Turbulent Flow Prediction using AI

Meiji University: Fluid Mechanic Laboratory

Final Year Supervised Research Report

Student: Youri Chancrin

Meiji University Supervisor: Yoshitsugu Naka

ENSEIRB-MATMECA Supervisor: Julien Allali

January 5, 2026

Contents

1 Abstract

This document presents my supervised research about turbulent flow prediction, done at Meiji University, in the Fluid Mechanics Laboratory, under the guidance of Prof. Naka Yoshitsugu, in the context of my studies at ENSEIRB-MATMECA, where Prof. Allali Julien was my supervisor for this internship. I will begin by explaining my motivations for this research, and why the subject of turbulence matters for the scientific community. I will then state my specific missions and how I leveraged my knowledge of computer science in this mechanical field of research. Finally, I will discuss how I tackled my tasks. Firstly by establishing an operational environment capable of handling vast amounts of data in a scalable manner and running extensive computation spanning several days. Secondly, by reproducing results from a reference paper with close resemblance. Finally, I will detail the various ideas I explored to improve prediction accuracy, such as input pretreatment and network architecture modifications, which yielded improvements ranging from 5 to 10% in the end part of the channel.

2 Subject's Context: Why and Where

In this introduction, I will explain my personal motivations and present Meiji University and the Fluid Mechanics Laboratory where I conducted this supervised research. Lastly, I will talk about the interest in turbulence in the scientific community.

2.1 Personal Motivations

My appeal for turbulence boils down to the complexity of the subject. I knew that it would involve dealing with vast amounts of data and extensive computations. In the context of my studies of computer science at ENSEIRB-MATMECA I wanted to tackle a task that would utilize and push me to develop skill sets related to my specialization in High Performance Computing (HPC) and Data Science. I believed that I would be capable of establishing an efficient workflow, whether it would be on an HPC cluster or a regular Linux server. I was also aware that I lacked sufficient knowledge of mechanical engineering but I trusted that it would be an interesting challenge as I

yearn to strengthen my general understanding of mathematics. Additionally, I believed that my beginner position on the subject could bring a fresh point of view to the possible solutions. This constitutes my motivation regarding the subject of my supervised research. I was very pleased as the subject exceeded my expectations regarding data analysis and interpretation. I had a great time thinking about dimension reduction and how to visualize results and velocity fields. Lastly, this supervised research helped me clear my vision as to which path I should take after my graduation. I now intend to pursue research through a PhD.

2.2 Meiji University Presentation

Meiji University is a private research university with four different campuses located in the greater Tokyo area. It has more than 30,000 students and 3,000 academic staff members. There are no less than 26 different schools, at both undergraduate and graduate levels, spanning across a wide range of disciplines, from law, business, politics, science, agriculture, art, letters and mathematics. As Meiji University aims to grow its international appeal, it created a new short-term research program to facilitate student exchanges amongst affiliated universities. The existence of my research is a result of this new opportunity.

2.3 Fluid Mechanic Laboratory Presentation

There are many laboratories in engineering-related fields on the Ikuta campus which comprises the Science and Technology School and the Agriculture School. They have three main purposes: conducting research regarding their subjects, providing lectures from basics to state of the art knowledge and finally, offering supervised research classes to students. The Fluid Mechanics Laboratory at the Ikuta campus is managed by my supervisor, Prof. Naka Yoshitsugu, and has 14 members, all of whom are Meiji University students, in both undergraduate and graduate schools. My role in the laboratory was to conduct research on my subject and present my progress every two weeks in a laboratory meeting. Additionally, I had a weekly meeting with my supervisor.

2.4 Interest of Turbulence

Turbulence is everywhere. From the movement on the surface of coffee, to the swirls detaching from the wing of an aircraft. It is a complex and chaotic phenomenon in which energy is transferred in intricate ways, through different scales of similar structures. One of the most common applications of turbulence studies is drag reduction, which allows for more efficient transportation or power generation. Turbulence is studied both with real world experiments in wind tunnels and simulations, such as Direct Numerical Simulations (DNS). Both methods present different advantages and can be combined in a study. Real-world experiments allow for realistic modeling but can prove cumbersome to set up and benchmark. Simulations can yield fast results for simple cases and all of the data of the system can be retrieved but it is hard to use on complex or edge cases. Bridging the gap between the two methods allows for combining their advantages. This is why the subject of this supervised research matters. It aims to retrieve the velocities inside a flow channel only from wall-measurements, which can be obtained with embedded sensors, without perturbing the flow.

3 Subject Presentation: Turbulent Flow Prediction Using AI

This supervised research is based on a paper named "Three-dimensional generative adversarial networks for turbulent flow estimation from wall measurements" [1]. This paper will be referred to as the reference paper throughout this document. The goal is to understand, reproduce, and improve their results. This subject mixes AI and turbulence. I will proceed with an explanation of the important terms, then I will present my missions.

3.1 Scientific Context: Reference Paper Explanation

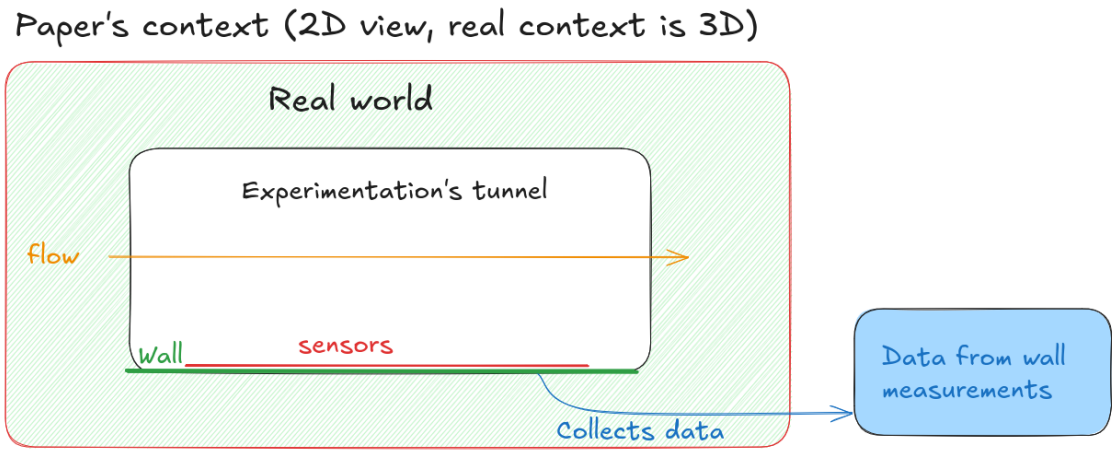


Figure 1: 2D paper context representation. A flow is going through a wall-bounded periodic channel.

Figure 2: Representation of the problem to solve: retrieving velocities inside the studied channel.

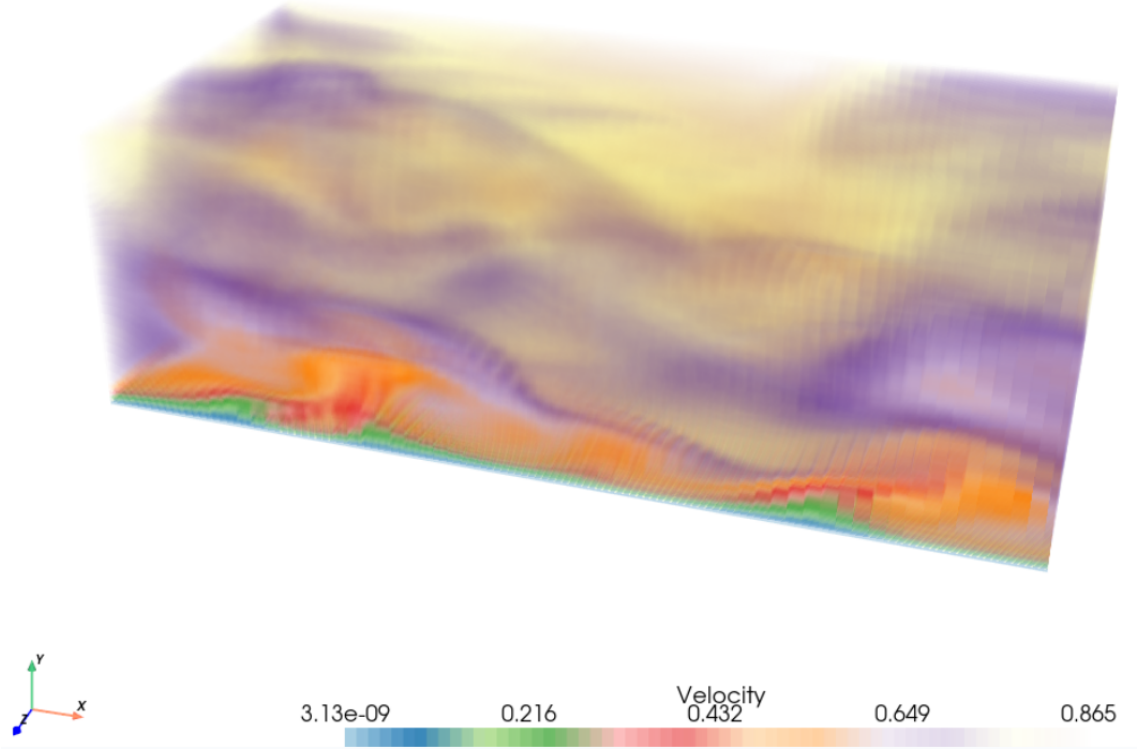


Figure 3: Visualization of the prediction area. Scalar represents the norm of the velocity.

