MSc thesis in Geomatics

Using deep learning to simulate wind in building area

Na Liu

June 2025

A thesis submitted to the Delft University of Technology in partial fulfillment of the requirements for the degree of Master of Science in Geomatics

The work in this thesis was carried out in the:

Delft University of Technology

Supervisors:	Dr. Azarakhsh Rafiee
	Dr. Frits de Prenter
Co-reader:	Shenglan Du

Abstract

Urban wind simulations are essential for assessing pedestrian comfort, pollutant dispersion, and microclimate design, yet high-fidelity CFD remains computationally expensive. This thesis investigates the potential of a Swin-Transformer-based surrogate model to approximate steady-state 2D results for 100 m × 100 m tiles under a single inflow direction. A dataset of 690 urban tiles was extracted from the 3DBAG and simulated in Ansys Fluent at 0.5 m resolution across five inflow speeds (5–15 m/s) and five types of building layouts. Velocity fields were rasterized to 1 m grids. The surrogate architecture preserves the original Swin backbone, evaluated with three loss variants: original RMSE, buffer weighting, and a divergence penalty. Encoding wind as xy-component consistently outperformed magnitude-only training. Errors scaled linearly with inflow speed and were highest in Mixed and Attached urban forms. Architectural resolution was the dominant factor influencing accuracy since it introduced visible artifacts, while the buffer and divergence losses offered only marginal improvements. Limitations remain near building facades, where sharp gradients are smoothed by the patch-based architecture. Nonetheless, the surrogate offers potential for rapid wind flow estimation suitable for early-stage design or preliminary analysis.

Acknowledgments

Looking back at the end of studying at TU Delft, I would like to express my sincere appreciation to all the people who have contributed not only to this thesis but also to my personal growth.

I would like to express my deepest thanks to my supervisor, Dr. Azarakhsh Rafiee, for her patient guidance, insightful feedback, and invaluable advice throughout this thesis. Her mentorship has significantly enhanced the quality of my work and deepened my understanding of the subject. I am also grateful to my other supervisor, Dr. Frits de Prenter, for his constructive suggestions, continuous support, and expertise. His insights have greatly contributed to the clarity and robustness of this thesis.

I would also like to express my heartfelt appreciation to my family and friends, whose understanding, encouragement, and unwavering support have sustained me throughout this journey. Their generosity far exceeds what I feel I deserve, and I often find myself at a loss for words to express the depth of my gratitude. I hope I can repay this kindness one day. Thank you.

Contents

List of Figures

List of Tables

Acronyms

CFD Computational Fluid Dynamics
ANN Artificial Neural Network
CNN Convolutional Neural Network
ViT Vision Transformer
GAN Generative Adversarial Network
PINN Physics-informed Neural Network
RANS Reynolds-averaged Navier–Stokes
LES Large Eddy Simulation
DNS Direct Numerical Simulation
ILES Implicit Large Eddy Simulation
UBL Urban Boundary Layer
FDM Finite Difference Method
FVM Finite Volume Method
FEM Finite Element Method
SIMPLE Semi-implicit Method for Pressure-linked Equations
RMSE Root Mean Squared Error
CAD Computer-aided Design

1 Introduction

1.1 Background and research motivation

Urban wind flow significantly affects a range of domains including building aerodynamics, pedestrian comfort, pollutant dispersion, and energy efficiency. These flow patterns are highly sensitive to physical characteristics of the built environment such as building geometry and spacing. These patterns can exhibit distinct behaviors, including creating eddies in streets perpendicular to the wind flow, accelerating along streets aligned with the wind, and diverging at street junctions which is discussed by BenMoshe et al. [2023]. Such patterns are intricately linked to the physical configuration of the urban settings, presenting a complex interplay between built structures and environmental forces.

Traditionally, Computational Fluid Dynamics simulations and wind tunnel experiments have been the standard tools for analyzing such flow behaviors. They are very accurate and widely adapted in various applications. However, under some circumstances challenges can still arise when these methods are applied. When simulating wind in urban environments, the main difficulty here is in two ways: firstly, the simulation domain for urban landscapes is typically larger, and secondly, it encompasses complex details of buildings and structures. To accurately capture these details, for CFD simulation, a finer mesh is required around buildings, which in turn leads to simulations that are both computationally demanding and time-intensive; for on-site measurements, the difficulties are similar. The installation of sensors requires a lot of effort, and it is also difficult to transfer the results between different cities. This complexity presents a significant bottleneck in terms of resource and time efficiency.

Beyond these challenges, the evolving landscape of technology has brought forth innovative solutions. The recent advancements in machine learning have open a new era in this field. These developments have paved the way for a variety of machine learning approaches aimed at streamlining the simulation process. Utilizing machine learning has the potential to expedite the process and to tackle the computational and time constraints inherent in traditional methods under certain circumstances. Kutz [2017] highlights the promising potential of deploying deep learning techniques in fluid dynamics.

Deep learning, a subset of machine learning, has shown impressive performance in areas such as natural language processing, computer vision, and time series prediction. For example, Convolutional Neural Networks (CNNs) have proven their excellent applications across various fields in GIS. In recent years, the rise of the Transformer model, a novel neural architecture, has garnered considerable interest, particularly in Natural Language Processing as mentioned by Vaswani et al. [2017]. Due to its ability to handle long-range dependencies, Zhao et al. [2021] has applied it to effectively capture spatial relationships in point cloud data. When simulating wind, a large number of non-uniform cells need to be calculated. This kind of vast, interrelated data is precisely where the strength of the Transformer model

lies. Yet, the application of such models in urban wind simulation is an area that remains underexplored. Considering the complex nature of the wind flow in urban area, some of the classic architectures used for image processing like convolutional neural network (CNN), though done excellent jobs in numerous areas like image classification, face recognition and image segmentation, has difficulties in handling dependencies between distant pixels due to limited receptive field. So in the most previous studies of using machine learning as surrogate models for fluid dynamics simulation, the focuses are usually to introduce the physical constraints into the loss function like PINN [Cai et al., 2021], or treating wind flow data as point-based data in time series to perform prediction. With the emergence of Transformer based architectures like ViT, TransGAN and Swin Transformer [Khan et al., 2021], it became possible to see wind flow as patterns in the space.

1.2 Scope

The central focus of this thesis is to explore the possibilities to apply the Swin Transformer architecture in urban wind flow simulation on micro-scale areas of $100m \times 100m$.

The research is specifically constrained to:

- Urban areas of $100m \times 100m$ micro-scale domains
- Steady-state wind flow conditions
- Wind flow patterns represent as 2D raster image

All the building footprints are obtained through real-world data, and processed through Ansys for wind simulations to generate training samples. The wind direction is maintained at a fixed orientation. Beyond evaluating the Swin Transformer's capability in wind flow simulation, this thesis aims to conduct detailed investigations into how wind speed variations and diverse building footprint configurations influence model performance, thereby systematically exploring the underlying factors affecting accuracy.

Given that the input data of model is 2D image, thus the height of the building is not considered. The influences of terrain and vegetation and the thermal interactions are also excluded. Though detailed turbulence modeling is not the main focus of this thesis, but turbulence are usually linked with building footprints, so the research aims to develop a foundational understanding of how geometric layouts influence wind flow patterns within the constraints of a 2D representation.

1.3 Research question

The research questions for which the thesis is oriented are stated in this section. There are two parts of the research questions including one main research question and several sub-questions.

To what extent can a Swin-Transformer-based surrogate accurately simulate wind fields in urban environments under specific initial conditions?

The goal of this study is to evaluate whether a Swin-Transformer-based surrogate can accurately reproduce steady-state urban wind fields. Based on the main research question, the sub-questions relevant to the thesis are:

- How do different urban building layouts affect the accuracy of the surrogate in simulating wind fields?
- How do varying inflow speeds affect the accuracy of the surrogate in urban wind field simulation?
- How does introducing a flow-aware loss function influences the surrogate's accuracy?

1.4 Outline

This thesis is organized as follows:

- **Chapter 2** provides the theoretical framework of the study, covering the fundamentals of CFD, particularly wind simulation in urban areas, and the principles of deep learning models, with a focus on the Swin Transformer architecture as well as reviewing related works relevant to this thesis.
- **Chapter 3** explains the conceptual choices: urban domain definition, mesh-resolution strategy, data processing, Swin-Transformer architecture and the evaluation methods.
- **Chapter 4** details the practical pipeline used to generate the dataset and run the experiments, including geometry generation, Ansys batch wind simulation, training hyperparameters.
- **Chapter 5** presents the results for different urban morphologies, inflow-speed variation, and the effect of the hybrid loss function design, followed by analysis.
- **Chapter 6** summarizes the main findings, states the key conclusions, discusses study limitations, and proposes future directions, including transient simulations, finer meshes, and alternative deep-learning frameworks.

2 Theoretical framework and related work

This chapter comprises two theoretical components of the thesis: computational fluid dynamics (CFD) and machine learning. The following diagram illustrates these concepts and shows their roles in the thesis.

2.1 CFD and wind dynamics

2.1.1 CFD

Computational fluid dynamics (CFD) is the science that uses numerical algorithms to simulate fluid flow, heat transfer and associated phenomena through computers [Versteeg and Malalasekera, 2007]. Together with pure theory and pure experiment, it helps people better understand fluid dynamics [Anderson, 1995]. There are three physical principles that determine the behaviors of fluid flow, which are: 1. Mass is conserved; 2. Newton's second law; 3. Energy is conserved.

From these conservation laws, comes the equations that govern all of fluid flows, they are continuity equation, momentum equation and energy equation [Anderson, 1995; Versteeg and Malalasekera, 2007].

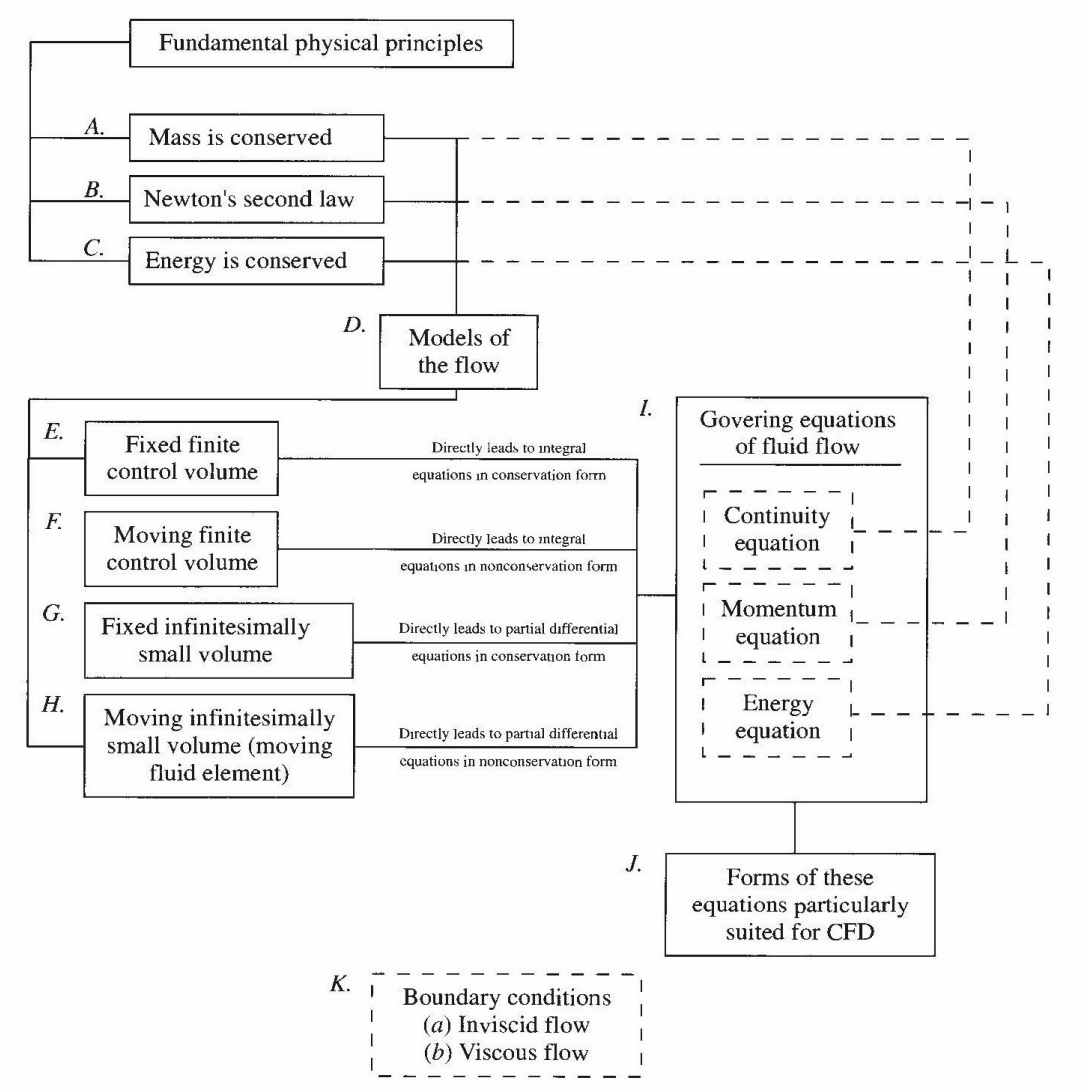


Figure 2.1: *Overview of the fundamental concepts in fluid dynamics [Anderson, 1995]*

Basic concepts

Fluid dynamics problems can be divided into categories based on how fluids behave when they flow, such as: ideal or viscous, steady or unsteady, compressible or incompressible, and laminar or turbulent. In CFD simulation, the type of fluid flow need first be determined to select the appropriate methods and equations. Often when studying a problem, only certain aspects of fluid flow require attention, while other minor factors are ignored. These classifications of fluid flow help highlight the problem and simplify the process.

The Reynolds number is a dimensionless quantities that used to describe the relationship between the inertia forces and viscous forces [LaNasa and Upp, 2014], which is defined as:

$$Re = \frac{Inertial\ Force}{Viscous\ Force} = \frac{\rho V D}{\mu} \tag{2.1}$$

where ρ is fluid density, V is velocity, D is diameter of the passage way and μ represents viscosity of the fluid. The Reynolds number gives the relative importance between inertial and viscous forces. When the Reynolds number is small, the flow is smooth and steady and is called laminar flow. On the other hand, if the Reynolds number is large, the flow would moves extremely irregularly, which is called turbulent flow[Versteeg and Malalasekera, 2007]. The critical point at which flow transitions from laminar flow to turbulent flow is called the critical Reynolds number.

Another nondimensional number that represents the effect of compressibility is called the Mach number. It is the ratio of the local fluid velocity to the local speed of sound [Sforza, 2012].

$$M = \frac{V}{c} \tag{2.2}$$

where V is the speed of the flow at any point and c is the sound speed at the same point. When $M < 0.3$, flow can be considered incompressible. Otherwise, the flow is compressible [Ferziger and Perić, 2002].

Numerical method

Previously, the conservation laws of mass, momentum and energy was mentioned. By integrating them into the same system, we can come up with a system of equations for an incompressible Newtonian fluid, called the Navier-Stokes equations [Blazek, 2001].

The Navier-Stokes equations take the fluid's viscosity into account. When the viscosity of the fluid is very small and can be ignored, the Navier-Stokes equations can be further simplified to Euler equations [Ferziger and Perić, 2002]. However, such equations are rarely solvable. Thus, engineers use numerical methods to simulate the fluid flow on computer, which is CFD. Numerical methods contains several key components including mathematical model, discretization method, grid, finite approximation, solution method and convergence criteria [Ferziger and Perić, 2002].

To find a numerical approximation to the solution, we replace differential equations into algebraic equations, which can be calculated, using a discretization method [Ferziger and Perić, 2002]. Time and space are discretized into small intervals, so that numerical solutions are given only on discrete points [Hirsch, 2007]. Commonly used discretization methods include finite difference method (FDM), finite volume method (FVM) and finite element method (FEM).

Turbulence models

Due to the random and chaotic nature of turbulent flow, it is very hard or even impossible to predict the irregular details of turbulence[Munson et al., 2013]. While Direct Numerical

Simulation (DNS) offers the most straightforward approach by directly solving the Navier-Stokes equations without additional modeling assumptions, it is not the most efficient approach. On most occasions, it is the effect of turbulence, rather than the specific details of turbulent fluctuations, that raises interest [Versteeg and Malalasekera, 2007]. This has led to the development of alternative approaches that employ various averaging techniques to make turbulent flow calculations more tractable. These methods can be categorized into Reynolds-Averaged Navier-Stokes (RANS), which averages the flow equations in time; Large Eddy Simulation (LES), which applies spatial filtering to resolve larger turbulent scales while modeling smaller ones; and Implicit Large Eddy Simulation (ILES), which relies on numerical dissipation to model subgrid-scale effects [Ferziger and Perić, 2002; Blazek, 2001].

