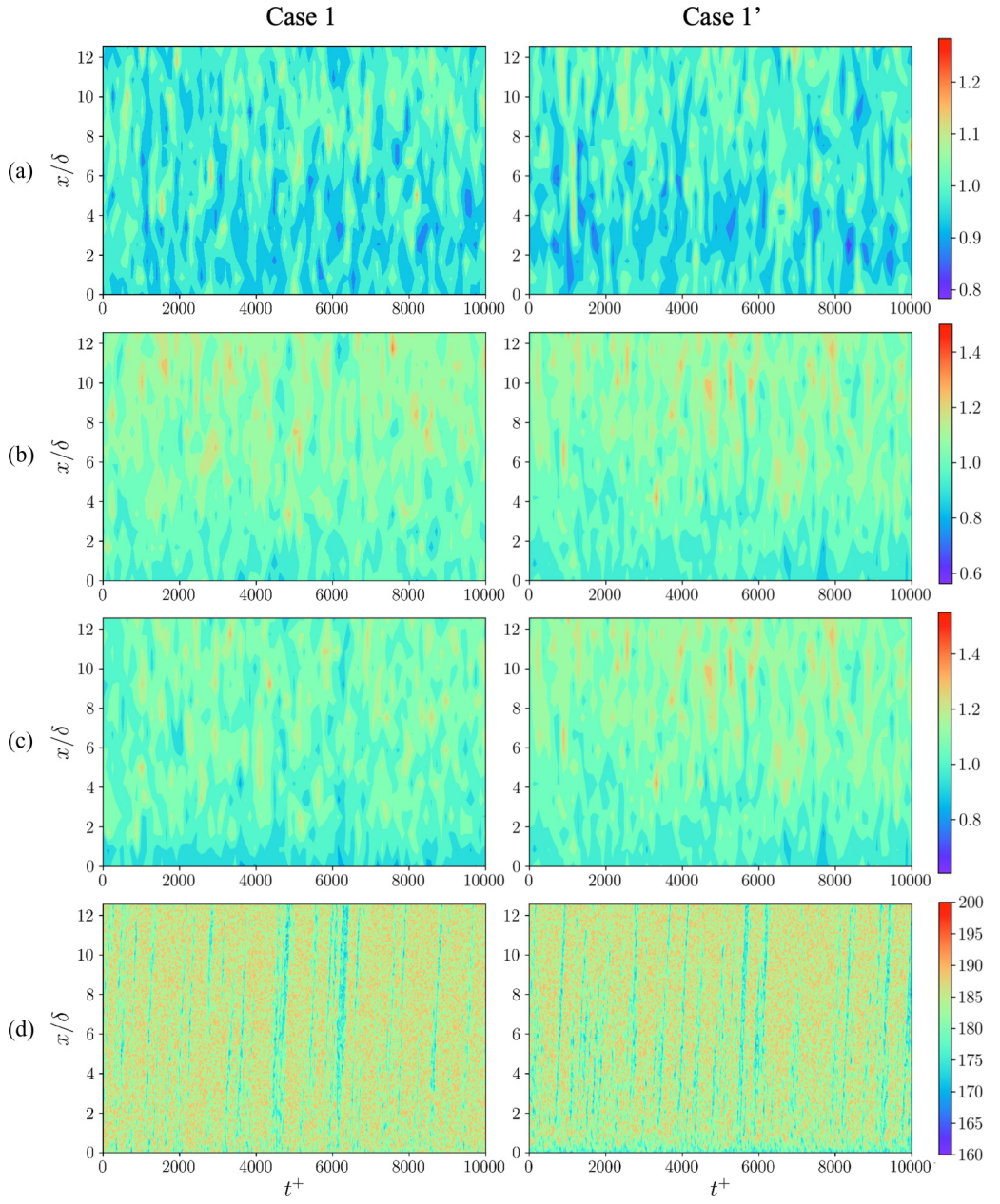

FIG. 10. Spatiotemporal development of peak RMS velocity: (a) $u'_{\mathrm{MLTG}}/u'_{\mathrm{DNS}}$; (b) $v'_{\mathrm{MLTG}}/v'_{\mathrm{DNS}}$; (c) $w'_{\mathrm{MLTG}}/w'_{\mathrm{DNS}}$; (d) Re_τ.