The reference paper aims to predict the state of a three-dimensional flow channel from the wall measurements at Reynolds number 200 using a Generative Adversarial Network (GAN). Figures 1 and 2 explain the context in a 2 dimensional way. There is a channel containing flow, circulating in the stream-wise direction over the X axis. the channel is wall-bounded at $y = 0$ and $y = 2$. In the real world, sensors can be embedded in the wall to retrieve information. This correspond to the first layer of the channel information from the simulation. Additionally, the channel used is said to be periodic over the stream-wise and span-wise (x, z) axes, meaning that flow going out from one side is used as input on the opposite side. The flow information consists of three velocity components, u, v and w, respectively, in the stream-wise, wall-normal wise and span-wise (x, y, z) directions, figure 3 illustrates the prediction area.

The wall measurements are the shear stress, in x and z direction, and the pressure. The shear stresses are the forces acting parallel to the wall surface, divided by its area.

The Reynolds number (Re) is a key concept in fluid mechanics as it roughly relates to the complexity of the flow. The lower the Reynold Number, the simpler the flow's patterns. Below 2000, the flow is said to be laminar, between 2000 and 3000, the flow is unstable and above 3000 it is turbulent. In this work, the Reynold number is 200, which is relatively low. (Comment from Prof. Naka: Here, Re is the friction Reynolds number based on the friction velocity and the wall unit length. Hence: it is fine to say that $Re_\tau = 200$ is relatively low).

In the context of neural networks, a Generative Adversarial Network (GAN) consists of two networks working against each other, see figure 4. The generator part takes the normalized wall measurements as inputs, and predicts velocities normalized at each y layer. The discriminator part takes a normalized velocity field as input, and classifies it as generated or realistic. In a GAN architecture, the discriminator is only used in the loss of the generator, and applies a penalty if it detects the generator's output as generated. Finally, the three dimensional aspect refers to the prediction area, whereas in previous papers, it used to be only a two-dimensional slice of the flow channel, with a different network for each y layer.

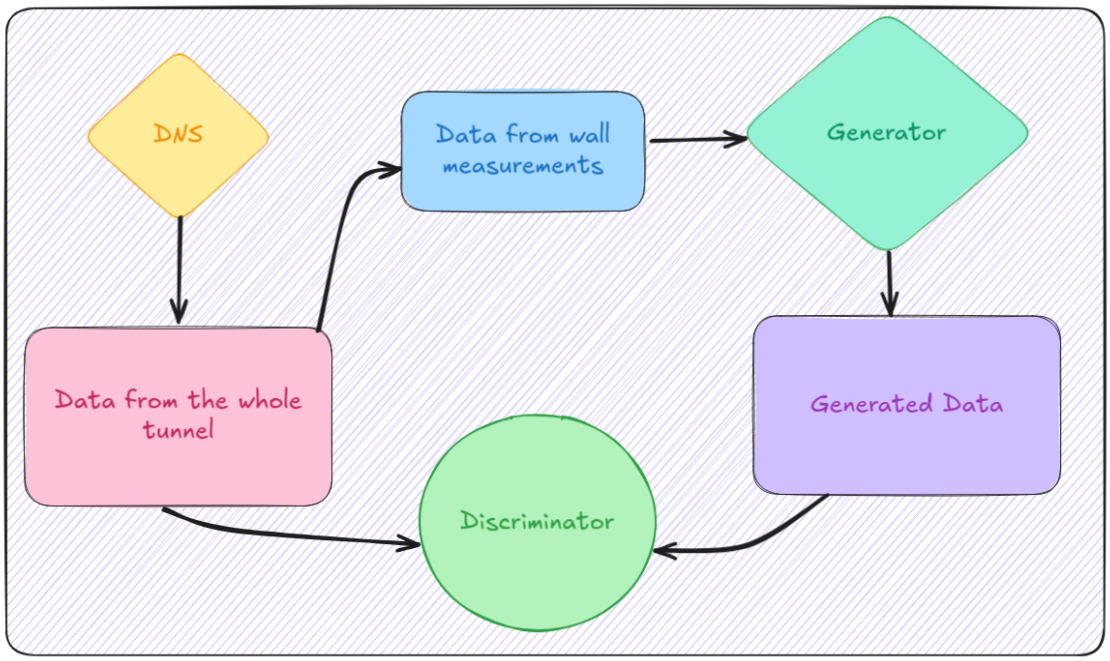


Figure 4: Representation of the GAN's different components. A Direct Numerical Simulation (DNS) is used to generate reference data. The generator is trained to recreate channel data. The discriminator has to distinguish between accurate data from the DNS and reconstructed data from the generator. They work against each other, hence the term 'adversarial' network.

I was initially concerned about the feasibility of predicting chaotic turbulence, as its study is often bound to statistical analysis. However, the chaotic factor depends on the Reynolds number of the flow, meaning that at low Re, the chaotic nature is limited. On top of that, chaotic does not mean that there is no link between wall information and the full velocity channel, but it remains a difficult task. Despite that, the reference paper still achieved good prediction accuracy in the near-wall region.

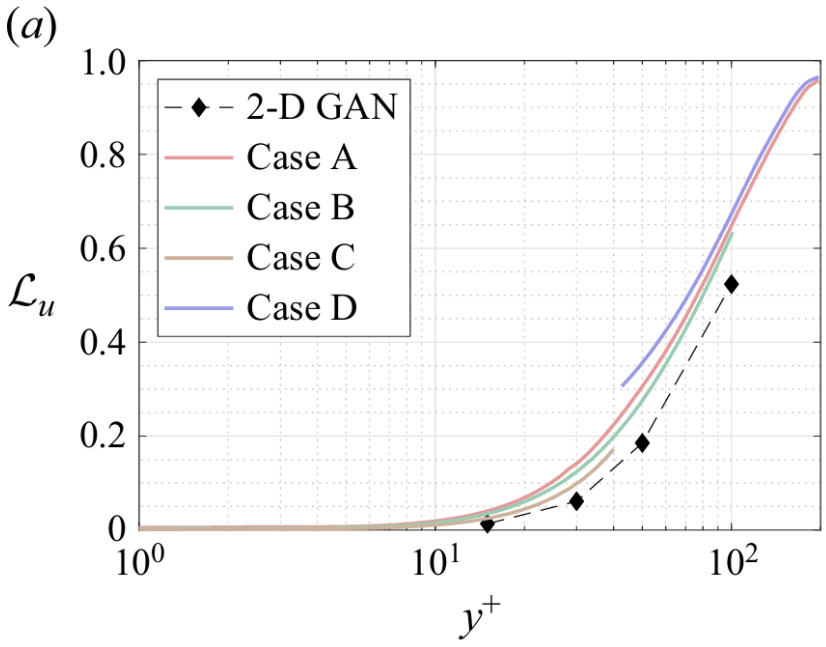


Figure 5: Mean Square Error of the u component. The lower, the better. This figure is taken directly from the reference paper [1]. Losses for components v and w have similar shapes and values.

As can be seen in figure 5, similar results to the 2-D GAN [2] methods were achieved, for a

fraction of the training cost. They present four different cases, which are defined by the domain of the prediction area. Case A covers from $y^+ = 0$ to 200, I chose to use it as a reference for my research. In this context, y^+ is dimensionless and based on Re; the half-channel point is when $y^+ = Re$, in our case 200. The full flow channel ranges from y^+ 0 to 400, since it is symmetrical, there is no need to consider more than $y^+ = 200$.

3.2 My Missions

My supervised research consists of three main tasks, which are understanding the reference paper, reproducing its results and finally, experimenting with ideas to improve the accuracy of the predictions.

To perform this research, I needed to establish an efficient environment, as computer science represents most of my skill set, I intend to leverage it to the fullest. This operational environment was built to handle data in a scalable manner, and to unify storage and treatment. Additionally, data visualization tools were included in order to facilitate comparison between benchmarks and ease training tracking. I will describe the technical needs and solutions I came up with, then I will explain how I handled reproducing the result of the reference paper. Finally, I will discuss my ideas to improve prediction accuracy.

4 Setting Up Environment

Setting up an operational environment is key for efficiency. I will run intensive computations such as turbulent flow simulations, deep learning training and data analysis, which all generate large amount of data. I will explain how I managed data in this project.

4.1 Technical needs: Large datasets and Scalability

The root of the data size boils down to the prediction area. I chose to keep the same as the reference paper, which is a 3-D grid of shape *(3, 64, 64, 64)*, 3 for the 3 components u, v, and w of the velocity and 64 points for x, y and z directions. When using *float32*, this results in $3 \times 64 \times 64 \times 64 = 3$ MB per sample. Training datasets used in the paper contains 20,000 samples, resulting in roughly *60 GB*. This amount is still manageable at the scale of one computer, but it starts to be challenging to manipulate. This becomes an issue when data needs to be refined and stored, whether raw simulation data, deep learning model weights, or benchmarks. This is why I built the following architecture.

4.2 Data Management Framework

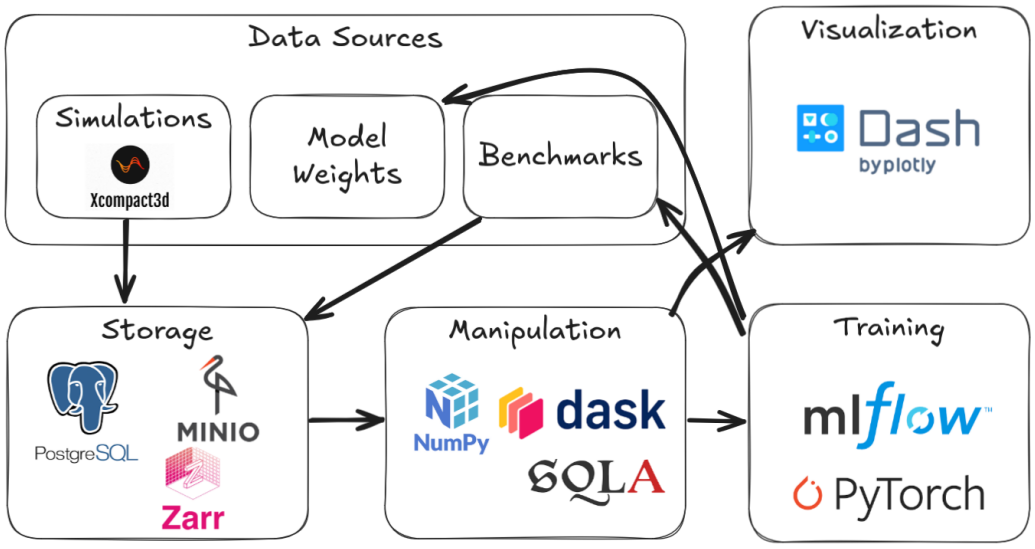


Figure 6: Illustration of the main libraries used for the *data management process*. An arrow indicates a path where data can go from one module to another. For instance, tensors used for Training require data from the manipulation module, such as *Numpy* arrays. Afterward, benchmarks can be generated, they are then stored back in *MinIO* as *Zarr*, *Pandas* dataframes or *Parquet* files.