Transient and steady flow

In a fluid system, the flow simulation can be characterized as either transient/unsteady or steady problem [Tu et al., 2018]. Transient flow occurs when the fluid properties at any point vary with time, making time a crucial parameter in the modeling process. Conversely, steady flow describes conditions where flow properties remain constant with time at each point in space. While transient flow is predominant in real-world applications, the additional time-dependent variables often increase computational complexity and make numerical solutions more challenging to converge [Munson et al., 2013]. In cases where the mean flow characteristics are of greater interest than instantaneous behavior, steady-state approximations can be employed to focus on the overall flow patterns. This approach simplifies the analysis while still providing valuable insights into the system's general behavior.

Grid

After developing the physical model, the computational domain must be discretized into elements where the governing equations will be solved. This discretization process creates a mesh or grid system that forms the foundation for numerical solutions. There are three rules applied to the mesh generated: complete coverage of the domain; no space in-between grids; no overlapping [Blazek, 2001]. The grid can take forms of various shapes, characterized by their features like structure, shape and arrangement [Moukalled et al., 2016]. These grids are generally classified into three main categories: structured grids, which follow a regular pattern with consistent connectivity; body-fitted grids, which conform to complex geometries while maintaining some structured characteristics; and unstructured grids, which offer maximum flexibility in element arrangement and shape [Tu et al., 2018].

Boundary conditions

The behavior of fluid flows can vary significantly under identical governing equations due to different boundary conditions [Anderson, 1995]. The specification of correct and appropriate boundary conditions is crucial for achieving accurate and robust numerical solutions in computational fluid dynamics [Blazek, 2001]. Common boundary condition types include solid walls, farfield boundaries for external flows, inlet/outlet conditions for internal flows, symmetry planes, coordinate cuts, periodic boundaries, and block interfaces [Blazek, 2001]. A similar but slightly different classification by Versteeg and Malalasekera [2007] identifies six primary categories: inlet, outlet, wall, prescribed pressure, symmetry, and periodic

boundary conditions. For urban wind flow simulations in two dimensions, the most critical boundary conditions are the wall conditions, which define building surfaces and their roughness characteristics, and the inlet/outlet conditions that specify the flow entry and exit locations.

2.1.2 Wind dynamics in urban area

Previously in this chapter, some of the basic concepts of CFD are covered, which are generally applicable for all kinds of fluid dynamic problems. For wind dynamics in urban area, these properties can be more specific. Wind flow in simulation is usually considered as viscous, incompressible flow. The interactions between atmosphere and the urban structures create complex patterns and turbulence. Aside from the turbulence created by the rough surfaces of ground and buildings, the shape of the air flow areas also has great impact on the velocity of the wind. For example, wind would accelerate when going though a narrow alley, and slows down when air way gets wider. Wind would also change directions when hitting the wall, reflected on the change of velocity magnitude on orthogonal directions.

Turbulence in urban environments

In urban boundary layer (UBL), the flow can be characterized on horizontal scales into street (10-100 m), neighborhood (100-1000 m) and city (10-20 km) based on urban morphology [Barlow, 2014]. There are two main types of approaches for urban environments: RANS approach; LES and DNS approach [Li et al., 2006]. When modeling on 2D level where the heights of the buildings are not considered, RANS, especially the k-ϵ and its variants are widely accepted for their computational efficiency and ability to provide reliable results in urban flow simulations [Blocken, 2018].

Aspects to be evaluated

Thermal comfort, wind comfort, building loads, and extreme wind event safety are key considerations when evaluating wind effects in urban areas. This evaluation typically involves two main aspects: the impact on pedestrian comfort and the pressure exerted on buildings. For pedestrian comfort, factors such as thermal conditions, humidity, solar radiation, precipitation, and wind speed play crucial roles. However, wind speed is often the main focus when assessing wind comfort [Blocken et al., 2016]. Various criteria exist to evaluate wind comfort, often based on mean wind speeds or gust wind speeds [Holger Koss, 2006]. In the Netherlands, time-mean wind speeds are commonly used to define wind comfort. Specifically, a pedestrian-level wind speed exceeding $V > 5\,\mathrm{m/s}$ is used as a threshold for discomfort, while $V > 15\,\mathrm{m/s}$ is considered dangerous for pedestrians [Willemsen and Wisse, 2007].

2.2 Deep learning

Machine Learning Models, especially deep learning models, have been proven excellent at pattern recognition and handling large datasets. Deep learning models utilize different

structured neural networks to learn local and global features of data, enable their ability to tackle complex tasks like natural language processing, image classification, object detection. Neural network is a kind of network that simulate how brain cells in human brain works. Figure 2.2 shows a classic fully Artificial neural network (ANN). Through backpropagation algorithm and gradient descent, models are able to find relatively optimal parameters, which is also how model "learns".

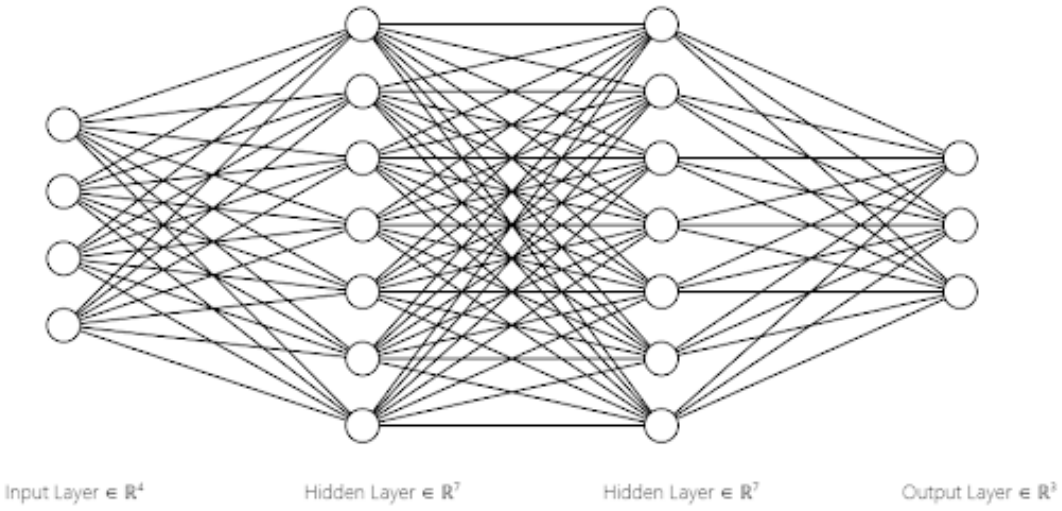


Figure 2.2: *Neural network*

2.2.1 Deep learning-based techniques for computer vision

Computer vision is a field that includes both traditional and deep learning-based techniques for extracting meaningful information from digitally represented visual inputs. One of the dominant architectures for computer vision tasks is convolutional neural networks (CNN).

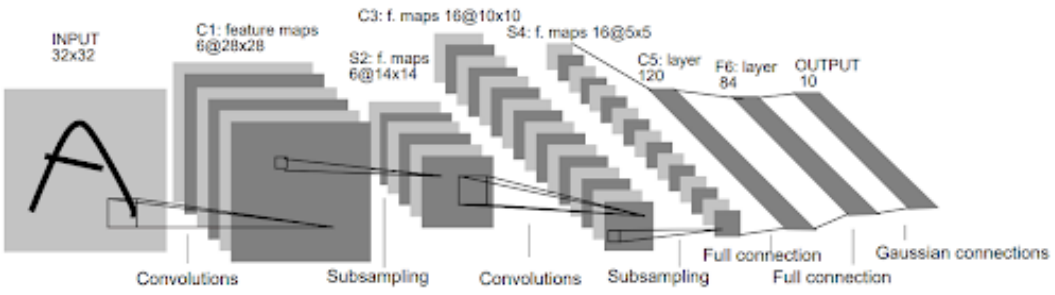


Figure 2.3: *An architecture of CNN from LeCun et al. [1998]*

CNNs perform convolution using kernels, to abstract the feature of the input, making them highly effective for computer vision tasks. When CNN processes high-dimensional data, such as images, it can significantly reduce the number of parameters compared to traditional ANN, which makes training easier [O'Shea and Nash, 2015]. At the same time, the existence of convolutional layers and pooling layers helps reduce the risk of overfitting. These advantages allow CNNs to thrive in computer vision fields such as target detection, image classification, image segmentation, and other fields. However, CNNs also have limitations in capturing long-range dependencies and understanding global context.

In 2017, a new model called Transformer Vaswani et al. [2017] based on a self-attention mechanism was proposed.

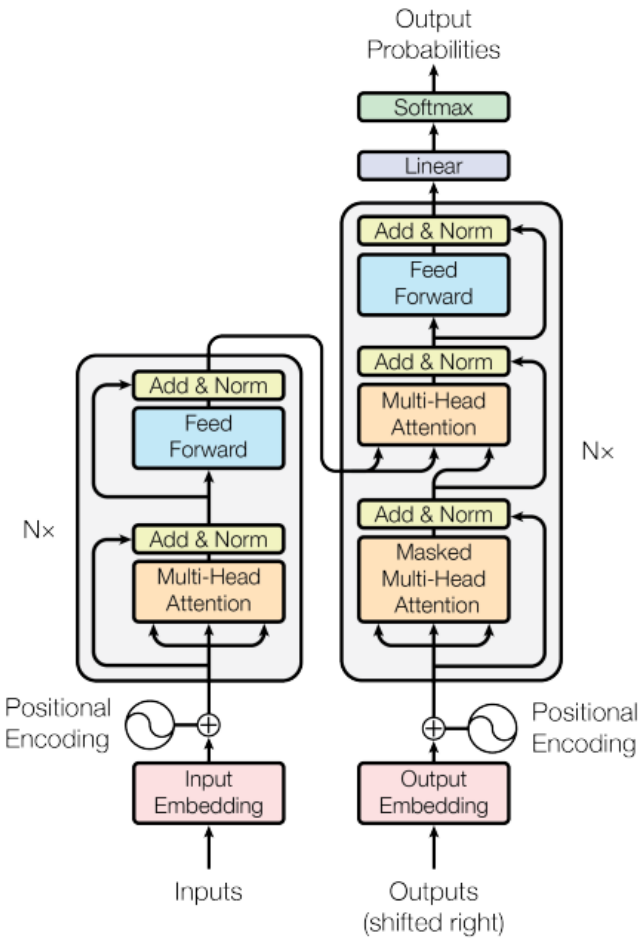


Figure 2.4: *The architecture of the Transformer from Vaswani et al. [2017]*

Transformer was originally designed for natural language processing (NLP), containing two key components, self-attention and positional encoding.

The query, key and value of the self-attention mechanism all come from the input sequence. Calculations are implemented through matrix operations. As shown in the left picture of

Figure 2.4, the self-attention score in Transformer is calculated through dot product. Its main function is to learn the dependencies within the sequence and obtain the attention weight of each position and other positions.

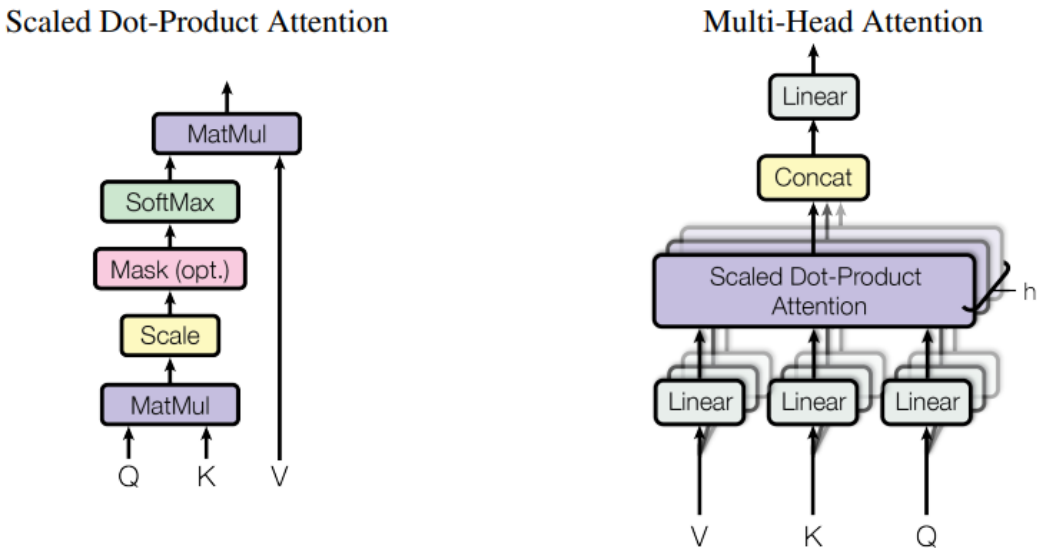


Figure 2.5: *[left] Scaled Dot-Product Attention. [right] Multi-Head Attention consists of several attention layers running in parallel from Vaswani et al. [2017]*

Transformer quickly gained widespread application in the field of natural language processing, demonstrating its superior performance over existing models. Considering positional encoding, which is, in fact, encoding the spatial relationship of content, it is easy to think of the potential application of the Transformer model in computer vision. However, given that images typically contain far more pixels than a sentence contains words, applying Transformers to image processing is not as straightforward. The volume of data in images poses significant challenges for direct implementation.

It wasn't until 2020 that Dosovitskiy et al. [2021] proposed a model that directly applies the standard Transformer to images, known as the Vision Transformer (ViT). ViT divides images into patches of equal size and incorporates position embedding into the patch embedding to preserve the positional information of pixels.

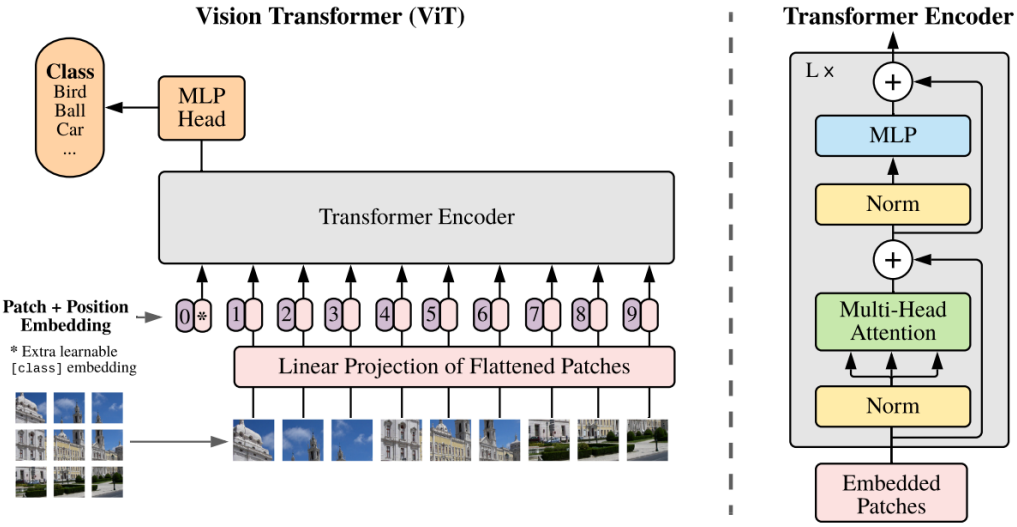


Figure 2.6: *The architecture of Vision Transformer from Dosovitskiy et al. [2021]*

The advantage of ViT, compared with traditional convolution, is that it can integrate the information of the entire image by using self-attention. With relatively cheap pre-training cost, its performance is not inferior to or even exceeds the best models at the time [Dosovitskiy et al., 2021]. But since ViT looks for relevant context by calculating the attention score between each patch and all other patches in the image, its computational complexity increases exponentially. Meanwhile, it lacks the ability to extract features at different scales, making it difficult to apply it to high-resolution images and tasks on pixel level.

2.2.2 Dense prediction

Dense prediction is a type of computer vision task. Common dense prediction tasks include image segmentation, edge detection, and depth prediction. Unlike image classification, it requires specifying a label or value for each pixel of the image. This brings two challenges, one is the cost of labeling the training set, and the other is the capability and efficiency of the model itself at the pixel level.

2.3 Swin Transformer

Swin Transformer is a cutting-edge model proposed by Liu et al. [2021] in 2021 based on ViT. The use of shifted window in their new model, allows Swin Transformer to perform very well on computer vision tasks such as target detection and image segmentation while having linear computational complexity relative to image size.