field is about 180 times faster on CPU (single core Intel Xeon E5-2680v4, 2.4 GHz) and about 580 times faster when a GPU (NVIDIA Tesla K40) is used than the driver DNS run on the same CPU. In addition, considering the fact that in the present ML case computes a cross-sectional velocity field every 10 time steps of DNS (since MLTG is not restricted by the Courant number), the actual speed-up rate is 10 times the values above, namely, 1800 times and 5800 times faster when CPU and GPU are used, respectively.

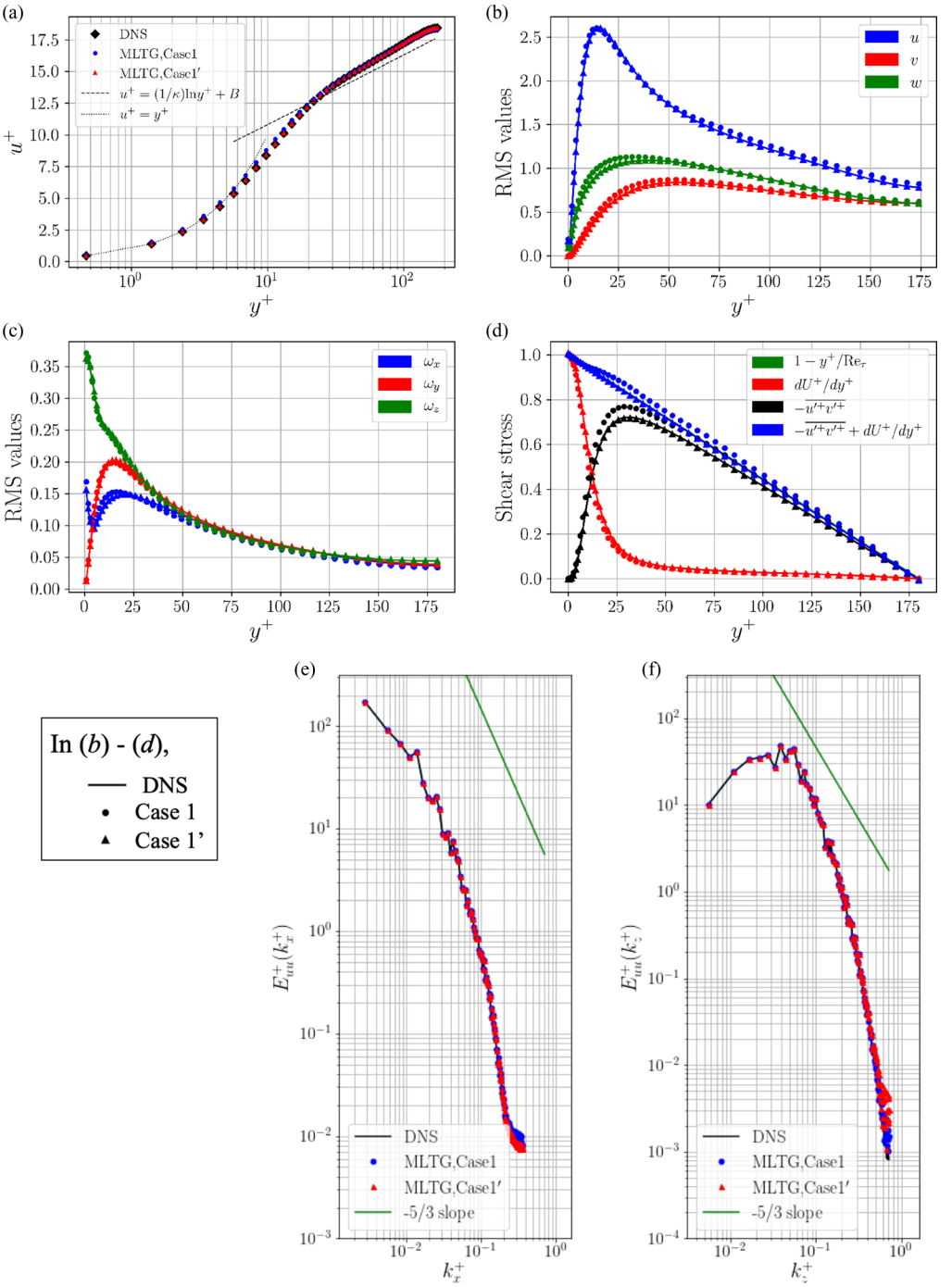

FIG. 11. Turbulence statistics in *a posteriori* test using MLTG: (a) mean velocity profile; (b) RMS of u_i'; (c) RMS of ω_i'; (d) shear stress balance; (e) streamwise energy spectrum of u'; (f) spanwise energy spectrum of u'.

TABLE V. Comparison of computation time in *a posteriori* test

Generator type	Time (s)	Ratio versus MLTG, Case 1 (CPU)	Ratio versus MLTG, Case 1 (GPU)
Driver DNS ($L_x = 2\pi\delta$)	2.39	181	582
MLTG, Case 1 (CPU)	1.32×10^{-2}	1.00	3.21
MLTG, Case 1 (GPU)	4.11×10^{-3}	0.311	1.00

In sum, it can be concluded that the present type of MLTG can be used also in practical simulations in terms of self-sustainability of turbulent structure, accuracy in reproduced turbulent statistics, and low computational cost.

IV. CONCLUSIONS

In this paper, we proposed a machine-learned turbulence generator (MLTG) using an autoencoder-type convolutional neural network (CNN) combined with a multilayer perceptron (MLP). For the test case, a turbulent channel flow at the friction Reynolds number of $\mathrm{Re}_\tau = 180$ is considered as a first step.

The machine-learning (ML) models were trained using a series of instantaneous velocity fields in a single cross section obtained by direct numerical simulation (DNS) so as to output the cross-sectional velocity field at a specified future time instant. In the *a priori* test, the present MLTG was found to accurately reproduce not only the turbulence statistics but also the spatiotemporal development of cross-sectional