Figure 6 explains the way data is handled. It all begins with flow simulations. I used an open-source *Direct Numerical Simulation (DNS)* named *XCompact3D* [3]. It generates snapshots sampled through the simulation at a specified rate, in *XDMF* format, readable by *ParaView*. I wrote a script to extract the relevant information, which are velocities, the wall shear stress and wall pressure and convert them into *ZARR* format stored in *Minio*, an object database. Unlike traditional *SQL* database, object databases are made to store large objects rather than large numbers of rows. The *ZARR* format handles compression and chunking, allowing for lazy loading of the data. Additionally, *PostgreSQL* is used to store metadata about the datasets, such as the channel mesh used, its storage location and the mean values.

To manipulate data in *Python*, I used the *SQLAlchemy* and *Dask* modules. *SQLAlchemy* is a well-known *Object-Relation Mapping (ORM)* library for *Python*, and *Dask* can be seen as a distributed version of *NumPy*, which can be less efficient on a single machine but has better scalability. I chose it because it simplifies retrieving data from *Minio*, and could support larger datasets and make it possible to port my environment to a computing cluster.

The deep learning models are implemented in *PyTorch*, and *MLflow* is used to track and record training.

Finally, I also built a web interface with *Dash* to ease visualization and comparison between the generated datasets and training runs.

This architecture, somewhat complex for the scale of my research, allows me to generate, manipulate, and benchmark data easily.

5 Reproducing Reference Paper Results

Reproducing the results of the reference paper marks an important milestone, as it establishes that my processes are valid and that the reference paper's results are trustworthy.

5.1 Generating Turbulence Datasets

The key to reproducibility is generating similar turbulence datasets, which is achievable using similar simulation parameters. However, to do so, some mechanical understanding of the variables is necessary.

Firstly, the channel has to have a similar mesh, domain size, and beta refinement. The beta refinement factor is used to increase the density of points near the wall region, where the scale of turbulence is smaller and requires higher mesh resolution.

The target Reynolds number is set to 200. The time unit used is in Eddy Turnover Time (ETOT), which corresponds to the duration of the rotation of the largest eddies in the flow. It means that in 1 ETOT, the largest eddies complete one revolution.

The time step is set to 0.005 ETOT. The time integration scheme is Runge–Kutta 3.

The higher the Reynolds number, the more turbulence in the flow. However, turbulence initially appears due to infinitesimal perturbations. In the real world, this is unavoidable, but in simulation, it is possible to have no perturbation, hence no turbulence; leading to unrealistic flow. To prevent this, an initial noise of 2% is added to the channel at the beginning. Additionally, the channel is rotated by 0.12 each timestep in a warm-up phase of 5000 time-steps. This is followed by a stabilization phase of 35,000 time steps and finally, the sampling phase begins, as illustrated in Figure 7.

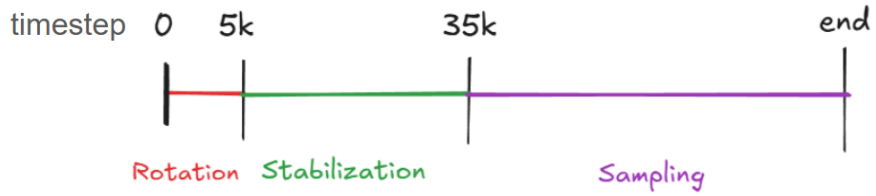


Figure 7: The three phases of the flow simulations.

Sampling also has some complexity. By sampling too frequently, each snapshot will be highly correlated to its neighbors, and it will degrade the statistical properties of the generated dataset. If sampling is too sparse, the total time needed to run the simulation increases, linearly.

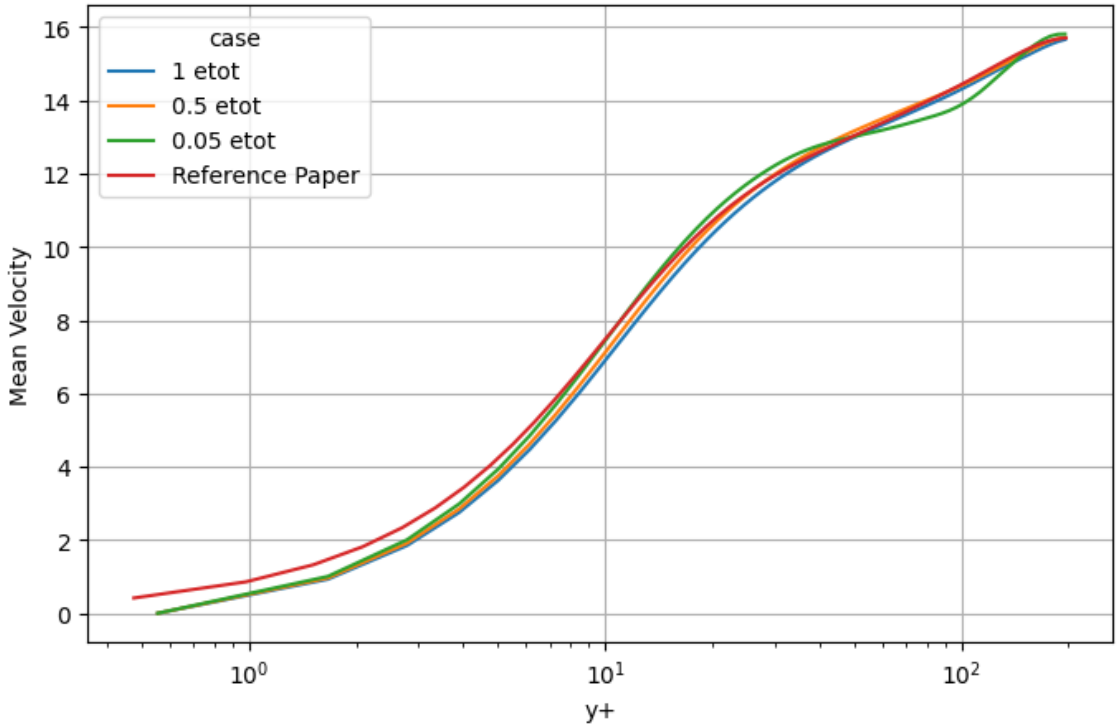


Figure 8: Mean streamwise velocity component U of the generated datasets and the validation dataset from the reference paper (paper-train). The closer the curves, the better.

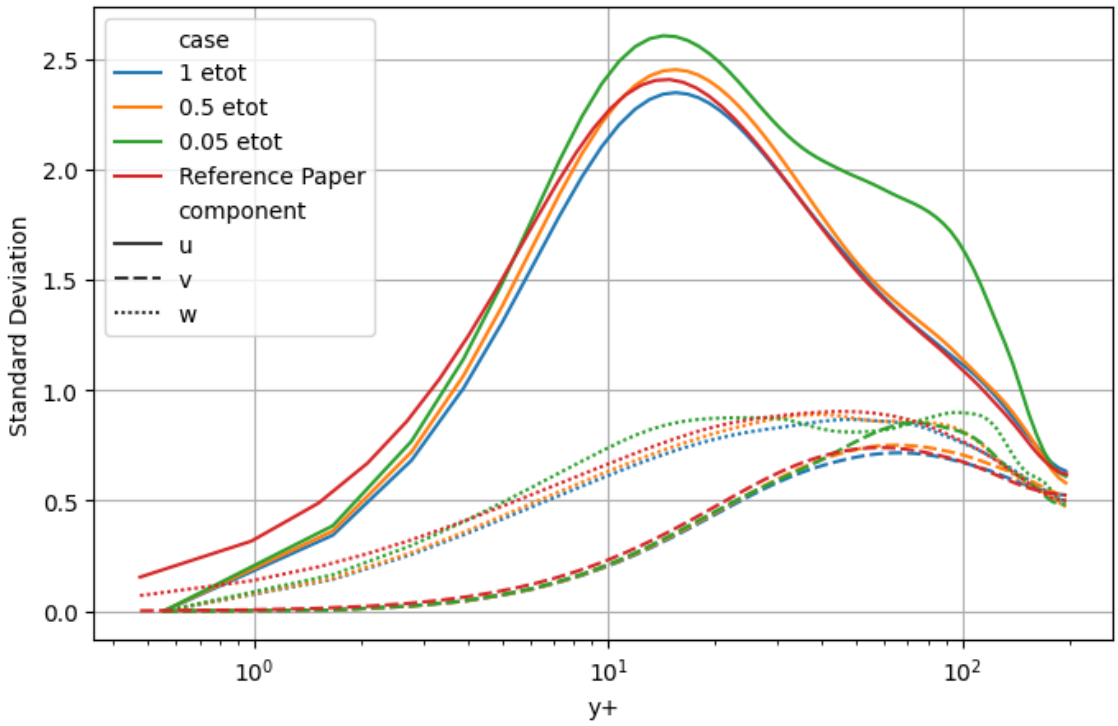


Figure 9: Mean Standard deviation of the generated datasets and the validation dataset from the reference paper (paper-train). The closer the curves, the better.