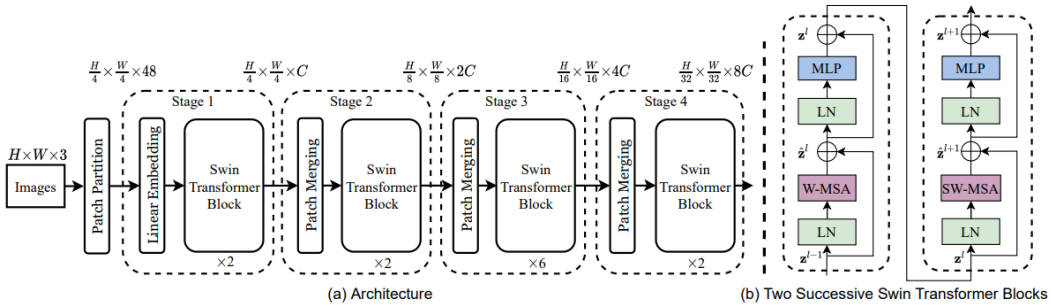


Figure 2.7: *The architecture of a Swin Transformer (Swin-T) from Liu et al. [2021]*

2.3.1 Shifted window

Figure 2.7 shows the local window in Swin Transformer and the window shifting process. The red boxes represent the range of self-attention calculation, called local windows. After the calculation is completed, the windows on the next layer (right in Figure 2.7) is shifted. The new windows encompasses the boundaries of the original windows, thus providing connections between them.

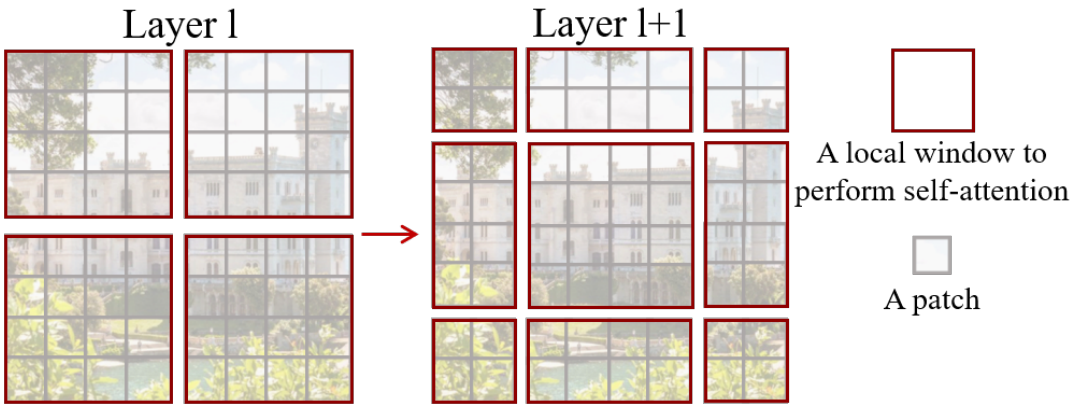


Figure 2.8: *An illustration of the shifted window from Liu et al. [2021]*

The shift of the window introduces a challenge: the change of the window size. As can be seen from Figure 2.7, the original window size is uniform, but after shifting, some windows of different sizes are produced. To address this, Liu et al. [2021] proposed a more efficient batch computation approach by cyclic-shifting toward the top-left direction. They pad the smaller windows so that the size and number of windows remain uniform. After cyclic shifting, masking is applied to restrict self-attention calculations within the shifted windows to only the originally adjacent patches in the image.

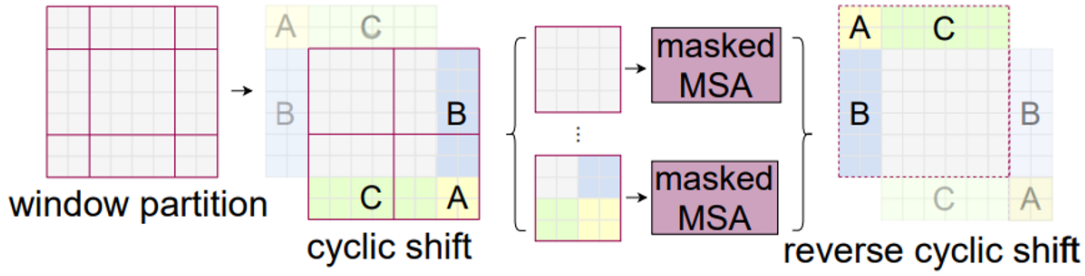


Figure 2.9: *Illustration of an efficient batch computation approach from Liu et al. [2021]*

2.3.2 Patch merging

In the patch merging layer, the features of adjacent 2×2 windows are merged, so the resolution becomes half of the original. In the Swin-T architecture (Figure 2.6), three patch merging layers are employed. As shown in Figure 2.9, each patch merging expands the image area covered by a single window. Patch merging enables Swin Transformer to extract features hierarchically, ranging from local details to global context.

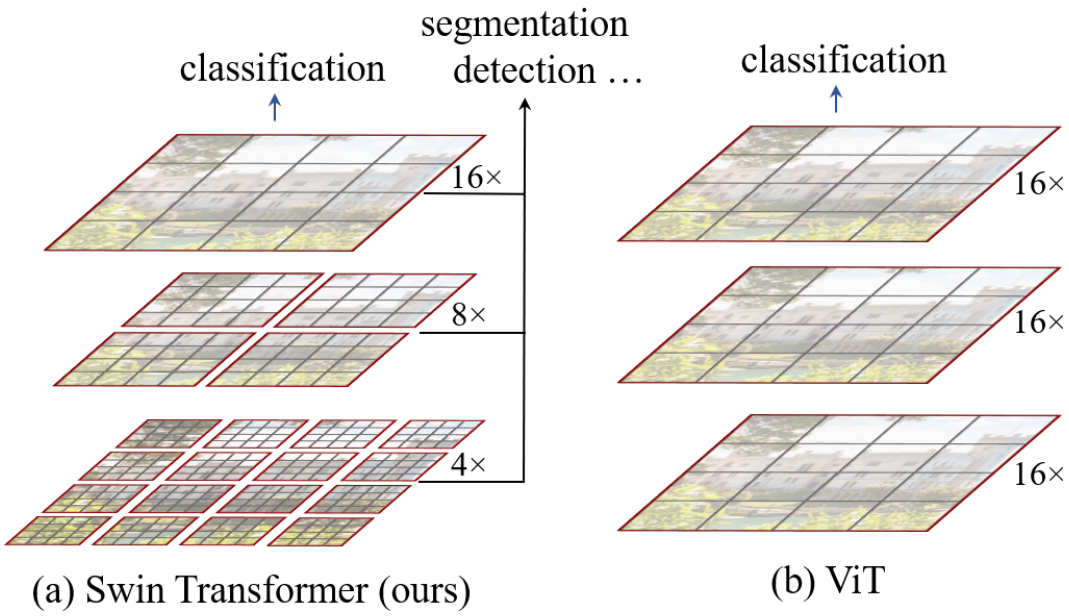


Figure 2.10: *Hierarchical feature map from Liu et al. [2021]*

2.4 Related work

Numerous studies have examined wind interactions within urban building areas, primarily focusing on two aspects: the geometry patterns of urban areas and their corresponding flow patterns. The application of deep learning in computational fluid dynamics (CFD) within urban contexts has become increasingly prevalent. Approaches involving deep learning algorithms diverge in two main directions. The first approach involves utilizing deep learning models to generate 3D building structures from varied data sources, such as point clouds, which are then suitable for traditional CFD simulations as the work done in Sun et al. [2021]. The second approach, and the one this thesis aligns with, seeks to expedite the simulation process without directly solving the Navier-Stokes equations' numerical solution but rather using deep learning algorithms like GAN in Kastner and Dogan [2023] and White et al. [2019].

Various models have been applied to simulate wind flow in urban environments. Among these, Physics-Informed Neural Networks (PINNs) are closely tied to the original Navier-Stokes equations. What sets PINNs apart is their incorporation of these equations directly into the loss function, a concept crucial in deep learning for measuring the discrepancy between model predictions and actual data during training. The model then adjusts its parameters based on this loss function through back-propagation. By integrating the Navier-Stokes equations into the loss function, PINNs lend physical significance to the optimization process, ensuring that the learned models adhere to fundamental fluid dynamics principles.

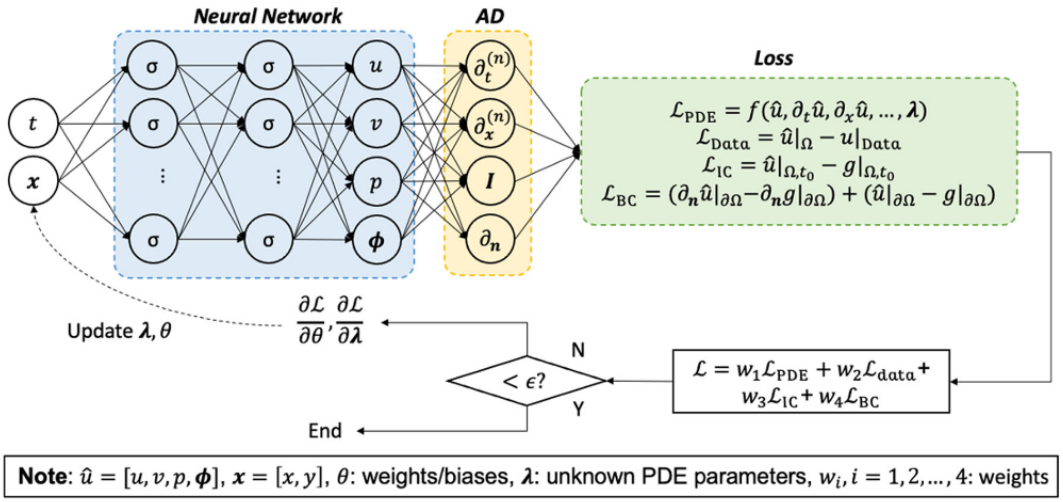


Figure 2.11: *Schematic of a physics-informed neural network (PINN) in Cai et al. [2021]*

However, PINNs face certain challenges, as noted by Cai et al. [2021], these challenges include the high-dimensional, non-convex nature of the loss function, which is a common hurdle in neural network optimization, and the potential for generalization and optimization errors depending on training data and optimizer choices. Despite these challenges, PINNs offer a novel and promising approach to fluid dynamics simulations, particularly for complex scenarios where traditional methods may fall short. They are especially adept at integrating scattered, partial spatio-temporal data, making them suitable for a range of fluid

mechanics problems, including urban wind simulation.

Convolutional Neural Networks (CNNs) are also broadly used for CFD in wind simulation. Typically associated with computer vision, CNNs utilize kernels to extract hidden patterns from images. Over the years, CNNs have evolved, with variants like 3D CNNs extending their application to three-dimensional spaces. Miyanawala and Jaiman [2018] demonstrate an efficient deep learning technique for the Navier-Stokes Equations, applying CNNs to predict unsteady wake flow dynamics with notable computational efficiency. Similarly, Guo et al. [2016] explore CNNs for steady flow approximation, highlighting their capability to process complex fluid dynamics data. These developments underscore the versatility of CNNs in handling spatial data, making them particularly suitable for the multifaceted challenges in urban wind simulation.

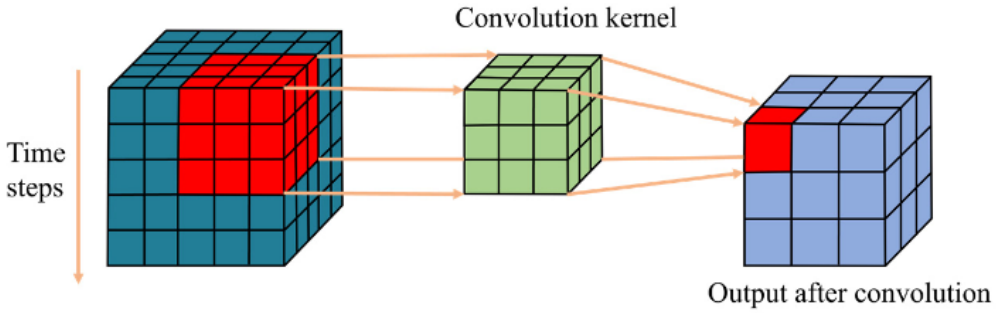


Figure 2.12: *3D convolution kernel in Huang et al. [2023]*

K-Nearest Neighbors (kNN) is another notable method employed for simulating wind in urban environments. Differing from models that require the entire building area scheme, kNN operates on a database of cells encoded with their geometrical surroundings. When tasked with predicting wind speed and pressure on a new cell, the model identifies the 'k' most similar cells from this database and calculates the value based on these known outcomes. BenMoshe et al. [2023] illustrate that this approach, despite its straightforward nature and ease of understanding, hinges critically on the quality and abundance of the pre-existing data. If the database lacks analogous data, the prediction accuracy may suffer. Additionally, the selection of the 'k' value is subjective and depends on the user, introducing a layer of variability in the model's application.

3 Methodology

This chapter outlines the methodological framework adopted in this research. It details the selection of turbulence models, numerical schemes, and boundary conditions used in the simulations. Additionally, it presents the early-stage parameter testing conducted to evaluate the sensitivity of key variables, ensuring the accuracy and reliability of the final simulation setup.

3.1 Wind simulation

3.1.1 Urban geometry and boundary design

The methodology begins with dataset creation, a crucial prerequisite for model training due to the absence of suitable existing datasets. The first step is to obtain the urban building characteristics of The Hague. 3DBAG by 3D geoinformation research group (TU Delft) and 3DGI [Peters et al., 2022] is the most detailed, open dataset on buildings in the Netherlands. The 3DBAG dataset provides 3D models of buildings along with 2D projections of rooftop surfaces. It excludes underground structures and overlapping buildings, making it perfect as the base data for wind simulation in 2D.

Once the 2D building footprints are obtained, the software used for meshing and urban wind simulation is Ansys Fluent. This is a well-established commercial software for fluid simulation, widely applied in academia and industry. Ansys offers the advantage of a full workflow from initial geometry setup to final simulation, with scripting capabilities for automation. Given the computational demands of model training, particularly for data-intensive architectures such as the Swin Transformer, the automation of sample generation becomes critically important. While OpenFOAM is also an excellent open-source CFD software, it does require more manual setup for batch processing. Ansys Fluent was selected not only for its extensive credits in urban wind simulation, but also because its built-in meshing tools and scripting capabilities, enables the possibilities of automating the simulation to the largest extent.

For turbulence modeling, the k-ϵ and SIMPLE algorithm were chosen, as they are widely used for their balance between computational efficiency and accuracy in steady-state flow simulations. To maintain consistency across all simulations, the wind inlet is set at the lower boundary of the domain for each simulation, while the outlet is set at the upper boundary. A 50 m margin is applied at both the upper and lower boundaries when defining the wind flow domain, helping minimize the boundary effects. This setup ensures a uniform approach to boundary conditions, which is critical for obtaining comparable results in urban wind flow studies.

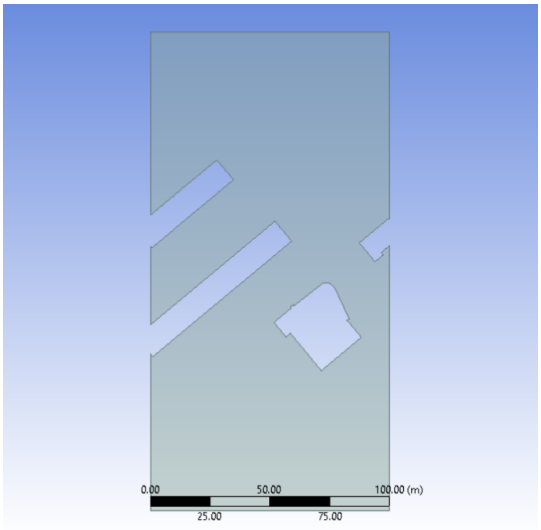


Figure 3.1: *Example of a wind flow domain in DesignModeler.*

3.1.2 Meshing: resolution, quality, convergence

When setting the mesh resolution, there are two elements that need to be considered: time cost and the accuracy of the result. A finer mesh generally improves accuracy but increases computational cost. Conversely, if the mesh resolution is too coarse, the simulation could fail to capture flow details accurately, potentially leading to divergence or erroneous results. The size of the grid in mesh is usually determined based on the specific research question. And the general practice of setting the resolution is to first set a coarser mesh based on empirical guess, then gradually refine the mesh and compare the results of the simulation to see if there are any significant differences that cannot be ignored. Additionally, mesh quality metrics, such as orthogonality and skewness are also commonly used as criteria for mesh quality evaluation. The following table presents the impact of mesh resolution on computational cost and quality metrics, using one of the geometries from the dataset.