structure, although some deviation in the flow rate was found. Moreover, unlike the conventional driver DNS using a periodic domain, the present MLTG is found to be free from the spurious periodicity. As an *a posteriori* test, we performed DNS of inflow-outflow turbulent channel flow with the trained MLTG as the time-dependent inflow condition. The MLTG was able to maintain the turbulent channel flow in a long time period up to 10 000 wall unit time, which is sufficient to accumulate turbulent statistics, with much lower computational cost than the conventional driver simulation.

The present results suggest that MLTG is an attractive alternative to the conventional methods. Although there is a computational overhead for training, MLTG should be useful in the cases where many simulations are performed under statistically the same inflow condition but different downstream conditions due to, e.g., control, roughness, and obstacles. Extension of the proposed methodology to other types of flows, such as a spatially developing boundary layer and flows around a body, is straightforward, but the accuracy should be assessed for each problem.

From the observation of the *a priori* test, the results are found to be sensitive against the parameters of machine learning, including the number of layers, units, and so on. Although we have obtained reasonable results in the present study, more extensive study should be made to find better network structures giving higher accuracy. For instance, the long short-term memory (LSTM) [36] proposed to deal with the complicated time series of data can be considered to increase the accuracy in temporal characteristics; in fact, the usefulness of LSTM has recently been demonstrated by Vlachas *et al.* [37] for a number of dynamical systems. In addition, an architecture independent of the shape and size of input data will also be needed. Also, the structure of a CNN can be modified so as to learn the different spatial scales more accurately, which is one of the ongoing studies in our group [38].