Figures 8 and 9 show statistics of the generated datasets and compare them with the validation dataset of the paper. I performed simulations with three different sampling rates, 0.05 ETOT between snapshots, 0.5 and 1. Since the simulation time step is 0.005 ETOT, this correspond respectively to sampling every 10, 100 and 200 time steps. The reference paper used 0.5 ETOT. We can observe that discrepancies appear when using 0.05 ETOT, showing that snapshots must be spaced sufficiently to achieve statistical independence and produce correct datasets.

I was satisfied with datasets having correct mean profiles over the y-axis (fig 8). However, I still thought that it would be interesting to conduct a deeper analysis about the representativity of my datasets. The prediction area covers half of the full channel height, resulting in $64 \times 64 \times 64$ (x, yz) points of u, v and w velocity components (the full simulation channel being $64 \times 128 \times 64$). Each point is stored as a 32-bit float, which can take up to 2^{25} distinct values. This means that there are approximately $3 * 64^3 * 2^{25} \approx 2.6 \times 10^{13}$ possible states of the channel, which is considerably large. Only a small subset of this are realistic channel states, and within this subset, many states are similar.

To ensure that our dataset are representative about the possible channel states, I wanted to perform a dimension reduction and display the dataset on a single segment for instance, as seen in figure 10. I did not follow with this idea, as it felt out of scope, though I still believe it would be important to ensure quality of the training datasets. I eventually followed the advice of my supervisor, Prof. Naka who explained that turbulence has a quasi-periodic nature (turbulence also share chaotic and non-linear nature). Hence, as long as sampling is performed over many ETOT, a good representation should be obtained, as illustrated in Figure 11.

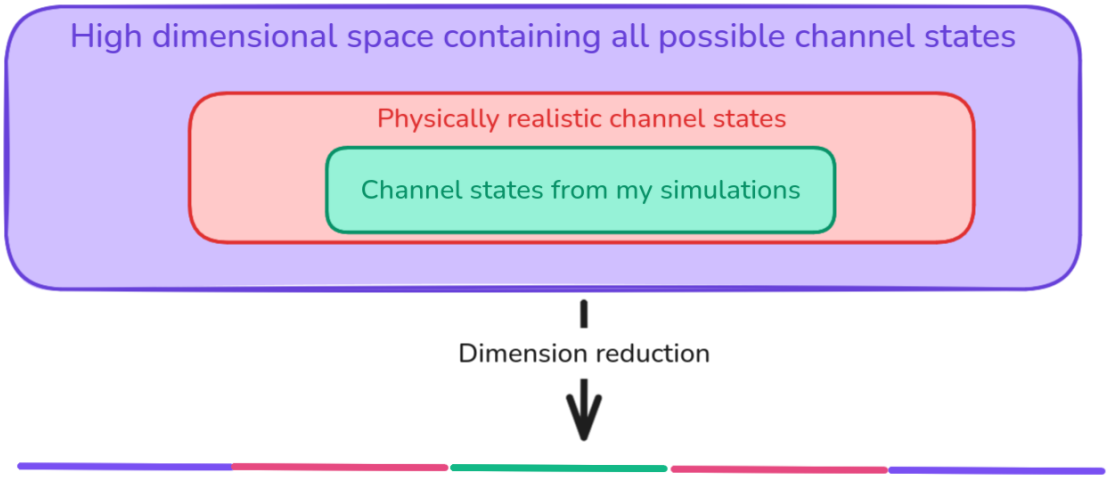


Figure 10: Illustration of dataset representativity on single axis. Hypothetical dimension reduction from $3 \times 64^3 = 786,432$ to 1.

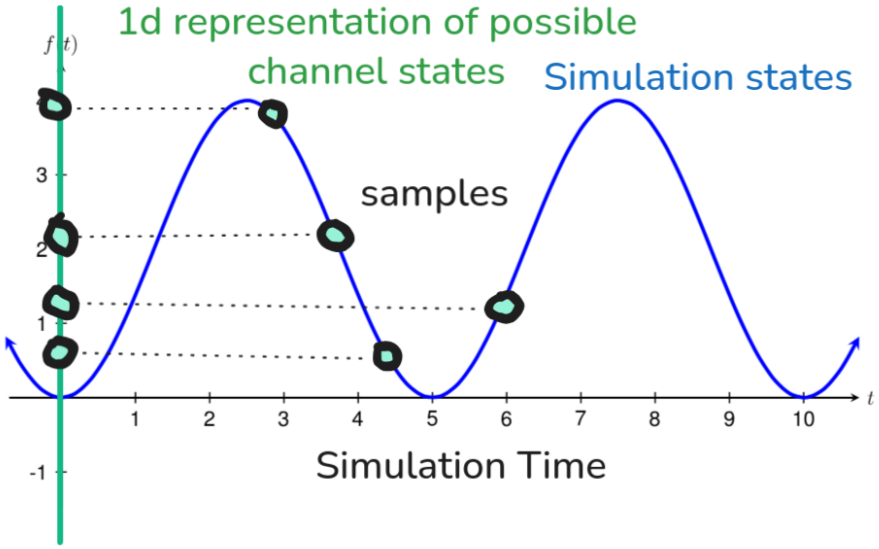


Figure 11: Illustration of the quasi-periodic nature of channel states. Note that this illustration does not explain the discrepancies in mean and standard deviation when using 0.05 ETOT, (see Figures 8 and 9). The main takeaway is that, between two ETOT, the simulation returns to a similar state, but with differences since the smaller structures evolve at a different rate. Hence, as long as the simulation is run for long enough, we can expect a good coverage of the possible states.

5.2 Training Models

Once valid datasets have been acquired, performing the network training is rather straightforward. I used the same network architecture with Pytorch (instead of Tensorflow, used in the reference paper) as the backend. I based my training cycle on the reference paper, using the Adam optimizer and a base learning rate of 10E-4. I added a learning rate scheduler to reduce it when hitting a plateau, as well as an early stopping criterion that ends training if no improvements are made after some epochs, instead of always running 20 epochs.

The reference paper used 20,000 samples for training, 4,000 for validation and 4,000 for testing. I chose to use 20,000 samples for training and validation, with sample rate of 0.5 ETOT, and another testing dataset of 1,000 snapshots, with a sample rate of 1 ETOT, obtained from a different simulation to prevent contamination.

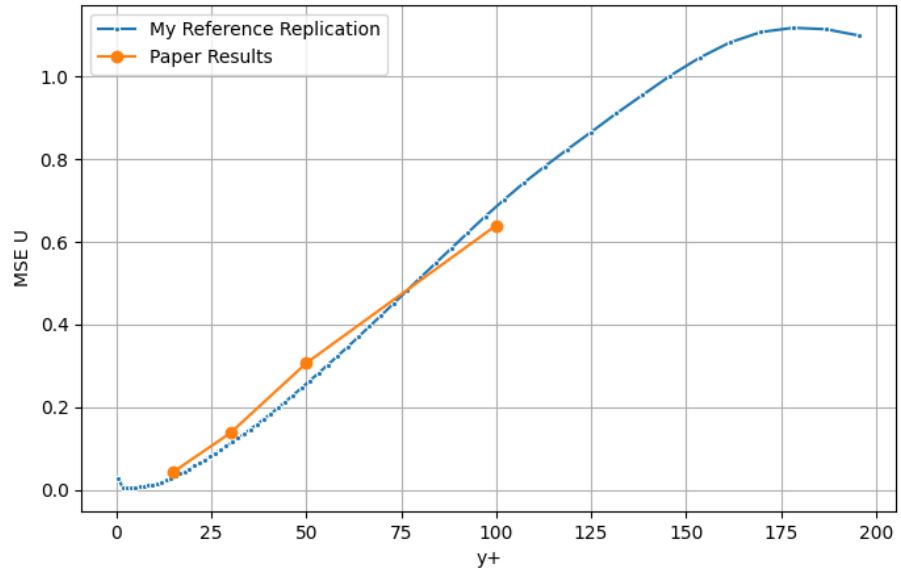


Figure 12: Comparison between the reference paper results and the results I obtained following their method. The lower, the better. MSE for v and w components have similar shapes and values.

Figure 12 shows that I have achieved similar results with my environment using a similar workflow to what the reference paper described. This proves that the paper process is replicable, and that my environment is functioning properly.

6 Exploring Ideas for Improvements

Compared to previous works [2], the reference paper [1] actually doesn't really improve accuracy, but reduces the complexity of the generator part by using only one network, at the cost of slightly worse accuracy. In previous papers, one network is used per y layer, hence velocities are normalized per layer. In the reference paper, they used the same per y layer normalization. This normalization deforms the prediction area as it becomes non-uniform. My first idea was to limit the normalization per component over the full channel to conserve scale uniformity. Hence, I wanted to study if scale uniformity would improve predictions. Secondly, when exploring the Github repository linked to the reference paper, I noted that there were many versions of networks with minimal changes, but I found little explanation regarding the motivations behind them. This raised my curiosity and I decided to investigate the architecture of the network, in particular, the influence of the discriminator. Finally, on the advice of my supervisor Pr. Naka, I analyzed the effect of applying filters on the inputs to extract different scales of patterns.

6.1 Inputs Normalization

In this first part, I will explain how I studied the effect of uniformity on predictions. It should be noted that, while we can preserve the scale uniformity of the prediction space, it is still not fully uniform as the spacing in the y dimension varies.

My starting hypothesis is that having a scale-uniform prediction space results in easier structure reproduction as scales should be maintained along the y-axis.