Element Size (m)	**Time Cost (s)**	**Min Orthogonal Quality**	**Max Skewness**
1.0	12	0.5111	0.74521
0.7	25	0.5111	0.64122
0.5	55	0.2681	0.79022
0.3	201	0.5111	0.73375

Table 3.1: *Mesh element size impact on computational metrics*

According to *Ansys Fluent User Guide* [ANSYS, Inc., 2009], the maximum skewness should be kept below 0.95 in most cases, while the minimum orthogonal quality should be greater than

0.15. Based on the results presented in Table 3.1, all tested meshes meet these criteria, indicating that mesh quality is acceptable. The primary concerns, therefore, are computational efficiency and suitability for rasterization sampling.

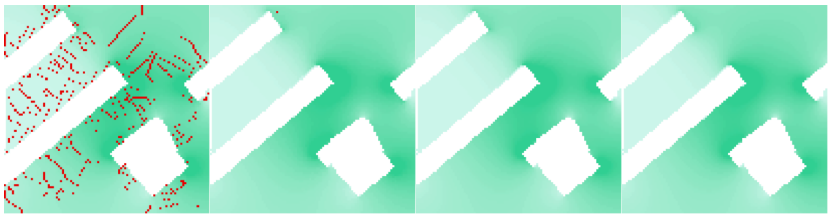


Figure 3.2: *Wind simulation results using different mesh element sizes, where the red pixels represent no-data regions.*

To convert point data into raster images with a resolution of 1 $m \times 1$ m, the mesh element size must be smaller than 1 m to ensure that sufficient data points are available for sampling within each grid cell. As shown in Figure 3.2, when the element size is exactly 1 m, many NoData pixels appear in the image due to insufficient sampling points. Even with an element size of 0.7 m, some NoData pixels persist, indicating the need for a finer mesh resolution to improve data coverage and accuracy.

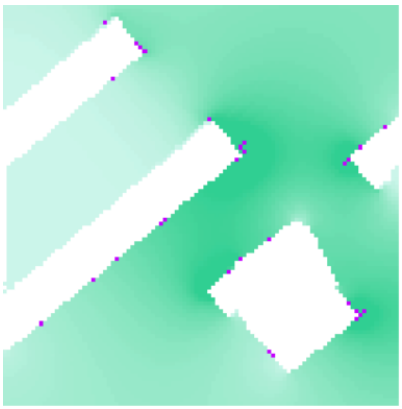


Figure 3.3: *Comparison of wind simulation results for element sizes of 0.5 m and 0.3 m under a wind inflow of 10 m/s. Purple pixels indicate velocity differences greater than 2 m/s.*

Figure 3.3 illustrates the difference in simulation results when the element size is set to 0.5 m and 0.3 m under a wind inflow speed of 10 m/s. The purple pixels highlight regions where the velocity difference exceeds 2 m/s, which primarily occur adjacent to building structures. Despite these localized deviations, the overall variation remains within an acceptable range.

Considering these factors, an element size of 0.5 m is chosen as the optimal balance between accuracy and computational efficiency.

3.2 Data

3.2.1 Data normalization

When it comes to wind speed data, the values can spread to a wide range and the vast majority of the pixel values are concentrate near the mean value.

Different inflow velocities can also have an impact on the distribution. As can been seen in the table below, with the initial wind speed growing, the standard deviation of the data also increases, indicating greater variability and dispersion. This increased complexity presents a greater challenge for the machine learning model to learn the pattern. A thorough experiment is required to review how these variations affect the model's predictive accuracy and generalization capability. Understanding this impact is crucial for ensuring the model's robustness across different wind conditions.

Inflow (m/s)	**Direction**	**Min**	**Max**	**Mean**	**Std**
5.0	x	-39.97	33.99	-0.15	2.81
	y	-20.32	68.66	5.98	3.22
	mag	0.00	75.25	6.55	3.36
7.5	x	-59.95	50.94	-0.23	4.22
	y	-30.49	102.97	8.97	4.84
	mag	0.00	112.90	9.82	5.03
10.0	x	-79.94	67.97	-0.31	5.63
	y	-40.64	137.33	11.96	6.45
	mag	0.00	150.48	13.09	6.45
12.5	x	-99.89	84.93	-0.38	7.03
	y	-50.79	171.64	14.95	8.06
	mag	0.00	188.20	16.36	8.39
15.0	x	-119.90	101.89	-0.46	8.44
	y	-60.98	205.93	17.94	9.67
	mag	0.00	225.79	19.63	10.07

Table 3.2: *Statistical data for wind speed.*

In order to better combine x-direction wind and y-direction wind data, and also better compare the results across the datasets, it is necessary to normalize the data. The datasets for different wind speeds are normalized by the following equation:

$$X_{\text{normalized}} = \frac{X - \min(X)}{\max(X) - \min(X)} \tag{3.1}$$

Where $\min(X)$ and $\max(X)$ are minimum and maximum value of x. After the normalization, both x-direction data and y-direction data are in the range of [0, 1].

3.2.2 Training and validation set

The training and validation samples are from the same dataset, which is randomly split into training and validation subsets. To ensure consistency across training sessions, a fixed random seed of 42 is employed during the dataset partitioning process.

3.2.3 Test set

To test how different building layouts influence the performance of the deep learning model, tests were designed to examine two aspects: building shapes and building spacing.

- **Building Shape**: The geometry of individual buildings primarily affects turbulence near the structures.
- **Building Spacing**: The distance between buildings influences both the velocity of the wind and the characteristics of the turbulence.

Given that urban environments are often non-uniform building clusters, it is hard to find areas with consistent layouts. However, building features are also linked to their functions. For example, apartment buildings in the Netherlands often appear as long, narrow rectangles with multiple units attached together on 2D maps, while industrial buildings typically appear as independent large polygons. To systematically evaluate the influence of urban building morphology on wind flows, test samples were grouped into five categories based on their geometric and functional features:

- **Attached**: Continuous residential structures.
- **Detached**: Standalone buildings.
- **High-rise**: Skyscrapers and tall commercial buildings.
- **Industrial**: Large warehouse-style buildings.
- **Mixed**: Areas containing a combination of different building forms.

In addition to the five categorized groups mentioned above, a larger test dataset contains uncategorized samples is included to evaluate the model's performance on a broader range of urban layouts.

Dataset Category	Number of Samples
Training and Validation Set	690
Test Set	
Uncategorized	60
Attached	22
Detached	30
High-Rise	17
Industrial	30
Mixed	26

Table 3.3: *Number of samples in training and test sets*

3.3 Swin Transformer

The Swin Transformer model used in this study is adapted based on the architecture proposed by Liu et al. [2021]. While the backbone of the model is remains the same with the original model, some modifications are introduced in order to fit the model with the data and wind flow simulation, including dense prediction head and a customized loss function to optimize pixel-wise predictions. Though Swin Transformer primarily serves for conventional computer vision tasks like classification or object detection, this study applies the model to dense prediction that requires pixel-level numerical outputs instead of labels. Therefore, the model is modified to support per-pixel regression.

3.3.1 Dense prediction head

The dense prediction head used Conv2d layers to reduce the number of channels from the feature maps followed by LeakyReLU activation function to introduce non-linearity and prevent dead neurons. ConvTransposed2d layers for upsampling. An AdaptiveAvgPool2d layer to ensure the size of the outputs and a Conv2d layer in the end to generate the required number of output channels such as magnitude or x-velocity and y-velocity.

3.3.2 Evaluation

RMSE

The primary loss of the model is evaluated by Root Mean Squared Error (RMSE) for its ability to penalize large errors more heavily without sacrificing too much on overall accuracy. When calculating losses, all the pixels that represent buildings are excluded so that only the wind flow zones can contribute to the optimization. The RMSE is calculated as:

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2} \tag{3.2}$$

where $\hat{y}_i$ is the predicted wind velocity, y_i is the ground truth, and N represents the total number of non-building pixels.

Buffering

In urban wind simulation, the main factor that influence the wind and introducing chaotic turbulence is the building. And the hardest parts to predict is also where are closest to the buildings. In some preliminary tests, the model showed high accuracy in the areas far from buildings, but struggled to capture details in the areas adjacent to the buildings.

Figure 3.4: *Illustration of the buffered area, where the blue pixels represent the high-weighted parts.*

To emphasize the importance of near-building area, a buffered mask was apply to give these areas more weights while training. Buffering was done by expanding the pixels that represent buildings and walls. By applying a weight on the areas that near the architectures, the loss is calculated by:

$$\mathcal{L} = \text{Outer RMSE} + w \cdot \text{Inner RMSE} \tag{3.3}$$

Where w is a scaling factor that amplifies near-building errors.

Divergence

A fundamental constraint in steady flow simulation is mass conservation, which means that the result should be fully converged and the total air entering a region equals the total air exiting it. Since divergence requires the directions of the wind to be calculated, this

constraint would only be applied when model is trained with both x-velocity and y-velocity. Divergence is added in the loss function as an additional loss term, and is computed as:

$$\text{Divergence} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = \frac{u_{i+1,j} - u_{i,j}}{h} + \frac{v_{i,j+1} - v_{i,j}}{h}$$

where u and v are the velocity components in the x and y directions, respectively. And h is the size of the grid, which in this study is set to 1.

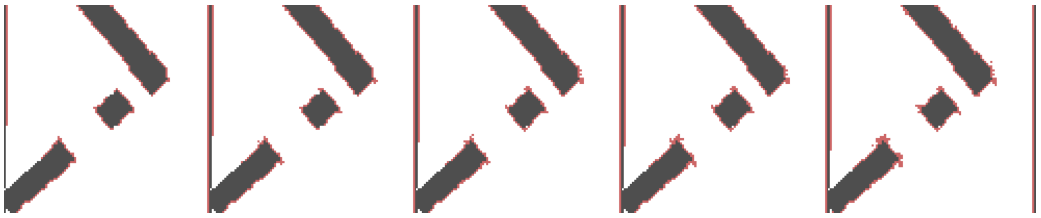


Figure 3.5: *Visualizations of divergence magnitude under different wind inflow speeds. Red pixels indicate regions where the absolute divergence is higher than 1.*

Although convergence is usually monitored by residuals during the CFD simulation, in order to visualize the convergence more and validate the divergence calculation procedure, figure 3.5 presents examples of divergence maps calculated from CFD results under different inflow speeds. Due to the process of resampling from point-based CFD results to raster images, divergence may appear to be higher. Notably, there are some pixels around buildings show a very high divergence, This phenomenon arises not only because the turbulence near buildings, but also from when calculating divergence, the values of building pixels would considered to be 0, causing the results around buildings to be discontinuities and inaccurate.

4 Implementation

This chapter lists practical steps for dataset generation for model training and fine-tuning preparations.

4.1 Dataset generation

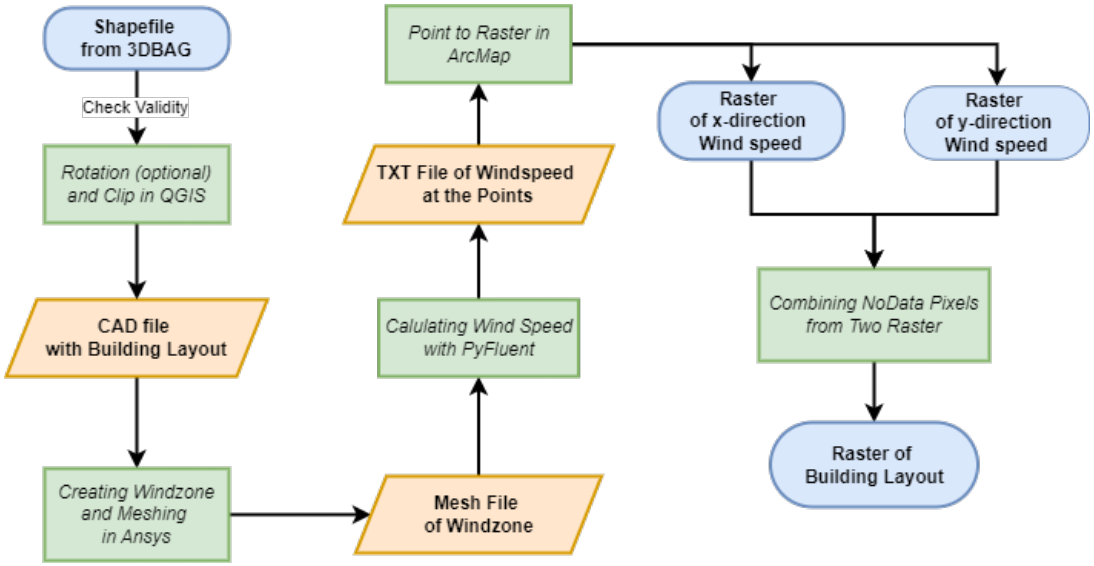


Figure 4.1: *Flowchart for dataset generation*

4.1.1 Building fingerprints generation

The dataset on buildings in the Hague is obtained from **3DBAG.nl** in shapefile format. The shapes of buildings are represented by The shape of the building is represented by 2D vector polygons, defining its footprint.

Figure 4.2: *Original buildings in the shapefile.*

After obtaining the shapefile, a validation check is performed using **QGIS** to ensure that all building polygons do not contain invalid geometries.

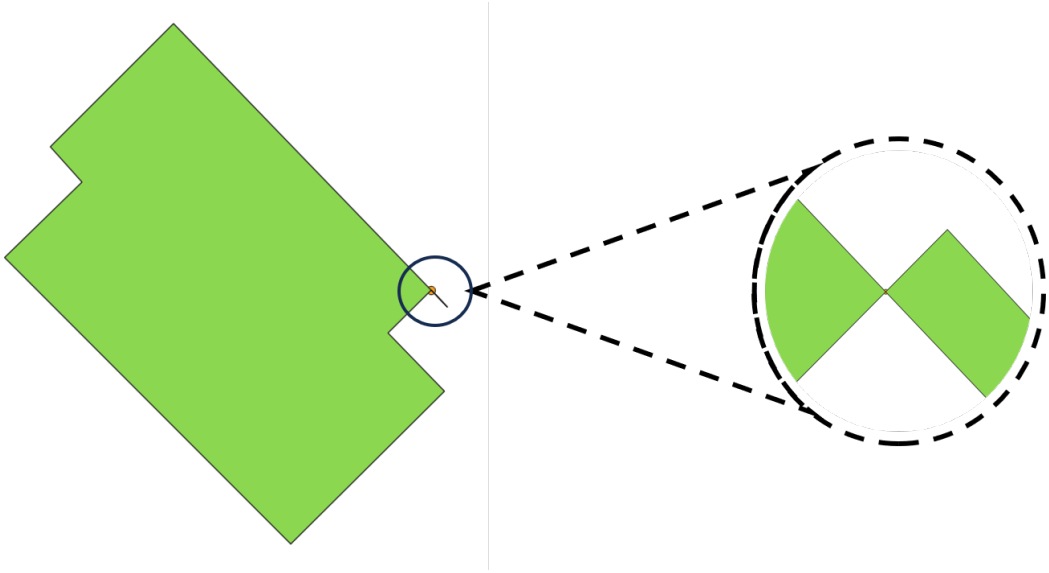


Figure 4.3: *Example of invalid building polygons detected during pre-processing.*

In order to keep the sample size uniform for wind simulation, the shapefile is split into tiles

of size $100m \times 100m$ using a Python script.

Figure 4.4: *Grid structure used to partition the study area into* $100m \times 100m$ *tiles.*

For simplicity, adjacent polygons are merged. Additionally, since buildings smaller than 100 m^2 are excluded in this step as they are too small in size and not representative enough. With $100m \times 100m$ tiles, further manual visual inspection is performed to filter out the tiles where proper air circulation is not possible. These tiles are excluded before exporting the dataset to CAD format for CFD simulation. The script for this step is listed in Appendix 1.