Despite the merits mentioned above, the major drawback of the present method in contrast to the conventional synthetic turbulence generators is that not only the statistics but also spatiotemporal data of the target flow are still required to train the network. The ultimate goal may be to construct a similar network which requires lower order information such as spatiotemporal correlations only. However, we believe that the present study will serve as a good starting point toward this direction; the remaining issues will be tackled in the future.

ACKNOWLEDGMENTS

The authors are grateful to Dr. S. Obi, Dr. K. Ando, Dr. Y. Aoki, and K. Endo (Keio University), Dr. M. Yamamoto and Dr. T. Tsukahara (Tokyo University of Science), Dr. K. Iwamoto (Tokyo University of Agriculture and Technology), Dr. Y. Hasegawa (University of Tokyo), Dr. N. Fukushima (Tokai University), Dr. H. Mamori (University of Electro-Communications), and Dr. P. Koumoutsakos (ETH Zurich) for fruitful discussions, for which K.F. and K.F. also thank Dr. K. Taira (UCLA). This work was supported through JSPS KAKENHI Grant No. 18H03758 by the Japan Society for the Promotion of Science.

[1] M. M. Rai and P. Moin, Direct numerical simulation of transition and turbulence in an spatially evolving boundary layer, J. Comput Phys. **109**, 169 (1993).
[2] X. Wu and P. Moin, Direct numerical simulation of turbulence in a nominally zero-pressure-gradient flat-plate boundary layer, J. Fluid Mech. **630**, 5 (2009).
[3] A. Smirnov, S. Shi, and I. Celik, Random flow generation technique for large eddy simulations and particle dynamics modeling, J. Fluids Eng. **123**, 359 (2001).
[4] A. Keating, U. Piomelli, E. Balaras, and H.-J. Kaltenbach, A priori and a posteriori tests of inflow conditions for large-eddy simulation, Phys. Fluids **16**, 4696 (2001).
[5] P. Druault, S. Lardeau, J. P. Bonnet, F. Coiffet, J. Delville, E. Lamballais, J. F. Largeau, and L. Perret, Generation of three-dimensional turbulent inlet conditions for large-eddy simulation, AIAA J. **42**, 447 (2004).
[6] L. Perret, J. Delville, R. Manceau, and J.-P. Bonnet, Turbulent inflow conditions for large-eddy simulation based on low-order empirical model, Phys. Fluids **20**, 075107 (2008).
[7] M. Klein, A. Sadiki, R. Manceau, and J. Janicka, A digital filter based generation of inflow data for spatially developing direct numerical or large eddy simulations, J. Comput. Phys. **186**, 652 (2003).
[8] L. di Mare, M. Klein, W. P. Jones, and J. Janicka, Synthetic turbulence inflow conditions for large-eddy simulation, Phys. Fluids **18**, 055102 (2006).
[9] J. Hœpffner, Y. Naka, and K. Fukagata, Realizing turbulent statistics, J. Fluid Mech. **676**, 54 (2006).
[10] S. T. Lund, X. Wu, and D. K. Squires, Generation of turbulent inflow data for spatially-developing boundary layer simulations, J. Comput. Phys. **140**, 233 (1996).
[11] P. R. Spalart, Direct simulation of a turbulent boundary layer up to $R_\theta = 1410$, J. Fluid Mech. **187**, 61 (1988).
[12] X. Wu, Inflow turbulence generation methods, Annu. Rev. Fluid Mech. **49**, 23 (2017).
[13] C. Lee, J. Kim, D. Babcock, and R. Goodman, Application of neural networks to turbulence control for drag reduction, Phys. Fluids **9**, 1740 (1997).
[14] C. Choi, P. Moin, and J. Kim, Active turbulence control for drag reduction in wall-bounded flows, J. Fluid Mech. **262**, 75 (1994).
[15] M. Milano and P. Koumoutsakos, Neural network modeling for near wall turbulent flow, J. Comput. Phys. **182**, 1 (2002).
[16] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, Learning representations by back-propagation errors, Nature (London) **323**, 533 (1986).
[17] M. Gamahara and Y. Hattori, Searching for turbulence models by artificial neural network, Phys. Rev. Fluids **2**, 054604 (2017).
[18] J. Smagorinsky, General circulation experiments with the primitive equations: I. Basic experiment, Mon. Weather Rev. **91**, 99 (1963).
[19] J. W. Deadorff, A numerical study of three-dimensional turbulent channel flow at large Reynolds numbers, J. Fluid Mech. **41**, 453 (1963).
[20] J. Ling, A. Kurzawski, and J. Templeton, Reynolds averaged turbulence modeling using deep neural networks with embedded invariance, J. Fluid Mech. **807**, 155 (2016).