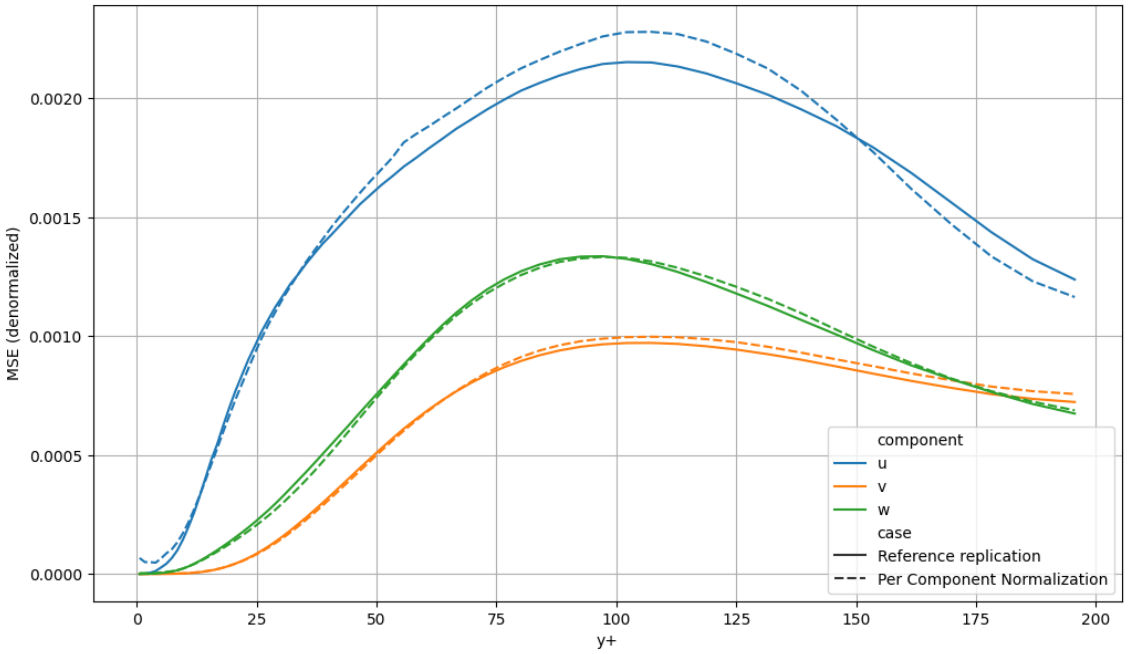
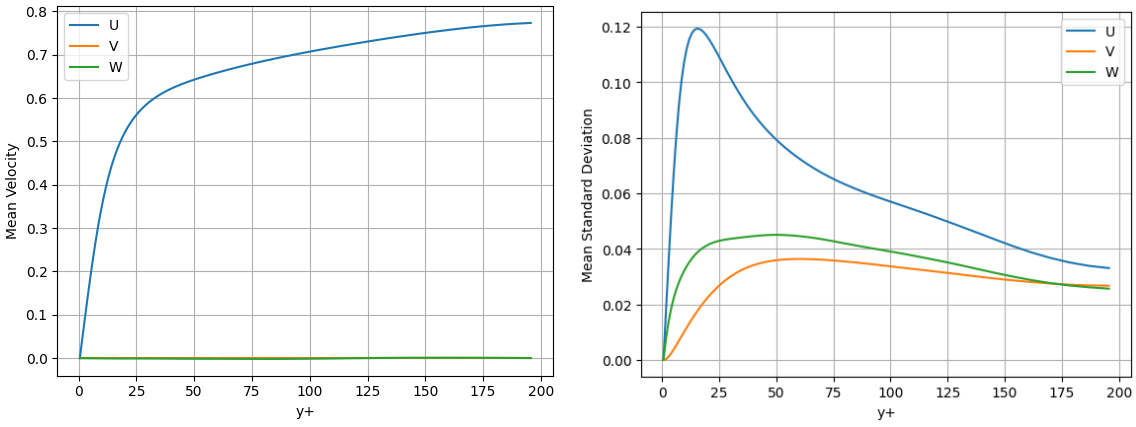


Figure 13: Comparison between my reference paper replication, which uses normalization for each y layer and for each velocity components, and another model that only normalize each component u, v and w. The predictions of the networks are then de-normalized to be compared together.

I ran a new training and benchmark, as can be seen in figure 13. We observe that, for components v and w, changing the normalization makes almost no difference; this can be explained since the y dimension mainly affects the streamwise velocity u, as shown in figure 14.



(a) Mean Velocity along y^+

(b) Mean Standard Deviation along y^+

Figure 14: Unscaled Velocity and Standard Deviation of the benchmark dataset, re200-sr1etot, which acts as ground truth. Note that mean velocities of v and w component are almost always 0.

When analyzing the influence of normalization on the u component, we can split the channel into three parts. From y^+ 0 to 50, almost no difference. From 50 to 150, the component only normalization performs worst. Beyond y^+ 150, it performs better.

My hypothesis is that, normalizing over each y layer yields good results because it minimizes the bias that the network has to learn, as in, the average expected output is closer to 0. The issue is that, as we get further away from the wall, the scale of the turbulent structures increases. The per-component only normalization offers a prediction space with better uniformity, which should make it easier to rebuild large structures, hence increasing accuracy.

I chose to keep using the reference paper normalization as it has slightly better accuracy overall.

6.2 GAN Architecture Study

I wanted to understand the influence of the adversarial part of the GAN. The discriminator is only used in the loss function of the generator, in this way, it can be thought of as a realistic or physical constraint penalty. I will first discuss some concerns about establishing a good loss function, and then I will explain my experiments regarding tuning the discriminator's influence.

6.2.1 Loss function concerns

Generally, GANs perform well in image generation tasks, as the discriminator helps determine the quality of an image because it is hard to define hard-coded metrics to judge it. However, in the case of turbulence, the realism of the velocity flow can be checked with fluid equations, such as continuity properties. Using physics equations in the loss function of a network's training is called making a Physics-Informed Neural Network (PINN), and it is a popular topic. I did not explore it because there are many physical constraints that can be applied, and they have different kinds of dimensionality.

The loss function used for benchmarking is the Mean Squared Error (MSE). It computes the difference between each value of the generated and expected fields, squares it and averages it. Hence, MSE has the same dimensionality as the data being compared, meaning that no dimensional information is lost when comparing sets. For instance, comparing the mean over the y axis of two sample sets would blur the x and z dimension information. This is not necessarily a bad thing. The issue is that MSE might not be the best loss function, as it is not enough to determine if a sample or a set of samples is realistic. It should also be noted that, when comparing sets of samples instead of single samples, it becomes possible to do dataset analysis, such as Standard Deviation (std) or mean.

My hypothesis is that some information from the wall allows us to infer the presence of partial structures in the channel. Since turbulence is chaotic, some information might be missing to fully reconstruct the structures. My guess in this situation is that there can only be a limited number of patterns explaining the information from the wall. By reasoning on a global scale, some patterns could be invalidated if their influences on neighboring structures is not physically correct. This would require a physical constraint for the network to consider, and a solver acting as a guide might also be necessary.

Currently, I believe that with MSE, the network ends up using the average of the possible structures that could exist in a place, resulting in physically incorrect predictions and lower standard deviation, as shown in figure 15.

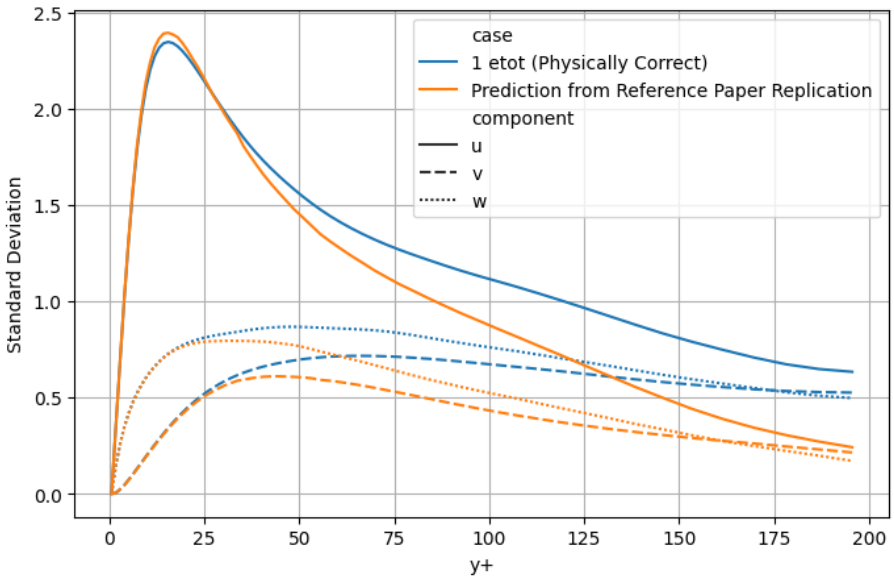


Figure 15: Comparison between a physically correct simulation (blue), and a dataset generated from the network of the reference paper replication, using wall data from the physically correct dataset (orange). The closer the orange curves are to the blue ones, the better. Note that, in the near-wall region, the std is correct. Beyond $y^+ = 30$, the orange curves' std decreases faster than the physically correct ones. Lower std can be interpreted as less confidence in the network's predictions.

To sum up, lower dimensional physical constraints would give indications to the network about the realism of its prediction. However, they might not improve predictions, as the network could lack guidance on how to improve them. For instance, it could result in skewing certain areas of the prediction to fit the global physical constraint while worsening certain local spaces.

6.2.2 Discriminator Influence

I believed that using a metric to judge the global realism of the prediction was a good idea, but I doubted that the discriminator was really improving results. For this reason, I wanted to understand the influence of the discriminator during the training and how it can be tuned.