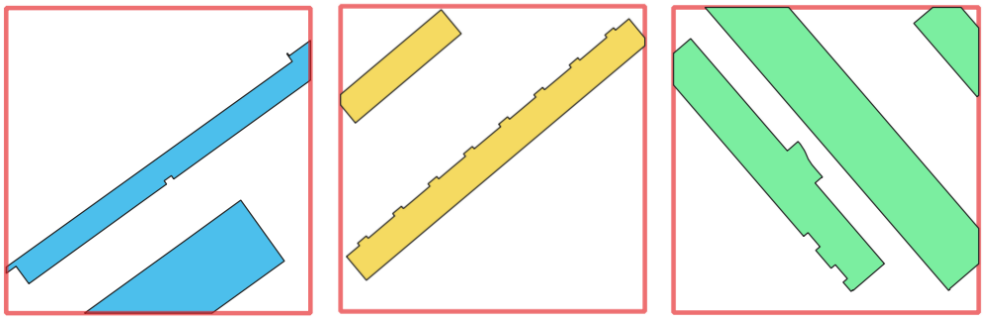


Figure 4.5: *Example of tiles excluded due to improper wind flow zone.*

4.1.2 Operational workflow for Ansys batch runs

The wind simulation is conducted using Ansys and consists of three key steps:

1. Wind zone definition:
    - The wind zone that needs to be simulated is created in Ansys **DesignModeler**.
    - The domain is extended by 50 m in both sides in the vertical direction, so that uniform airflow can be ensured.
    - In the vertical direction, the lower boundary is set as inlet, while the upper boundary is set as the outlet.
2. Mesh generation
    - The mesh is created by **Ansys Meshing**.
    - The grid size is set to 0.5 m.
3. Wind flow simulation
    - The wind simulation is done with **Ansys Fluent** accessed through **Ansys PyFluent**.
    - The k-epsilon turbulence model and the SIMPLE solver are applied.
    - The simulation is run for 2000 iterations.
    - The simulation outputs velocity magnitude as well as its components in the x-direction and y-direction at each sampled point, stored in TXT format as shown in the Figure 4.6.

```
nodenumber,    x-coordinate,    y-coordinate,      x-velocity,      y-velocity,velocity-magnitude
         1, 4.745040894E+01, 1.076950760E+02,-4.606513023E+00, 1.130503845E+01, 1.220760059E+01
         2, 4.744303513E+01, 1.071796341E+02,-4.676264763E+00, 1.135389900E+01, 1.227925682E+01
         3, 4.793317795E+01, 1.071626434E+02,-4.630705833E+00, 1.142044640E+01, 1.232362175E+01
         4, 4.794179916E+01, 1.076788635E+02,-4.561842918E+00, 1.137207317E+01, 1.225300312E+01
         5, 4.695777893E+01, 1.077105255E+02,-4.651576996E+00, 1.123671818E+01, 1.216152668E+01
         6, 4.695264053E+01, 1.071958923E+02,-4.722319603E+00, 1.128634548E+01, 1.223452473E+01
         7, 2.550842094E+01, 9.358595848E+00, 3.441128016E+00, 9.511457443E+00, 1.011491585E+01
         8, 2.501681137E+01, 9.365110397E+00, 3.460210800E+00, 9.420621872E+00, 1.003611469E+01
         9, 2.507768440E+01, 8.878129959E+00, 3.369159698E+00, 9.417509079E+00, 1.000214577E+01
        10, 2.557164955E+01, 8.873749733E+00, 3.351234198E+00, 9.505628586E+00, 1.007918358E+01
```

Figure 4.6: *Example of simulation output stored in TXT format.*

4.1.3 Point-to-Raster conversion

After the CFD simulation, the velocity field data stored in TXT format is converted into a raster grid representation using **ArcMap**. The velocity magnitude, x-velocity and y-velocity are saved as three separate TIFF files. The building masks, which are later used as input images, are generated by identifying pixels with a value equal to 0 in any of the previously mentioned TIFF files. Building pixels are assigned a value of 0, whereas non-building pixels are assigned a value of 1, indicating areas where model is required to make predictions.

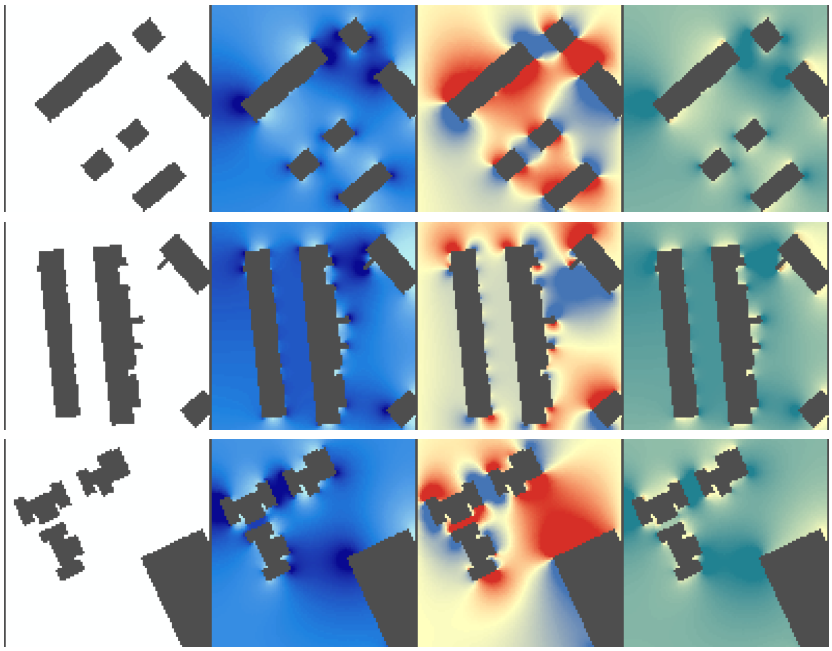


Figure 4.7: *Example of building masks and corresponding wind velocity rasters.*

4.2 Hyper-parameter tuning

In this study, no pretrained weights are employed and all the models are trained from random initialization. To approximate the best performance of the Swin Transformer model, a series of tuning experiments for hyperparameters have been done, aiming to find the best settings for window size, patch size, buffer range and weights. Due to computational constraints and efficiency consideration, the fine-tuning process was conducted on a small dataset consisting of 100 wind simulation images with an inflow speed of 10 m/s. The dataset was split into 85 training images and 15 validation images.

4.2.1 Window size and patch size

Considering how and where attention is calculated in Swin Transformer, the window size and patch size significantly impact the performance of the Swin Transformer model especially when for dense prediction tasks. Given the size of the images used in this study is different from those in the original Swin Transformer paper or most of the studies using Swin Transformer, it is necessary to do some experiments to determine the optimal setting of patch size and window size. The possible configurations are brought under the idea that the size of the image can be divisible by the combination of window size and patch size. Configurations with a patch size of 1 were excluded due to excessive GPU memory requirements.

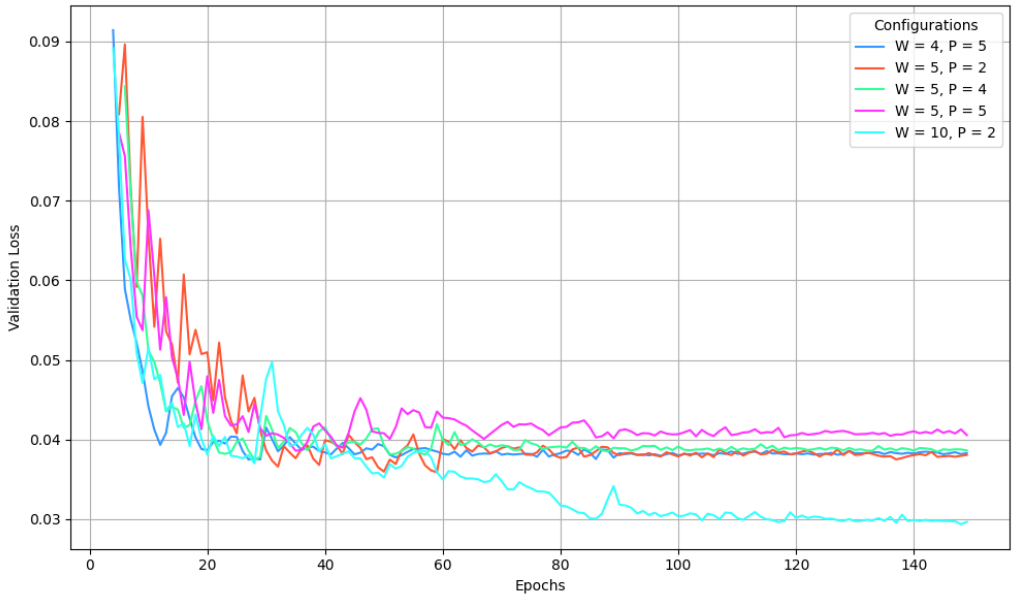


Figure 4.8: *Validation loss across different window size and patch size configurations.*

From the results shown in the picture above, the window size of 10 and the patch size of 2 yielded the lowest validation loss of 0.030. This result aligns with expectations, as a larger

window size and a smaller patch size allow the model to compute attention across larger receptive fields while preserving details.

4.2.2 Buffering

The buffer size (where the weight would be applied) and the weighting factor were optimized using **Optuna**, a hyperparameter optimization framework. The test range for buffering area was set at [1, 20] and the test range for weight was set at [1, 10]. 49 trials are completed. The best configuration is the buffer size of 8 and the weight of 4, resulting in a validation loss of 0.0223.

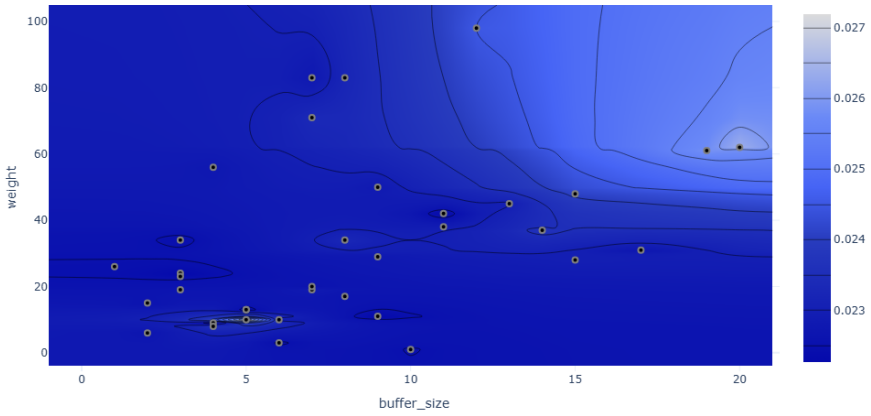


Figure 4.9: *Optimization results for buffer size and weight using Optuna.*

4.2.3 Divergence

In order to enforce the mass constraint to predictions, a divergence penalty was added into the loss function. To find the optimal divergence weight, an **Optuna** search of [1, 1000] was conducted with 40 trials in total. The best result is with divergence weight of 15, the best validation loss is 0.0222. It is worth noticing that this experiment was done without the setting of the buffering mentioned previously, therefore the influences of each modification can be examined separately.

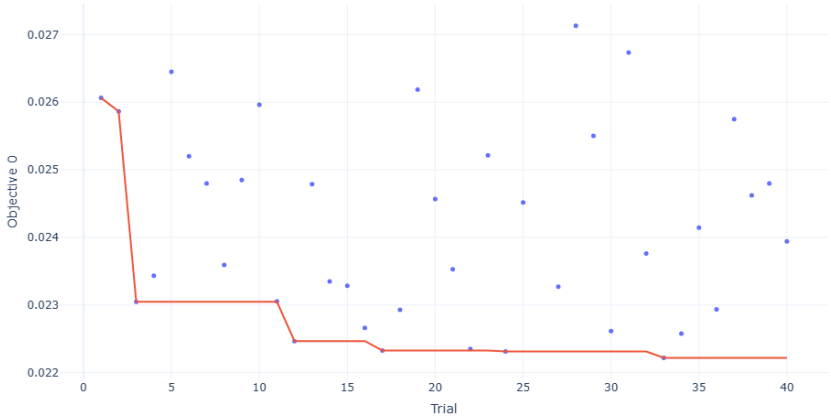


Figure 4.10: *Optimization results for divergence weight using Optuna.*

5 Results and analysis

This chapter presents the experimental results following the order of model configurations tested. Each section reports prediction performance under a specific condition or design choice. To avoid redundancy, results are presented using representative wind speeds and layout conditions. Detailed results for all experiments are included in the appendices.

5.1 Target format

To evaluate the impact of different target representations on prediction accuracy, two training targets were considered: wind speed magnitude (Mag) and decomposed x-velocity and y-velocity components (XY). Each model was trained separately using the same inflow settings, ranging from 5.0 m/s to 15.0 m/s.

Inflow (m/s)	Mag MAE	XY MAE	Mag RMSE	XY RMSE
5.0	1.1168	1.0434	2.1923	2.0568
7.5	1.6913	1.5772	3.3054	3.0724
10.0	2.2663	2.1085	4.3963	4.1628
12.5	2.8184	2.6378	5.4884	5.2062
15.0	3.4003	3.0828	6.5728	6.2352

Table 5.1: *Errors between prediction values and target values using magnitude-based and XY-based training.*

The XY-based model consistently achieves lower MAE and RMSE across all inflow speed settings. As shown in Figure 5.1, the performance gap between the two models widens as inflow speed increases.

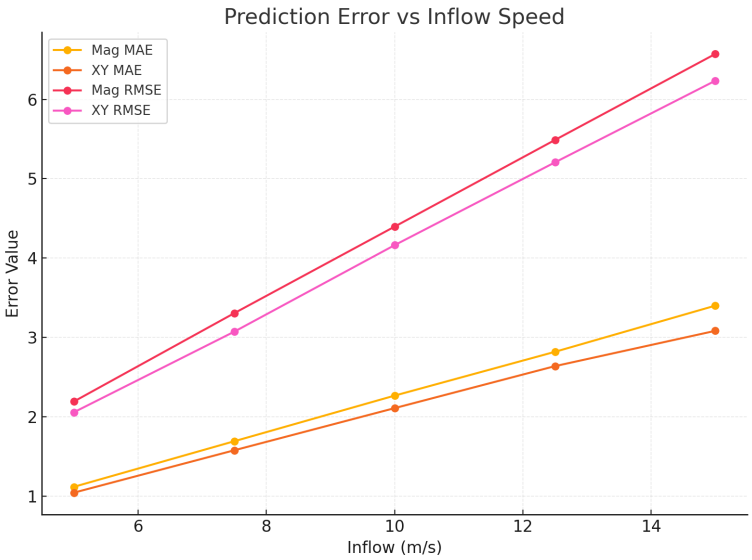
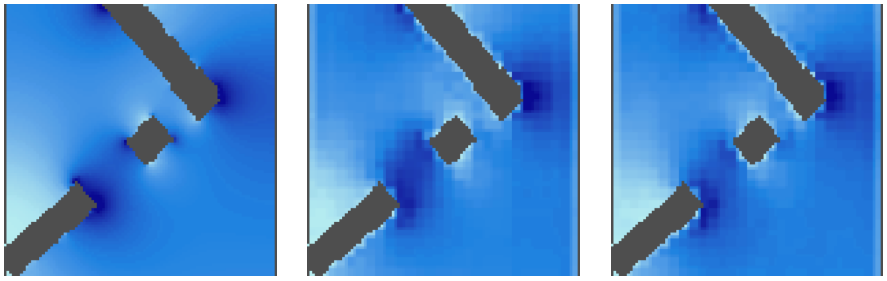


Figure 5.1: *Prediction errors (MAE and RMSE) at different inflow speeds for models trained with magnitude and XY velocity inputs.*



CFD simulation | Mag model prediction | XY model prediction

Figure 5.2: *Comparison of predicted magnitude fields at an inflow speed of 10.0 m/s.*

Figure 5.2 compares the predicted magnitude fields from both models against the CFD reference at 10.0 m/s. The XY-based prediction more closely resembles the CFD simulation, especially in the narrow passage between the two buildings, where it preserves sharper gradients and avoids blurry predictions shown in the magnitude-based result. Additionally, in the upper region of the domain, the XY-based output exhibits a more continuous gradient pattern, aligning more closely with the simulated wind speed transition. Despite the improved overall performance of the XY-based model, both models exhibited significant underestimation of wind speed in regions adjacent to building surfaces.

This change from magnitude-based input to XY-based input not only improves numerical accuracy, but also enables the model to incorporate directional information, making it capable of reconstructing vector gradients and flow patterns. Notably, prediction errors are most pronounced near building surfaces, where flow direction and magnitude change abruptly.

However, open-flow regions away from buildings often yield near-exact predictions, sometimes with maximum absolute errors below 1. The disproportionate error near boundaries contributes significantly to the overall RMSE, and also reveals the model's limitations in capturing fine-scale flow behavior in high-gradient areas.

5.2 Inflow speed

To evaluate the effect of inflow speed on prediction accuracy, both models were tested under five inflow settings ranging from 5.0 m/s to 15.0 m/s.