[21] J. Huang, L. Duan, J. Wang, R. Sun, and H. Xiao, High-Mach-number turbulence modeling using machine learning and direct numerical simulation database, in *55th AIAA Aerospace Sciences Meeting*, AIAA Paper 2017-0315 (2017).

[22] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE **86**, 2278 (1998).

[23] X. Guo, L. Wei, and E. Iorio, Convolutional neural networks for steady flow approximation, in *KDD' 16: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM* (ACM, New York, 2016), p. 481.

[24] E. Yilmaz and B. J. German, A convolutional neural network approach to training predictors for airfoil performance, in *18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, AIAA Paper 2017-3660 (2017).

[25] Y. Zhang, W. Sung, and D. Mavris, Application of convolutional neural network to predict airfoil lift coefficient, *AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, AIAA Paper 2018-1903 (2018).

[26] G. E. Hinton and R. R. Salakhutdinov, Reducing the dimensionality of data with neural networks, Science **313**, 504 (2006).

[27] K. Fukagata, N. Kasagi, and P. Koumoutsakos, A theoretical prediction of friction drag reduction in turbulent flow by superhydrophobic surfaces, Phys. Fluids **18**, 051703 (2006).

[28] R. D. Moser, J. Kim, and N. N. Mansour, Direct numerical simulation of turbulent channel flow up to $\mathrm{Re}_\tau = 590$, Phys. Fluids **11**, 943 (1999).

[29] M. S. Shanker, M. Y. Hu, and M. S. Hung, Effect of data standardization on neural network training, Omega **24**, 385 (1996).

[30] Keras, Keras Documentation, https://keras.io/.

[31] Y. LeCun, L. Bottou, G. B. Orr, and K. R. Muller, Efficient backprop, in *Neural Networks: Tricks of the Trade*, 2nd ed. (Springer, Berlin, 2012), pp. 9–48.

[32] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization, arXiv:1412.6980 [cs.LG] (2014).

[33] L. Prechelt, Automatic early stopping using cross validation: Quantifying the criteria, Neural Netw. **11**, 761 (1998).

[34] Fukagata Lab. Keio University, Machine-learned turbulence generator, Ver. 2, http://kflab.jp/en/index.php?MLTG2.

[35] M. Quadrio and P. Luchini, Integral space-time scales in turbulent wall flows, Phys. Fluids **15**, 2219 (2003).

[36] S. Hochreiter and J. Schmidhuber, Long short-term memory, Neural Comput. **9**, 1735 (1997).

[37] P. R. Vlachas, W. Byeon, Z. Y. Wan, T. P. Sapsis, and P. Koumoutsakos, Data-driven forecasting of high-dimensional chaotic systems with long short-term memory networks, Proc. R. Soc. London A **474**, 20170844 (2018).

[38] K. Fukami, K. Fukagata, and K. Taira, Super-resolution reconstruction of turbulent flows with machine learning, J. Fluid Mech. **870**, 106 (2019).