The generator loss is computed in the following manner: $generator\_loss = content\_loss - discriminator\_influence * discriminator\_penalty$, where $content\_loss$ is the MSE between the predictions and the expected values, and $discriminator\_penalty$ is 1 if the discriminator judges the prediction as fake and 0, if judged as realistic. In the reference paper, discriminator influence is set to 10E-3. I fidgeted with different configurations, ranging from no discriminator to 10E-1.

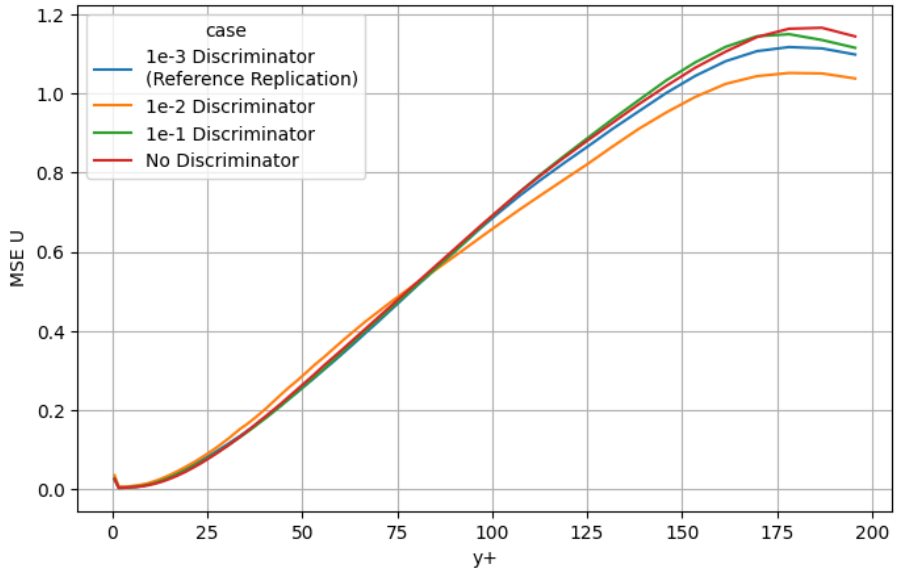


Figure 16: Comparison of the u MSE when using different discriminator influences in the loss function. The lower, the better. MSE of components v and w have similar scales and values.

Figure 16 shows that below y^+ 100, the results are very similar regardless of the discriminator influence. Above y^+ 100, an influence of 0.01 has the best results, while using no discriminator has only slightly worse results than when its influence is too high or too low. It seems that there is a sweet spot in the middle. Interestingly, in the below y^+ 100 part, the 0.01 influence performs the worst, while the no discriminator case is the best. As I ran only one training for each case, these results could be influenced by randomness, especially considering that the higher the discriminator influence, the more unstable the training.

My hypothesis is that, in the near wall region, there is a strong correlation between wall-measurements and velocity, hence the discriminator offers very limited improvements. It might even penalize results, since the loss combines two factors, the content and the discriminator parts, if the latter is too influential, it overshadows the content part and limits the network improvements. As the correlation with wall-measurements decreases the further in the channel, the need for global consistency increases, resulting in improvements thanks to the discriminator.

To sum up, the discriminator offers no improvement for the near wall part of the channel, but when tuned correctly, better results are achieved in the end part of the prediction area.

6.3 Scale Filtering on the Inputs

The idea behind scale filtering is that the wall data contains information about all of the flow channels, but the close patterns have greater intensity, shading the influence of the larger patterns, which have smaller intensities but extend in broader ways. By applying certain band-pass filters, I aim to extract information that is tied to the larger-scale patterns existing in the middle of the fluid channel.

For this experiment, I looked at what kind of filter to use, then I searched which parameters would be best-fitting with Bayesian optimization and reduced their amount with clustering. Finally, I ran full-scale training to evaluate the performance of this idea.

I chose to use a difference of Gaussian (DoG) filters, see figure 18, as they have good properties for extracting larger scales, as shown in figure 19. They consist of the difference of two gaussian blurs. Each gaussian blur has one parameter named *sigma*, which decides the shape of the blur applied, see figure 17. Hence, a DoG filter has two parameters, s1 and s2.

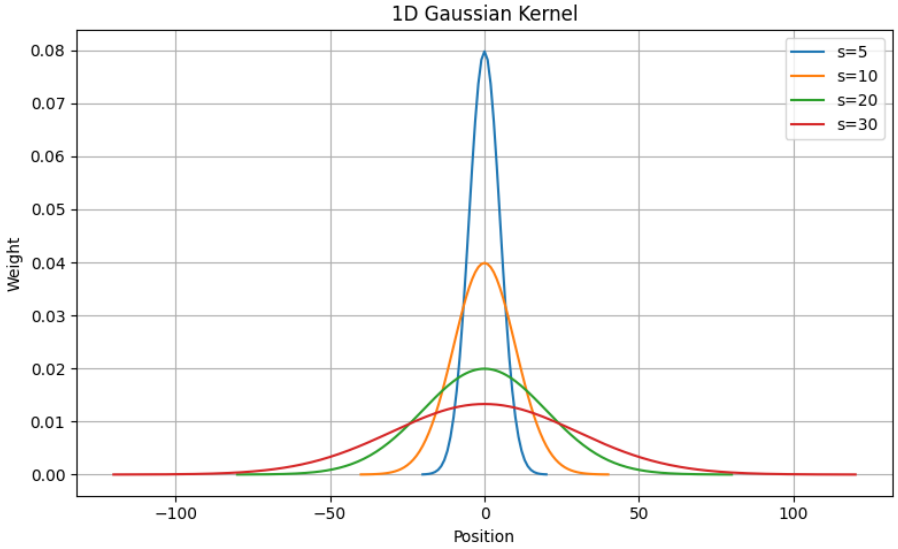
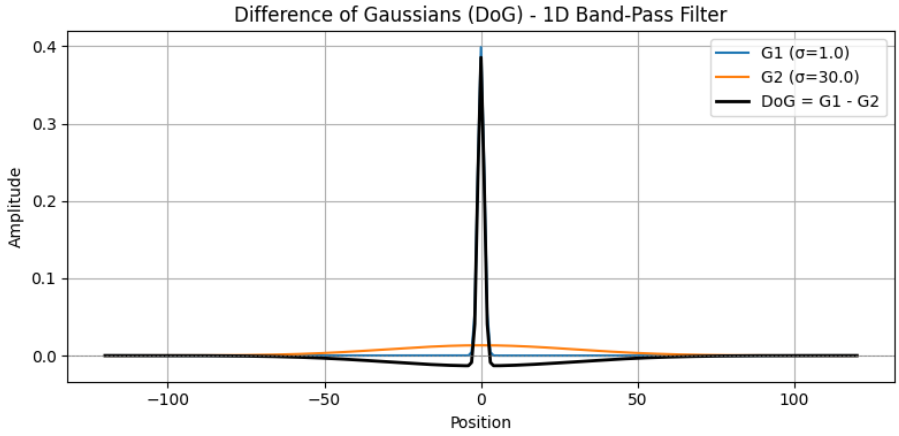
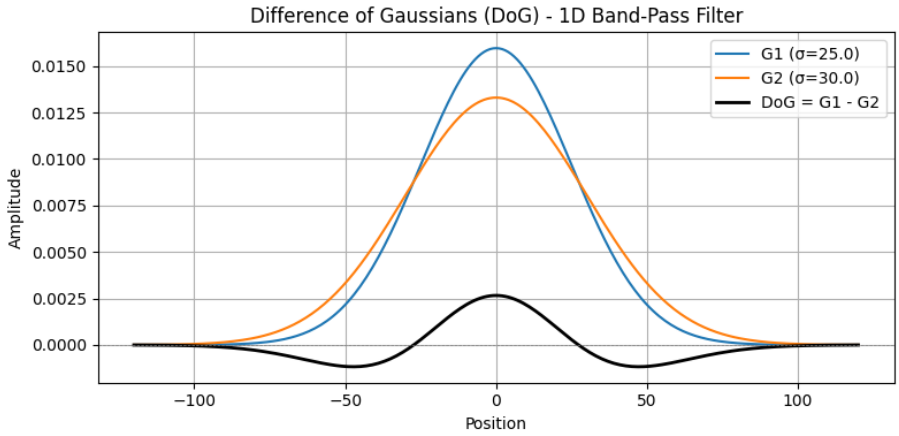


Figure 17: Illustration of various 1 dimensional gaussian filters. The bigger the sigma, the bigger the blur.



(a) Sharp DoG filter: Using low s1 and high s2 creates a filter retaining small scale patterns.



(b) Wide DoG filter: Using high s1 and s2 creates a filter retaining only wide patterns.

Figure 18: Illustration of different DoG filters.

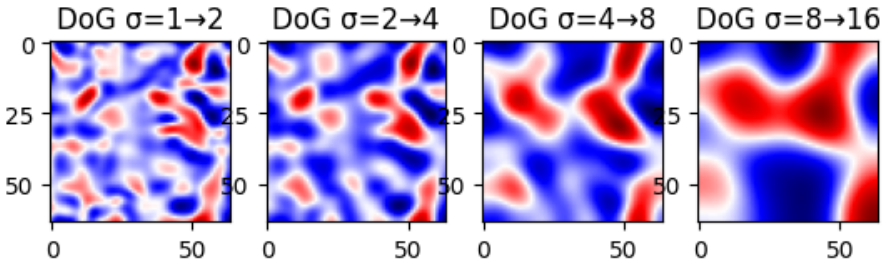


Figure 19: Difference of Gaussian filter illustration. The bigger the sigmas, the broader the extracted patterns.