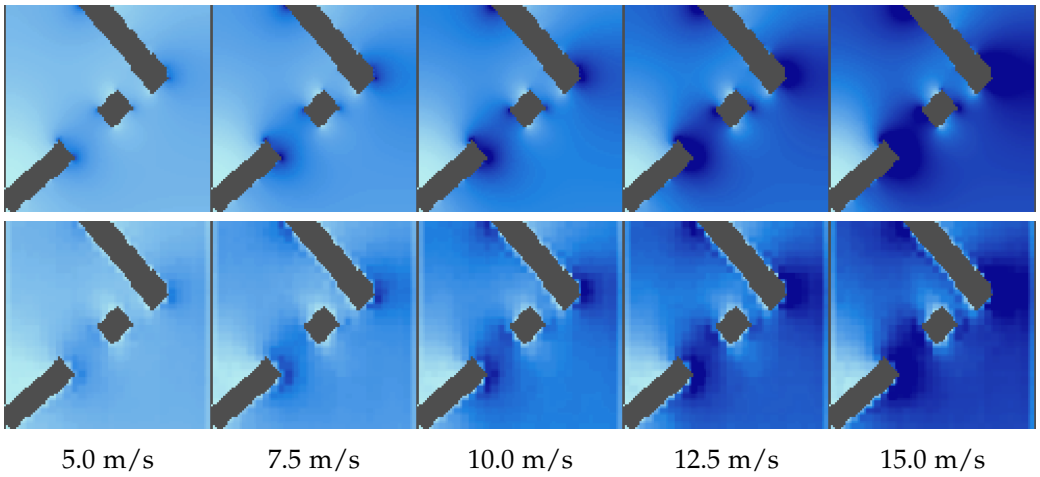


Figure 5.3: *Comparison of ground truth (top) and predicted magnitude fields (bottom) at five inflow speeds.*

Figure 5.3 shows that as inflow speed increases, spatial gradients become more pronounced, particularly along near-wall regions. This corresponds to a gradual increase in prediction error, especially under higher inflow conditions.

At 5.0 m/s, the prediction retains the global structure and local details well. At 15.0 m/s, however, error increases sharply, with both models underestimating velocity near wind-facing surfaces. This suggests that higher flow intensity introduces sharper transitions that the model struggles to resolve, revealing a gradient-related limitation in both formats. Prediction quality degrades almost linearly with inflow speed. RMSE rises from 2.0568 m s^{-1} at 5 m s^{-1} to 6.2352 m s^{-1} at 15 m s^{-1}, reflecting the larger gradients introduced at higher Reynolds numbers.

5.3 Layout

This section presents the model's performance across five urban layout types, using the XY-based model. The inflow speed is fixed at 10.0 m/s, and all other training parameters remain unchanged.

Layout Type	Prediction MAE	Prediction RMSE
Attached	2.8163	5.1063
Detached	1.5176	2.6771
High-rise	1.8168	3.2231
Industrial	2.1914	4.5721
Mixed	2.5982	5.2886

Table 5.2: *Prediction error across five urban layout types under fixed inflow speed (10.0 m/s). All results are from the XY-based model.*

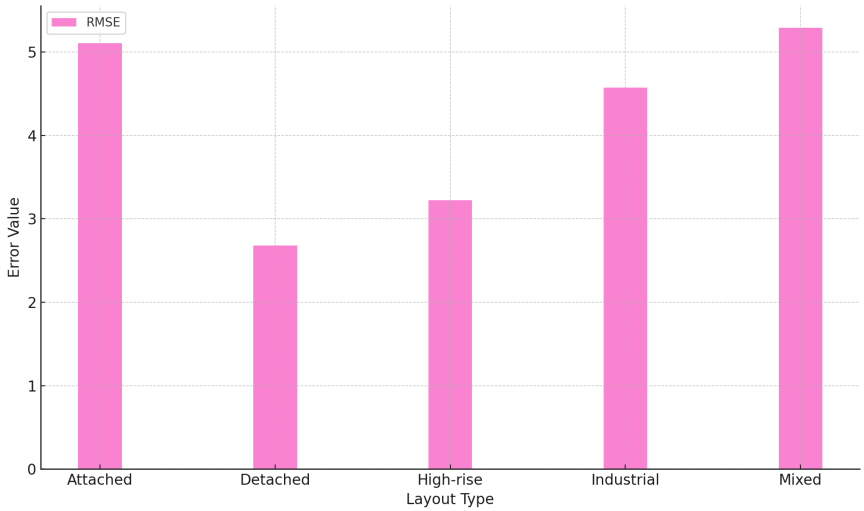


Figure 5.4: *RMSE distribution for different layout types. The error is highest for Mixed and Attached layouts.*

Table 5.2 and Figure 5.4 show that the ”Attached”, ”Mixed”, and ”Industrial” layouts result in higher prediction errors, whereas the ”Detached” and ”High-rise” layouts yield lower RMSE values. The RMSE of the ”Mixed” layout is nearly twice that of the ”Detached” layout.

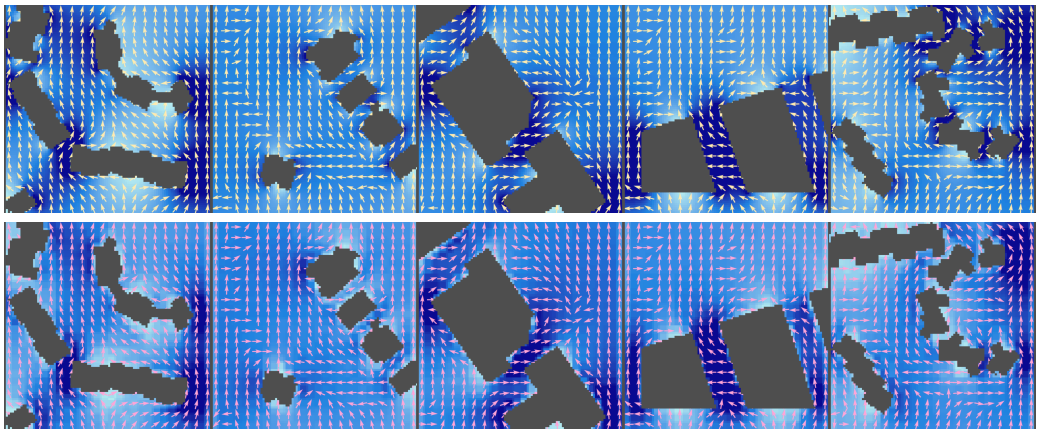


Figure 5.5: *Predicted wind magnitude fields and directions (bottom row) and corresponding ground truth (top row) for each layout type. All cases use the XY-based model at 10.0 m/s.*

In Figure 5.5, across all five layouts, errors tend to appear in regions near buildings, which is consistent with the findings in Section 5.2. In the Attached and Mixed cases, the predicted wind directions differ significantly from the ground truth, especially on the concave leeward side of buildings. The model incorrectly predicted wind reversal in these areas, treating the flow as bouncing off the building edges, whereas in the ground truth it follows the concave surfaces. This underestimation is particularly noticeable in the Mixed layout, where the model strongly underpredicts wind magnitude in the passage between buildings. In contrast, for tunnel-shaped spaces near the image edges, the model successfully captures the wind acceleration behavior. These results highlight that most prediction errors occur near building surfaces, where the flow interacts with complex geometry. Urban morphology is a primary driver of model error. Detached layouts record the lowest RMSE of 2.6771 m s^{-1}, whereas the mixed configuration peaks at 5.2886 m s^{-1}, underscoring the challenge posed by irregular street canyons, as well as the shape and orientation of buildings.

5.4 Buffer loss

After analyzing the effect of urban morphology, the next experiment introduces a spatial weighting strategy during training, motivated by the observation that near-building areas are the main source of error. Buffer loss is introduced to help the model focus more on regions adjacent to buildings and improve prediction accuracy in those areas. The buffer size is set to 8 and the weight to 4, based on the experiment results in Chapter 4. All evaluations in this section are conducted under 10.0 m/s inflow speed.

Model	Prediction MAE	Prediction RMSE
Baseline	2.1085	4.1628
With Buffer	2.0860	4.1315

Table 5.3: *Prediction error of XY-based model with and without buffer loss.*

Table 5.3 shows that, compared to the original XY-based model, the model with buffer loss achieves slightly lower error, reducing RMSE by only 0.03.

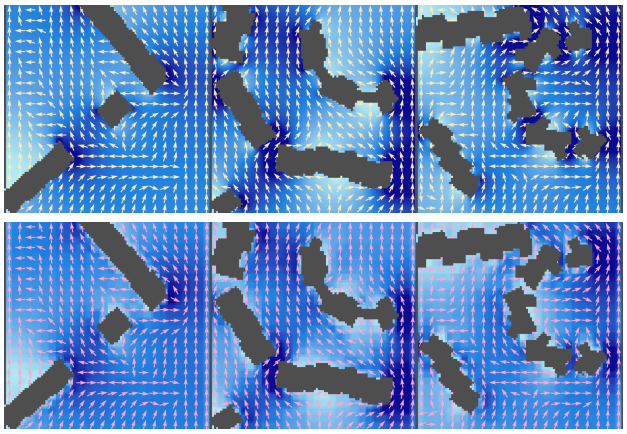


Figure 5.6: *Predicted velocity fields from the buffer-loss model (bottom row) and corresponding ground truth (top row).*

When comparing the results in Figure 5.6 to those in Figure 5.5, the same issues can still be observed. Wind direction errors persist, and wind magnitude in narrow passages continues to be underestimated. In the middle example, misjudgment of direction appears more severe than in the original model, with the predicted flow in some of the new areas changing further from the reference pattern. The example on the right shows improved prediction magnitude in certain passage areas, with higher values closer to the ground truth. In the preliminary tuning, an 8-pixel buffer with a weighting factor of 4 was singled out as the most effective choice. Across the 49 buffer–weight pairs explored with Optuna, this configuration yielded the lowest validation error: in the normalized early-test subset the RMSE dropped from 0.003 to 0.0223. Applied to the full test set, RMSE decreases from 4.1628 to 4.1315 $\mathrm{m\,s^{-1}}$ (–0.0313 $\mathrm{m\,s^{-1}}$).

5.5 Divergence

In addition to spatial prioritization, a physical constraint is also tested. This section introduces a physical consistency constraint into the loss function, namely the divergence loss, which encourages the output velocity field to reach a steady-state condition. The weight for divergence loss is set at 15 based on validation results in Chapter 4 and conducted under 10.0 m/s inflow speed.

Model	Prediction MAE	Prediction RMSE
Baseline	2.1085	4.1628
With Divergence	2.0559	4.1195

Table 5.4: *Prediction error of XY-based model with and without divergence loss.*

Table 5.4 shows that the divergence-loss model achieves slightly better performance than the buffer-loss model, with RMSE reduced by approximately 0.04.

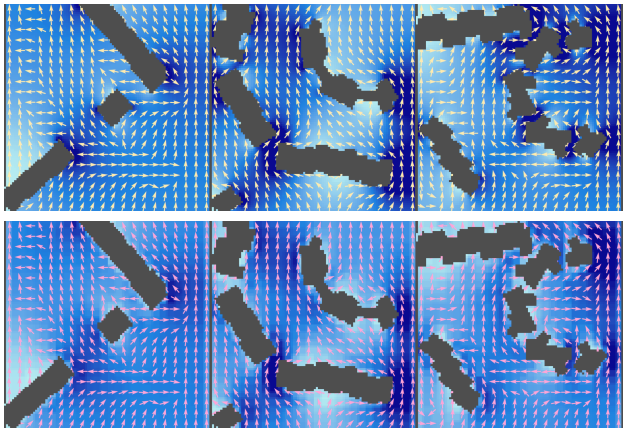


Figure 5.7: *Predicted velocity fields from the divergence-loss model (bottom row) and corresponding ground truth (top row).*

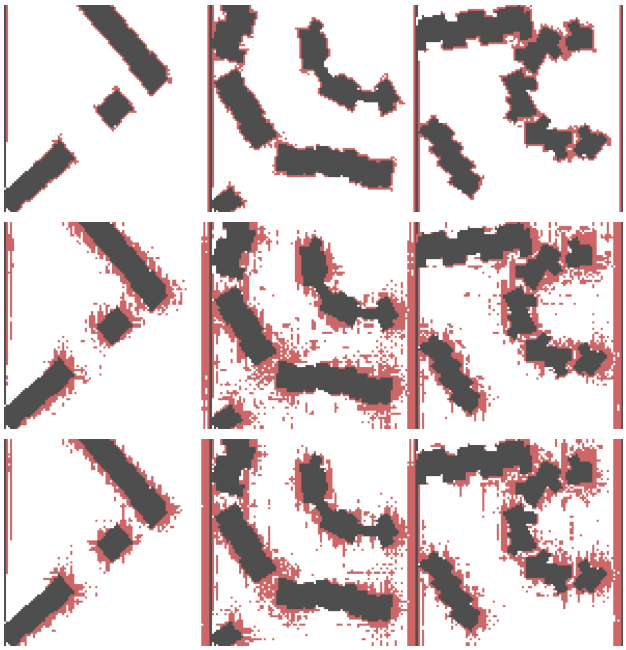


Figure 5.8: *Divergence comparisons across ground truth (top row), baseline model without divergence loss (middle row), and model with divergence loss (bottom row). Red pixels indicate locations where the absolute divergence exceeds 1.*

Given the small numerical gain on RMSE, no fundamental improvement is observed in the

predicted velocity fields. However, Figure 5.7 suggests that the outputs appear slightly more spatially continuous when divergence is included as a constraint. Figure 5.8 highlights regions where the absolute divergence exceeds 1. Compared to the baseline, the divergence-loss model reduces the occurrence of such regions, particularly in open areas away from buildings. Overall, the improvement remains small and localized. Introducing a divergence penalty offers only marginal global gains, the overall RMSE decreases only by 0.0433 m s^{-1}.

5.6 Buffer and divergence

In this section, buffer loss and divergence loss are jointly applied during training. Evaluations are conducted under 10.0 m/s inflow speed.

Model	Prediction MAE	Prediction RMSE
Baseline	2.1085	4.1628
With Buffer	2.0860	4.1315
With Divergence	2.0559	4.1195
With Buffer & Divergence	2.0939	4.1199

Table 5.5: *Prediction error of the baseline model, buffer-loss model, divergence-loss model, and combined-loss model.*

As shown in Table 5.5, the combined-loss model achieves an RMSE of 4.1199, which is comparable to the model with divergence loss alone and slightly better than the buffer-only model. However, the differences between these variants are small, all within 0.05 RMSE.

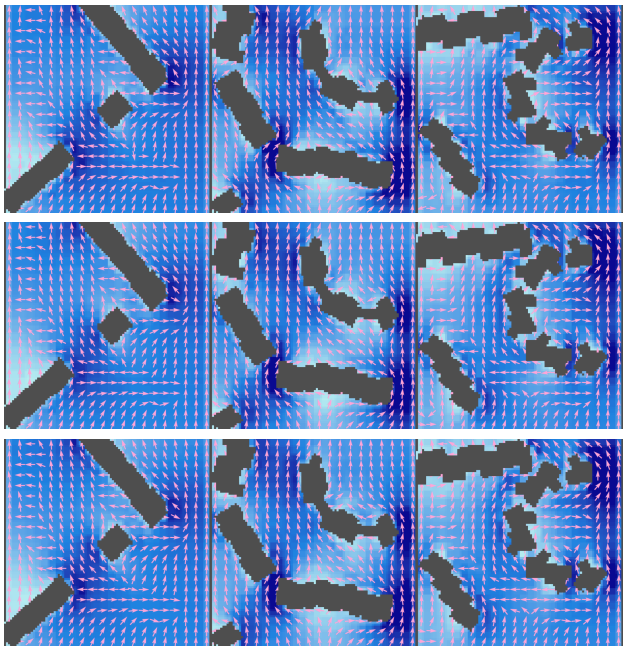


Figure 5.9: *Predicted velocity fields from buffer-loss model (top row), divergence-loss model (middle row), and combined-loss model (bottom row).*

Figure 5.9 compares the predicted velocity fields under all three configurations. Despite the additional loss terms, the previously identified issues still exist, including the underestimation of wind magnitude in passages and directional inconsistencies near building surfaces. Some improvement is visible in the upper central areas of the rightmost examples, where higher magnitudes are more accurately captured. However, these changes are subtle, and the overall predictive quality remains largely unchanged across configurations.

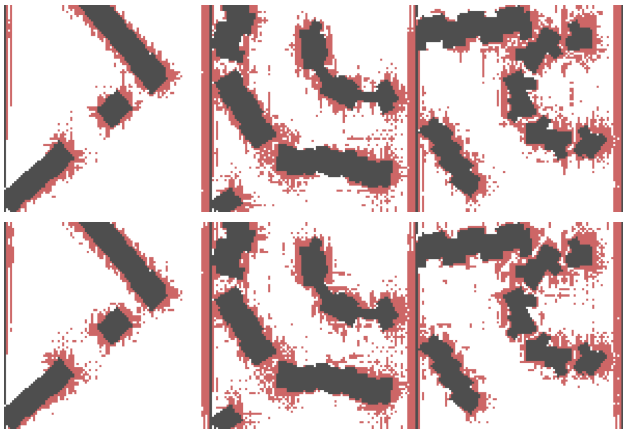


Figure 5.10: *Divergence comparisons across divergence-only (top row) and combined-loss (bottom row) models. Red pixels indicate locations where the absolute divergence exceeds 1.*

Figure 5.10 shows that both the divergence-loss model and the combined-loss model reduce high-divergence regions compared to the baseline, particularly in open areas. However, the divergence map of the combined-loss model is nearly indistinguishable from that of the divergence-only model,reaching only an RMSE of 4.1199 m s^{-1}.