To figure out which parameters to use is to figure out which scales we want to expose for the network. There are three ways to solve this, first with a critical analysis; using previous scientific data we could have an idea of the scale to extract because the wall-bounded flow case is well studied. Secondly with spectral analysis; by using a 2-dimensional Fourier transform on each y layer, we can observe the scale of structures. I was successful in doing such analysis but not in interpreting it. Finally, by running Bayesian optimization; it only requires an evaluation function for a set of parameters. I chose to use Pearson (linear) correlation for each y layer and I obtained for each pair of network input (pressure, shear stress x and shear stress z) and output (u, v and w velocity components). I chose to keep only the ones with the best correlation improvement.

I set two constraints for the Bayesian Optimization search, $s2 > s1$ and $s2 < 30$, to limit the search space. I chose 30 as a limit because the input field size is only 64 by 64, and using a *sigma* of more than 10 is already too wide, and loses some information, as illustrated in figure 17.

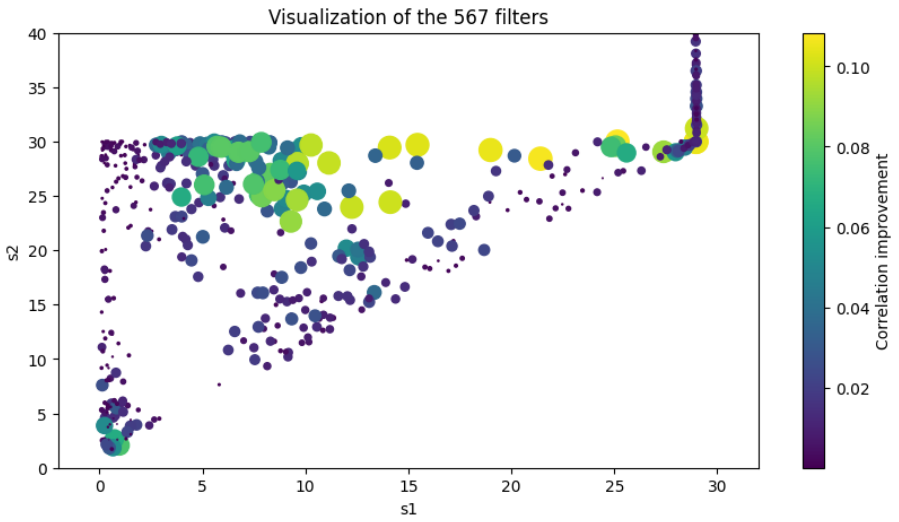


Figure 20: Two-dimensional representation of the DoG filters. The color and size represent the potential correlation gain of a filter. Note that the missing lower triangle is a result of the constraint $s2 > s1$. The maximum s2 limit was set to 30, for some reason when s1 was too close to it, the Bayesian Optimization suggested higher s2.

This results in the study of $3 * 3 * 63 = 567$ filters, as there are 3 network inputs: pressure, shear stress x and z, and 3 outputs: the u, v and w velocity components. It results in 9 pairs of input-output to analyze over the 63 y layers.

I represented the filters on a 2-D plane, as visualized in figure 20. We can see that most of them have s2 close to 30, but s1 ranges from 5 to 30. This is a sign of good results as it reflects the wide range of scales in our flow data. Figure 21 shows the average gains in correlation thanks to the filters. I chose to discard some of the pairs of input-output to focus on the ones with the highest correlations. I kept the pressure to u component, the shear stress x to u component, the shear stress x to v component, and finally, the shear stress z to w component. Figure 21 also shows that the average correlations of the 4 best combinations are far greater. The gains are limited in the near wall region, as correlation is already high, but the middle part of the channel presents average correlation improvements of 5%.

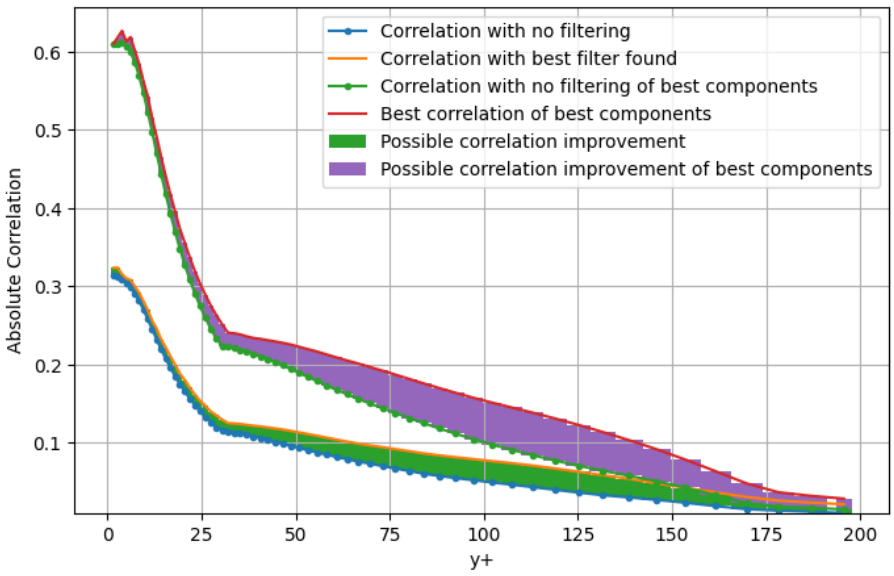


Figure 21: Comparison of correlation improvements using DoG filters. The bottoms of the areas are the average correlations with no filters. The top of the areas are the average correlations of the best filters. The green area is made using all of the pair of input to velocity components, whereas the purple area is made when retaining only the best combinations.

There are still too many filters to use them all in a practical application. Luckily, most of them

are very similar, so I used clustering to regroup them. Clustering takes the number of clusters as a parameter, to figure it out, I tested a range of integers from 1 to 24, see figure 22, and computed the best correlation using the reduced sets. I chose to take 8 clusters as it presented the best improvements with reasonable input increase (figure 23).

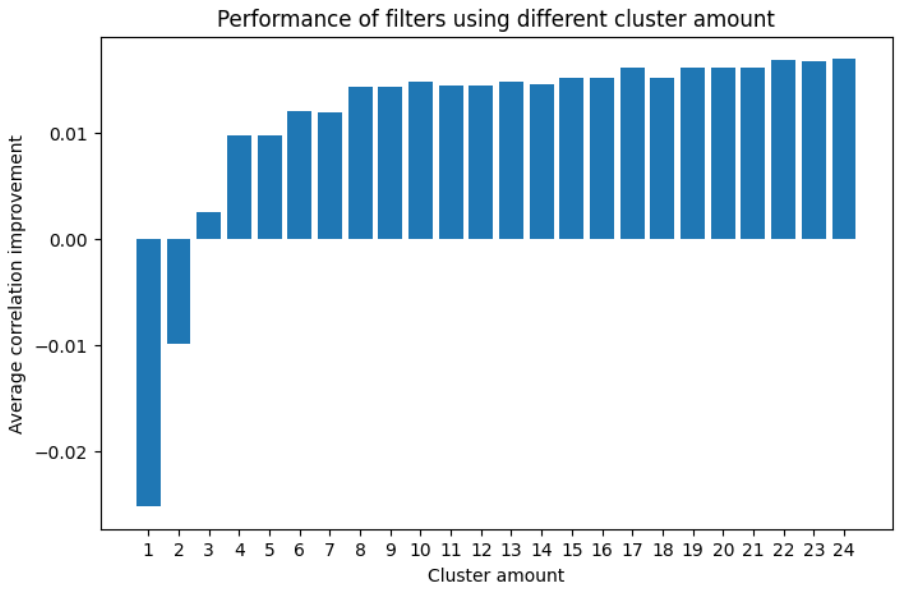


Figure 22: Average correlation improvement when using different numbers of clusters. When using less than 3 clusters, the obtained correlation is lower than when using no filters. Note that the logarithm shape of the correlation improvement against the number of clusters shows that a small amount of clusters can express the information of many filters.

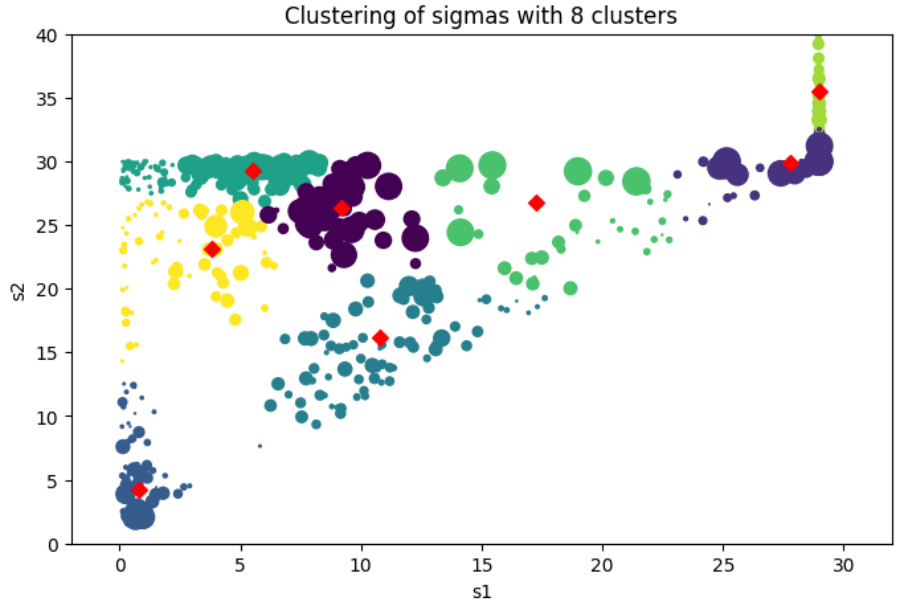


Figure 23: Representation of the filters clustering in 8 groups. The size of the filters is the correlation gain they provide. The clustering weights are initialized with these correlations.

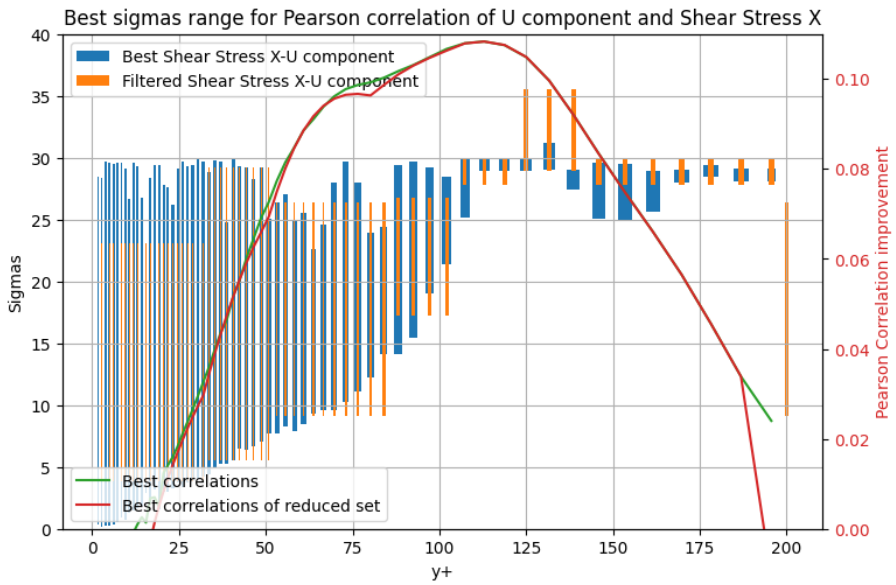


Figure 24: Illustration of the various sigma parameters and the correlation gain with their corresponding DoG filter, along the y dimension. The blue set is the one found by the Bayesian Optimization. The orange set is the reduced set from the clustering.

Figure 24 presents some interesting facts. Firstly, the Bayesian Optimization were performed independently, but we observe an effect of linearity in the filters extracted, which is a good sign of its validity. Secondly, the correlation improvements from the reduced set are very close to the original set, which means that the clustering was successful.

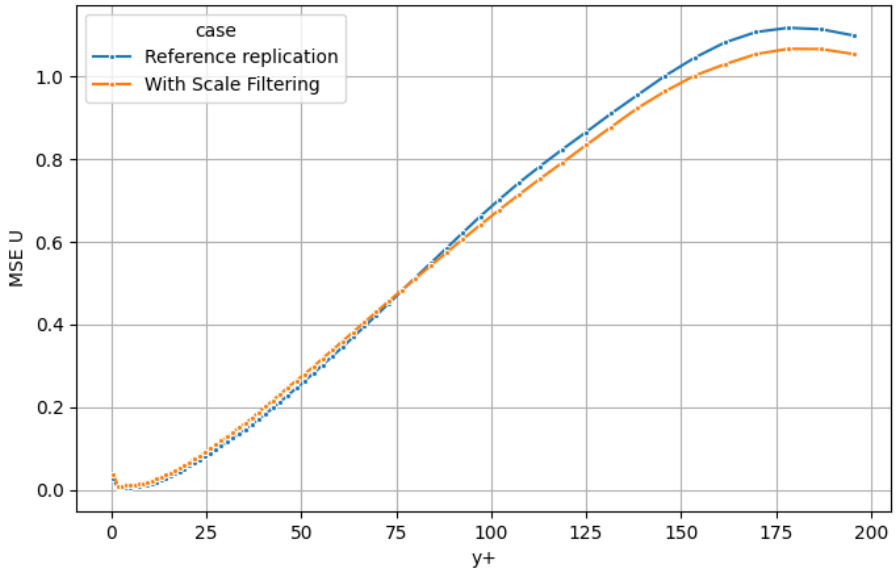


Figure 25: Component u MSE comparison between reference replication and model using additional channel inputs from scale filtering. The lower, the better. Components v and w have similar scales and values.

I ran a training with the limited filter set as additional input layers. This has limited influence on the network size as only the input part is affected, the hidden layer which constitutes the bulk of the network stays unchanged. I obtained improvements in the end part of the channel, but losses in the 0 to 75 range of y^+, see figure 25. This improvement is very encouraging as it shows the importance of highlighting the wider-scale patterns. My hypothesis concerning the decrease in the near-wall region is that increasing the number of input results in a harder task for the network, as it now has to choose which input to use for which part of the channel. I believe finding a way to convey the locality of the new input channels to the network could solve the issue, as well as reducing the training instability, which is another problem that arose with the scale filtering

approach.

7 Conclusion

During this five-month supervised research, I established a complete working environment, replicated results from the reference paper, and investigated different ways of improving prediction accuracy. I was successful at all my tasks, but found only limited ways of improving predictions, mainly in the end part of the prediction area. Finally, my study offers a wider understanding of the influence of the discriminator, normalization, and input filtering.

Scale filtering inputs and fine tuning the discriminator influence offered prediction improvements in the far-end of the channel, but in return the near-wall part accuracy decreased, as shown in figure 26. Figure 27 shows up to 10% prediction improvement, with greater success for v and w components, and especially for the far end of the channel.

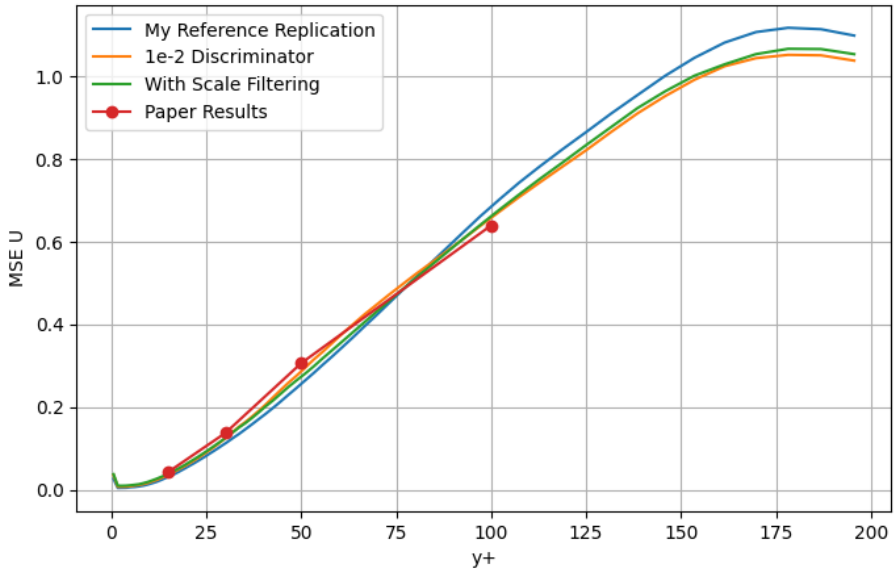


Figure 26: u Component MSE comparison of reference replication, scale filtering, and discriminator tuning.

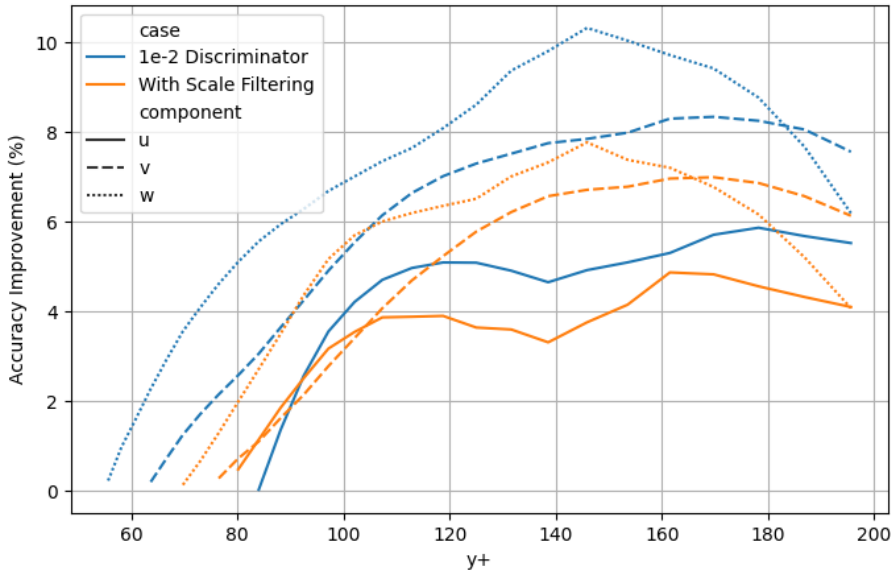


Figure 27: Final results. Accuracy improvements over my paper replication case, in %.

I believe that major changes in network architecture and workflow would be required to significantly improve predictions. UNet models with diffusion would be interesting to explore. I have

other ideas that I did not explore at all, which are: expanding the problem over multiples time steps at a time to increase the amount of accessible information, in the philosophy of SLAM algorithms (Simultaneous Localization And Mapping). Creating an intermediate representation of the flow, that encapsulates the diversity of scales and the recursive nature of turbulent structures, and combining wall sensor information and other methods, such as Particle Image Velocimetry (PIV).

References

[1] Antonio Cuéllar, Alejandro Güemes, Andrea Ianiro, Óscar Flores, Ricardo Vinuesa, and Stefano Discetti. Three-dimensional generative adversarial networks for turbulent flow estimation from wall measurements. *Journal of Fluid Mechanics*, 991, July 2024.

[2] A. Güemes, S. Discetti, A. Ianiro, B. Sirmacek, H. Azizpour, and R. Vinuesa. From coarse wall measurements to turbulent velocity fields through deep learning. *Physics of Fluids*, 33(7), July 2021.

[3] S. Laizet and N. Li. Incompact3d: a powerful tool to tackle turbulence problems with up to $o(10^5)$ computational cores. *International Journal for Numerical Methods in Fluids*, 67(11):1735–1757, 2011.