5.7 Patch size

The only variable in this section is the patch size. All other settings remain the same as in the basic XY-based model. The primary goal is to analyze how the structural resolution of the Swin Transformer affects prediction performance.

Patch Size	Window Size	Prediction MAE	Prediction RMSE
2	10	2.1085	4.1628
5	10	2.6494	4.9148

Table 5.6: *Prediction error under different patch sizes. Window size is fixed at 10. All results are evaluated with inflow speed of 10.0 m/s.*

Increasing the patch size from 2 to 5 results in an immediate RMSE increase from 4.16 to 4.91. This effect is more significant than any change observed from non-architectural modifications.

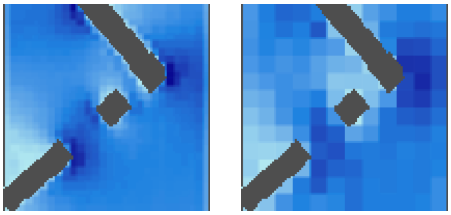


Figure 5.11: *Predicted magnitude fields from models with patch size 2 (left) and patch size 5 (right).*

Figure 5.11 shows that the patch size directly influences the visual granularity of the prediction. Compared to the result with patch size 2, the output from the model with patch size 5 appears significantly more blocky and less detailed. The results indicate that architectural resolution has a substantial impact on prediction performance. This suggests that the model's structural capacity, particularly its ability to capture finer spatial features, plays a more dominant role. Spatial resolution set by patch size dominates every other design choice. Increasing the patch from 2 to 5 pixels enlarges RMSE from 4.1628 m s^{-1} to 4.9148 m s^{-1}, confirming the need for fine-grain tokens in dense-prediction settings.

6 Discussion, conclusion and future work

This chapter interprets the experimental results presented in Chapter 5 and reflects on the model's capacity to simulate wind behavior in urban settings. The discussion is structured around the main hypotheses and derived insights. Based on these findings, the chapter concludes with an overall assessment of the model's performance and applicability, followed by a reflection on its limitations and possible directions for future work.

6.1 Discussion

Based on the research question in Chapter 1, the following hypotheses were formulated:

- Using decomposed XY-velocity components improves wind flow prediction compared to basing only on magnitude.
- Higher inflow speed increases prediction error due to stronger flow gradients.
- Denser and more irregular urban layouts lead to higher errors, especially near building boundaries.
- Incorporating buffer and divergence into loss function enhances prediction accuracy near structures and improves flow consistency.

To address the question and evaluate these hypotheses, a series of experiments was conducted to evaluate the effects of inflow speed, building layouts, training format, loss function and patch size on prediction accuracy. Among all tested configurations, the patch size had the strongest impact on prediction error. The second most influential factor was the input format, where using XY-velocity components instead of magnitude reduced RMSE by approximately 0.23. Other loss-related modifications, such as buffer and divergence terms, led to minor improvements but had no substantial impact on overall accuracy. Inflow intensity and urban morphology were further analyzed to identify conditions under which the model tends to fail, particularly near buildings and high-gradient regions.

Consistent with the first hypothesis, using XY-velocity as input led to the second-largest improvement across all configurations tested. Switching from magnitude to XY format reduces RMSE by approximately 0.23, and this improvement is consistent across inflow speeds. In addition to lowering error, this choice introduces directional information, enabling the model to represent vector gradients and directional flow patterns more effectively. These benefits are particularly valuable in near-building regions. Open areas are typically predicted with high accuracy by both models. The use of XY-velocity partially mitigates directional mispredictions near buildings, though significant underestimation still persists in high-complexity areas. This improvement is also illustrated in Figure 5.2, where the XY-based model captures finer details in the passage between buildings. Overall, using XY-velocity enhances the model's representation capacity by encoding directional information. Additionally, since x-

and y-components tend to exhibit lower individual variation than the combined magnitude, their gradients are generally smoother and thus easier to learn.

Prediction error increases with inflow speed, as shown in Section 5.2. This trend is consistent with the hypothesis that higher flow intensity introduces greater prediction difficulty. In all five cases, the maximum pixel value is approximately 15 times the inflow speed, and the average slightly exceeds the inflow speed. As inflow increases from 5.0 m/s to 15.0 m/s, the absolute range between inflow speed and maximum values grows from around 70 m/s to over 210 m/s. The CFD simulations show that as inflow speed increases, near-wall velocity gradients become steeper, especially along building edges, which increases the complexity of the flow field the model must learn to approximate. This is also reflected in Table 3.2, where the rasterized input shows wider value distributions, as indicated by the increasing standard deviation. The error pattern under increasing inflow is both global and localized. Globally, the RMSE increases linearly with the inflow speed as also illustrated in Figure 5.1. Locally, significant underestimation occurs along building surfaces that are partially aligned with the incoming wind. This may indicate that the model misinterprets these surfaces as impermeable boundaries, similar to fully orthogonal walls, thereby suppressing the predicted wind speed. In real urban flows, wind tends to accelerate around aligned structures and decelerate behind obstacles. The model may either lack sufficient training examples to learn this behavior, or be structurally limited in capturing directional variation. Overall, increasing inflow speed amplifies spatial gradients, exposing the model's limitations in resolving near-wall dynamics.

The third hypothesis predicted that denser and more irregular urban layouts would lead to higher RMSE, which is generally supported by the results, though with notable exceptions. Mixed layout produced the highest RMSE of 5.29, followed by Attached layout RMSE of 5.11. This is consistent with the fact that Mixed layout contains a wide variety of building shapes and spatial arrangements. However, Industrial layout yielded higher RMSE than High-rise layout, despite its more regular building arrangement. This contradicts the assumption that regular shapes would be easier to predict. This observation, when considered alongside the inflow analysis, suggests that the primary source of error may lie in the near-building regions.

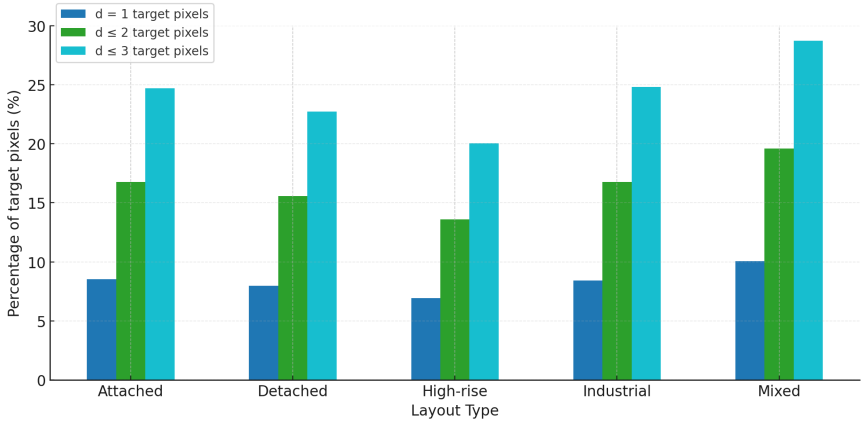


Figure 6.1: *Proportion of target pixels adjacent to buildings (based on Chebyshev distance).*

Figure 6.1 provides supporting evidence for this interpretation. As all five layout test sets share the same inflow speed, this comparison isolates how differences in urban morphology influence prediction error. Each input image consists of two pixel types: building pixels and target pixels, the latter being the prediction domain. Figure 6.1 shows the proportion of target pixels that are adjacent to buildings. The pattern closely mirrors the RMSE distribution across layouts shown in Figure 5.4. However, the Detached layout deviates from this trend. It has a higher proportion of near-building pixels than High-rise layout, yet yields a lower RMSE. This could be because, unlike High-rise layout, which contains large buildings and often leaves only narrow passages for wind to flow, Detached layout consists of small, dispersed buildings, which results in smoother gradients around structures and thus lower local prediction difficulty.

The findings in the previous two hypotheses suggest that increased input complexity, whether due to inflow intensity or urban geometry, makes it more difficult for the model to achieve accurate predictions in near-building regions, and correlates strongly with higher overall error.

The results do not provide strong evidence of a notable improvement, thus only partially supporting the fourth hypothesis. Buffer loss improves prediction accuracy near building surfaces by weighting error in boundary regions. Divergence loss, on the other hand, enforces global physical consistency by penalizing divergence in the velocity field. The buffer-loss model shows localized improvements in wall-adjacent passages, while divergence loss yields smoother transitions in open-flow zones. These two mechanisms appear to target complementary aspects of the prediction task. However, when combined, they do not produce cumulative improvement. This suggests potential interference in optimization, or a limitation in the model's capacity to respond to added constraints. While minor improvements are visible in specific regions, the overall enhancement remains very limited. One explanation is that these loss terms are soft constraints, which means they only encourage, but do not enforce, certain behaviors. Unlike physics-informed neural networks (PINNs), where physical laws are explicitly integrated into the loss formulation, the Swin Transformer still optimizes primarily for pixel-wise prediction accuracy. As a result, the model may not fully internalize global consistency or near-wall behavior. It continues to prioritize minimizing average error rather than enforcing flow-like behavior in its outputs. These findings suggest that while flow-aware modifications on loss function can influence output behavior, they are insufficient to overcome the architectural limitations that constrain the model's overall predictive capacity.

Although not part of the original hypotheses, the previous experimental results naturally led to another hypothesis: the Swin Transformer's capacity to simulate urban wind flow is largely governed by its architectural setting.

As mentioned previously, patch size stands out as the dominant factor affecting model performance. Smaller patch sizes allow the model to capture finer spatial details and localized flow features. This effect is based on the fact that patches serve as the smallest spatial unit of representation in Swin Transformer. The Swin Transformer operates on these non-overlapping patches and computes attention within windows. Therefore, spatial resolution is coarsened through patch merging layers, which progressively reduces feature map size and leads to information loss. As shown in Figure 5.11, larger patch sizes result in markedly coarser and more blurred predictions, with low-resolution results throughout the whole image rather than just near-building regions. Furthermore, since the upsampling is performed

mainly by convolutional layers, the pixel-level differences inside patches are not fully recovered during decoding, which amplifies the importance of patch size on result quality. Due to the limitation of the computational resource, patch size = 1 could not be tested, and patch size = 2 was the smallest feasible setting. But the RMSE increase of over 0.7 when patch size changes from 2 to 5, far exceeding the impact of all other configurations, strongly suggests that the architectural ability of the model itself plays a pivotal role in using Swin Transformer to simulate wind flow in urban environments.

The most pronounced advantage of the surrogate over the conventional CFD simulation is runtime. Table 6.1 list the time needed per sample both approaches. A complete CFD simulation run takes approximately 260 per sample, whereas the surrogate can returns a prediction in 4 seconds on the same hardware.

Step	**CFD**	**ML surrogate**
Ansys DesignModeler	3 min	–
Ansys Mesh	1 min 10 s	–
Ansys Fluent	10 s	–
Total per sample	**4 min 20 s**	**4 s**

Table 6.1: *Per–sample time cost of the conventional CFD run vs. the trained ML surrogate.*

The experiments demonstrate that the Swin Transformer has potential in approximating urban wind patterns, but remains limited in resolving highly complex zones. For wind simulation in urban areas, the model's architectural capacity is the decisive factor. Regardless of inflow speed, building layout, or input format, the results consistently point to one limitation: the model struggles to handle sharp local gradients between adjacent pixels. Accordingly, the model performs better when the target field is smoother and exhibits fewer localized extremes. Despite this shortcoming, the surrogate's ability of rapid producing predictions makes batch producing flow predictions on a much larger scale and real-time interactive design feasible. At its current configuration, the model, in its current form, remains unsuitable for use in pixel-level simulation.

6.2 Conclusion

This thesis investigates to what extent the Swin Transformer can simulate wind fields in urban area. The goal was to explore how factors related to geometry and flow dynamics affect the model's predictive ability.

Beyond applying the Swin Transformer as a surrogate for urban wind prediction, this work also contributes by constructing a 2D urban wind field dataset and developing a workflow for batch wind field data generation.

The results reveal that architectural resolution defined by patch size and input encoding play more decisive roles than building geometry, inflow speed or loss function design. While the current setup uses 2D data, the insights may inform future surrogate modeling for urban fluid dynamics.

The research question brought in Section 1.3 can be answered as following, begins with the sub-research questions:

- **How do different urban building layouts affect the accuracy of the surrogate in simulating wind fields?**

  Across five tested morphologies, densely packed and irregular urban layouts, such as Mixed and Attached types, lead to higher prediction error. However, the error is not solely determined by building density or size. Alignment and spacing also play critical roles, as demonstrated by the Detached layout type, which yielded the lowest RMSE.

- **How do varying inflow speeds affect the accuracy of the surrogate in urban wind field simulation?**

  Prediction error increases consistently with inflow speed, both globally and locally. A near-linear relationship is observed between inflow intensity and RMSE, reflecting the model's limited capacity to resolve sharp gradients at higher flow speeds.

- **How does introducing a flow-aware loss function influences the surrogate's accuracy?**

  Loss function modifications that incorporate flow-awareness, such as buffer and divergence terms, lead to subtle improvements in localized flow behavior. However, in terms of overall RMSE, their impact is negligible. The key finding is that even with explicit loss emphasis on near-wall accuracy, these techniques cannot overcome the architectural limitations of the model itself.

To conclude on the main research question:

- **To what extent can a Swin-Transformer-based surrogate accurately simulate wind fields in urban environments under specific initial conditions?**

  Overall, the model is capable of capturing the main structure of the wind field, but exhibits consistent limitations in high-gradient zones near buildings. The results also show that soft constraints have limited impact on accuracy compared to architectural factors when emulating physical systems. In practical terms, the model may be suited for quick estimation of overall flow patterns in regular urban geometries, but should not be relied on for detailed and accurate near-wall analysis or safety-critical airflow prediction. Its performance degrades substantially in regions of high velocity gradient, which are often the areas of most practical concern.

6.3 Future work

As this study demonstrates some of the potential and limitations of Swin Transformer models for 2D urban wind prediction, there are several directions arise from the present findings.

Geometric scale The current raster covers only 100×100 m and therefore captures differences in individual building shapes rather than city-scale planning. A larger coverage would allow neighborhood-scale planning questions to be addressed, while a 3D voxel input would allow better modeling of vertical wind dynamics and building height effects.

Transient inflow All experiments in this study assume steady boundary conditions. However, many practical assessments like pollutant dispersion, depend on transient inflow to capture time-varying wind behavior.

Physics-informed loss Neither buffer nor divergence settings enforces strict mass conservation. the incorporation of physics-informed loss functions such as PINN-based residual constraints may help improve near-wall accuracy and enhance physical interpretability of model outputs.

Bibliography

Anderson, J. D. (1995). *Computational Fluid Dynamics: The Basics with Applications*. McGraw-Hill, Inc., New York.

ANSYS, Inc. (2009). *ANSYS Fluent 12.0 User Guide*. ANSYS, Inc., Canonsburg, PA, USA.

Barlow, J. F. (2014). Progress in observing and modelling the urban boundary layer. *Urban Climate*, 10:216–240. ICUC8: The 8th International Conference on Urban Climate and the 10th Symposium on the Urban Environment.

BenMoshe, N., Fattal, E., Leitl, B., and Arav, Y. (2023). Using machine learning to predict wind flow in urban areas. *Atmosphere*, 14:990.

Blazek, J. (2001). *Computational Fluid Dynamics: Principles and Applications*. Elsevier, Oxford, UK, 3rd edition.

Blocken, B. (2018). Computational fluid dynamics for urban physics: Importance, scales, possibilities, limitations and ten tips and tricks towards accurate and reliable simulations. *Building and Environment*, 91:219–245.

Blocken, B., Stathopoulos, T., and van Beeck, J. (2016). Pedestrian-level wind conditions around buildings: Review of wind-tunnel and cfd techniques and their accuracy for wind comfort assessment. *Building and Environment*, 100:50–81.

Cai, S., Mao, Z., Wang, Z., Yin, M., and Karniadakis, G. E. (2021). Physics-informed neural networks (PINNs) for fluid mechanics: A review. *Acta Mechanica Sinica*, 37(12):1727–1738.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

Ferziger, J. H. and Perić, M. (2002). *Computational Methods for Fluid Dynamics*. Springer, Berlin, Germany, 3rd edition.

Guo, X., Li, W., and Iorio, F. (2016). Convolutional Neural Networks for Steady Flow Approximation. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 481–490, New York, NY, USA. Association for Computing Machinery.

Hirsch, C. (2007). *Numerical Computation of Internal and External Flows*. Butterworth-Heinemann, Oxford, UK, 2nd edition.

Holger Koss, H. (2006). On differences and similarities of applied wind comfort criteria. *Journal of Wind Engineering and Industrial Aerodynamics*, 94(11):781–797. Urban Civil Engineering (UCE), Impact of wind and storms on city life and built environment.

Huang, X., Zhang, Y., Liu, J., Zhang, X., and Liu, S. (2023). A Short-Term Wind Power Forecasting Model Based on 3D Convolutional Neural Network–Gated Recurrent Unit. *Sustainability*, 15(19):14171.

Kastner, P. and Dogan, T. (2023). A GAN-Based Surrogate Model for Instantaneous Urban Wind Flow Prediction. *Building and Environment*, 242:110384.

Khan, S., Naseer, M., Hayat, M., Zamir, S. W., Khan, F. S., and Shah, M. (2021). Transformers in vision: A survey. *arXiv preprint arXiv:2101.01169*.

Kutz, J. N. (2017). Deep learning in fluid dynamics. *Journal of Fluid Mechanics*, 814:1–4.

LaNasa, P. J. and Upp, E. L. (2014). 2 - basic flow measurement laws. In LaNasa, P. J. and Upp, E. L., editors, *Fluid Flow Measurement (Third Edition)*, pages 19–29. Butterworth-Heinemann, Oxford, third edition edition.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Li, X.-X., Liu, C.-H., Leung, D. Y., and Lam, K. (2006). Recent progress in cfd modelling of wind field and pollutant transport in street canyons. *Atmospheric Environment*, 40(29):5640–5658.

Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., and Guo, B. (2021). Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10012–10022.

Miyanawala, T. P. and Jaiman, R. K. (2018). An Efficient Deep Learning Technique for the Navier-Stokes Equations: Application to Unsteady Wake Flow Dynamics.

Moukalled, F., Mangani, L., and Darwish, M. (2016). *The Finite Volume Method in Computational Fluid Dynamics: An Advanced Introduction with OpenFOAM and MATLAB*. Springer, Cham, Switzerland.

Munson, B. R., Young, D. F., Okiishi, T. H., and Huebsch, W. W. (2013). *Fundamentals of Fluid Mechanics*. Wiley, Hoboken, NJ, USA, 7th edition.

O'Shea, K. and Nash, R. (2015). An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*.

Peters, R., Dukai, B., Vitalis, S., van Liempt, J., and Stoter, J. (2022). Automated 3d reconstruction of lod2 and lod1 models for all 10 million buildings of the netherlands.

Sforza, P. M. (2012). Chapter 2 - quasi-one-dimensional flow equations. In Sforza, P. M., editor, *Theory of Aerospace Propulsion*, Aerospace Engineering, pages 35–53. Butterworth-Heinemann, Boston.

Sun, C., Zhang, F., Zhao, P., Zhao, X., Huang, Y., and Lu, X. (2021). Automated Simulation Framework for Urban Wind Environments Based on Aerial Point Clouds and Deep Learning. *Remote Sensing*, 13(12):2383.

Tu, J., Yeoh, G. H., and Liu, C. (2018). *Computational Fluid Dynamics: A Practical Approach*. Butterworth-Heinemann, Oxford, UK, 3rd edition.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention Is All You Need.

Versteeg, H. K. and Malalasekera, W. (2007). *An Introduction to Computational Fluid Dynamics: The Finite Volume Method*. Pearson Education Limited, Harlow, England, 2nd edition.

White, C., Ushizima, D., and Farhat, C. (2019). Fast Neural Network Predictions from Constrained Aerodynamics Datasets.

Willemsen, E. and Wisse, J. A. (2007). Design for wind comfort in the netherlands: Procedures, criteria and open research issues. *Journal of Wind Engineering and Industrial Aerodynamics*, 95(9):1541–1550.

Zhao, H., Jiang, L., Jia, J., Torr, P. H., and Koltun, V. (2021). Point transformer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 16259–16268.

Scripts

1 Clipping geometry in QGIS

```
from qgis.core import QgsProject, QgsVectorLayer, QgsGeometry, QgsFeature,
    QgsVectorFileWriter, QgsCoordinateReferenceSystem
from qgis.core import QgsProcessingException
import processing
import os

# Input and output paths
large_vector_layer_path = ".../tiles/tile.shp"

# Load the large vector layer
large_vector_layer = QgsVectorLayer(large_vector_layer_path, "Mother Layer", "ogr")

rotate_large_vector = processing.run("native:rotatefeatures", {
                                'INPUT':large_vector_layer,
                                'ANGLE':0,
                                'ANCHOR':None,
                                'OUTPUT':'memory:'
                            })['OUTPUT']

if not rotate_large_vector.isValid():
    print("Layer failed to load!")
else:
    # Generate grid covering the extent of the large vector layer
    extent = large_vector_layer.extent()
    extent_string = f"{extent.xMinimum()},{extent.yMinimum()},
    {extent.xMaximum()},{extent.yMaximum()}" # Format extent as string
    print(extent_string)

    grid_layer = processing.run("qgis:creategrid", {
        'TYPE': 2, # Rectangle (Polygon)
        'EXTENT': extent,
        'HSPACING': 100,
        'VSPACING': 100,
        'HOVERLAY':0,
        'VOVERLAY':0,
        'CRS': large_vector_layer.crs(),
        'OUTPUT':'memory:'
    })['OUTPUT']

    QgsProject.instance().addMapLayer(grid_layer)
    QgsProject.instance().addMapLayer(large_vector_layer)
```

```
# Process each grid cell
cells = [ft for ft in grid_layer.getFeatures()]
i = 1
for cell in cells:
    try:
        grid_geometry = cell.geometry() # Get the geometry of the feature
        grid_bbox = grid_geometry.boundingBox() # Get the bounding box of the
            geometry
        xmin = grid_bbox.xMinimum()
        ymin = grid_bbox.yMinimum()
        clipped_layer = processing.run("native:extractbyextent", {
            'INPUT': rotate_large_vector,
            'EXTENT':grid_bbox,
            'CLIP':True,
            'OUTPUT':'memory:'})['OUTPUT']

        # Filter out small polygons
        ids_to_delete = [f.id() for f in clipped_layer.getFeatures() if
            f.geometry().area() < 100]
        clipped_layer.dataProvider().deleteFeatures(ids_to_delete)

        total_area = sum(f.geometry().area() for f in clipped_layer.getFeatures())
        if total_area < 1000:
            continue

        # QgsProject.instance().addMapLayer(clipped_layer)
        print("Total Area: ", total_area)

        # Dissolve remaining polygons
        dissolved_layer = processing.run("native:dissolve", {
            'INPUT': clipped_layer,
            'OUTPUT': 'memory:'
        })['OUTPUT']

        # Filter out small polygons
        # ids_to_delete = [f.id() for f in dissolved_layer.getFeatures() if
            f.geometry().area() < 100]
        # dissolved_layer.dataProvider().deleteFeatures(ids_to_delete)

        # total_area = sum(f.geometry().area() for f in
            dissolved_layer.getFeatures())
        # if total_area < 1000:
        #     continue

        # Translate geometry to set the left bottom corner to (0, 0)
        center = dissolved_layer.extent().center()
        translated_layer = processing.run("native:translategeometry", {
            'INPUT': dissolved_layer,
            'DELTA_X': -xmin,
            'DELTA_Y': -ymin,
            'OUTPUT': 'memory:'
        })['OUTPUT']

        QgsProject.instance().addMapLayer(translated_layer)
```

```
        except QgsProcessingException as err:
            print(f"QgsProcessingException occurred for grid cell {i}: {err}")
        except Exception as err:
            print(f"An unexpected error occurred for grid cell {i}: {err}")

        # Export to DXF
        # dxf_path = os.path.join(output_folder, f"industrial_{i}.dxf")
        # QgsVectorFileWriter.writeAsVectorFormat(translated_layer, dxf_path, "utf-8",
            dissolved_layer.crs(), "DXF", skipAttributeCreation=True)
        # dxfFile.close()

        # print(f"Exported DXF {i} successfully.")

        i = i + 1

    print("Process completed.")
```

2 Ansys DesignModeler

```
function CreateWindZone (p)
{
p.Plane = agb.GetActivePlane();
p.Origin = p.Plane.GetOrigin();
p.XAxis = p.Plane.GetXAxis();
p.YAxis = p.Plane.GetYAxis();

p.Sk1 = p.Plane.NewSketch();
p.Sk1.Name = "Sketch1";

with (p.Sk1)
{
  p.Ln7 = Line(100.00000000, 150.00000000, 0.00000000, 150.00000000);
  p.Ln8 = Line(0.00000000, 150.00000000, 0.00000000, -50.00000000);
  p.Ln9 = Line(0.00000000, -50.00000000, 100.00000000, -50.00000000);
  p.Ln10 = Line(100.00000000, -50.00000000, 100.00000000, 150.00000000);
}

//Dimensions and/or constraints
with (p.Plane)
{
  //Constraints
  HorizontalCon(p.Ln7);
  HorizontalCon(p.Ln9);
  VerticalCon(p.Ln8);
  VerticalCon(p.Ln10);
  CoincidentCon(p.Ln7.End, 0.00000000, 150.00000000,
              p.Ln8.Base, 0.00000000, 150.00000000);
  CoincidentCon(p.Ln8.End, 0.00000000, -50.00000000,
              p.Ln9.Base, 0.00000000, -50.00000000);
```

```
  CoincidentCon(p.Ln9.End, 100.00000000, -50.00000000,
              p.Ln10.Base, 100.00000000, -50.00000000);
  CoincidentCon(p.Ln10.End, 100.00000000, 150.00000000,
              p.Ln7.Base, 100.00000000, 150.00000000);
}

p.Plane.EvalDimCons(); //Final evaluate of all dimensions and constraints in plane

// Create surface
ag.selectedFeature = ag.gui.TreeviewFeature(p.Sk1.Name, 0);
var Surf1 = ag.gui.CreateSurfSk();
Surf1.Name = "Zone";
Surf1.Operation = ag.c.Frozen;

return Surf1;
} //End Plane JScript function: planeSketchesOnly


function ImportCAD (path)
{
// Import CAD file
var imp=ag.b.Import(path);
imp.Name="CAD_geom";
imp.LineBodies = ag.c.Yes;
imp.Operation = ag.c.Frozen;
imp.PutBasePlane(ag.b.GetXYPlane());
agb.Regen();

var medges = ag.m.ModelEdges();

for (var i = 1; i <= medges.Count; i++)
{
  var edge = medges.Item(i);
  agb.AddSelect(agc.TypeEdge3d, edge);
}
var Surf2 = agb.SurfFromLines();
Surf2.Name = "Building";
agb.regen();
return Surf2;
} //End CAD import JScript function: importCAD

ag.gui.NewFile();
ag.m.ClearAllErrors();
ag.m.NewSession (true);
ag.gui.setUnits(ag.c.UnitMeter, ag.c.UnitDegree, ag.c.No);

var Sf2 = ImportCAD(path_CAD);
var Sf1 = CreateWindZone (new Object());
ag.m.SuppressLineBodies();
ag.b.Regen();
var numBody = ag.fm.BodyCount;
var fBoolean = ag.gui.CreateBoolean();
fBoolean.Name="windZone";
fBoolean.Operation = 2;
```

```
fBoolean.Preserve=ag.c.No;

ag.listview.ActivateItem("Target Bodies");
ag.gui.SelectAll();
ag.listview.ItemValue = "Apply";
ag.listview.ActivateItem("Tool Bodies");
for (var j = 1; j < numBody-1; j++){
body = ag.fm.Body(j);
agb.AddSelect(agc.TypeBody, body);}
ag.listview.ItemValue = "Apply";
ag.b.Regen();

var fedges = ag.m.ModelEdges();
var egCount = fedges.Count;
var inletIdx = egCount/2 + 1;
var outletIdx = egCount/2 - 1;
var inletEdge = fedges.Item(inletIdx);
var outletEdge = fedges.Item(outletIdx);

agb.AddSelect(agc.TypeEdge3d, inletEdge);
ns1 = ag.gui.CreateSelectionSet();
ns1.Name = "inlet";

agb.AddSelect(agc.TypeEdge3d, outletEdge);
ns2 = ag.gui.CreateSelectionSet();
ns2.Name = "outlet";

//Finish
agb.Regen(); //To insure model validity
//End DM JScript
```

3 Ansys Mesh

```
# encoding: utf-8
# 2023 R2

SetScriptVersion(Version="23.2.142")

# Define a list of geometry file paths
import os
system1 = GetSystem(Name="SYS")
CAD_files = "\".../cad";
geometry_files = ".../msh"

list = [69, 152, 161, 210, 319, 376, 397]

for i in range(1013, 1028):
   try:
      geometry1 = system1.GetContainer(ComponentName="Geometry")
      geometryProperties1 = geometry1.GetGeometryProperties()
```

```
    geometryProperties1.GeometryImportAnalysisType = "AnalysisType_2D"
    # Set the file path for the current iteration
    path_CAD = CAD_files + str(i) + ".dxf\";\n"
    new_line = "var path_CAD = " + path_CAD
    js_path = ".../create_windzone.js"
    with open(js_path, 'r') as file:
       existing_content = file.read()
    modified_content = new_line + existing_content
    temp_js_path = ".../create_windzone_temp.js"
    with open(temp_js_path, 'w') as file_temp:
       file_temp.write(modified_content)
    geometry1.Edit()
    command = """
    var scriptPath="...create_windzone_temp.js";
    runIt(scriptPath);
    function runIt(path) {
       ag.m.BeginUserScript();
       try {
          ag.runningScriptPath = path;
          ag.wb.ScriptEngine.AddNamedItem("ag", ag);
          ag.wb.ScriptEngine.RunScript(path);
       } catch(e) {
          ag.m.DisplayMessage("Error in JScript-> " + path, 2);
       }
       ag.m.EndUserScript();
    }
    """
    geometry1.SendCommand(Command=command)
    geometry1.Exit()
    os.remove(temp_js_path)

    geometry_file_path = geometry_files + str(i) + ".agdb"
    mesh1 = system1.GetContainer(ComponentName="Mesh")
    meshProperties1 = mesh1.GetMeshProperties()
    meshProperties1.saveMeshFileInSeparateFile = True

    meshComponent1 = system1.GetComponent(Name="Mesh")
    meshComponent1.Refresh()
    mesh1.Edit()
    mesh1.SendCommand(Command="""WB.AppletList.Applet
        ("DSApplet").App.Script.doToolsRunMacro('.../mesh.py');""")

    mesh_filename = geometry_file_path.split('.')[0] + ".msh"
    command_to_export_mesh = """var DS = WB.AppletList.Applet("DSApplet").App; SC =
        DS.Script; SC.doFileExport(FilePath="{}");""".format(mesh_filename)
    mesh1.SendCommand(Command=command_to_export_mesh)
    mesh1.Exit()
    meshComponent1.Update()
  except:
    pass
```

```
# encoding: utf-8
# 2023 R2
```

```
SetScriptVersion(Version="23.2.142")

# Define a list of geometry file paths
geometry_files = ".../agdbs/"

system1 = GetSystem(Name="SYS")

for i in range(1,4):
    try:
      geometry_file_path = geometry_files + str(i) + ".agdb"
      geometry1 = system1.GetContainer(ComponentName="Geometry")
      geometryProperties1 = geometry1.GetGeometryProperties()
      geometryProperties1.GeometryImportAnalysisType = "AnalysisType_2D"
      # Set the file path for the current iteration

      geometry1.Edit()

      geometry1.SendCommand(Command = """
      ag.gui.NewFile();
      ag.m.ClearAllErrors();
      ag.m.NewSession (true);
      ag.gui.setUnits(ag.c.UnitMillimeter, ag.c.UnitDegree, ag.c.No);
      var CAD_files = ".../cads/";
      var path_CAD = CAD_files + "cad" + i + ".dxf";
      """)

      export_path = CAD_files + "dm" + i + ".agdb";
      geometry1.Export(export_path)


      mesh1 = system1.GetContainer(ComponentName="Mesh")
      meshProperties1 = mesh1.GetMeshProperties()
      meshProperties1.saveMeshFileInSeparateFile = True

      meshComponent1 = system1.GetComponent(Name="Mesh")
      meshComponent1.Refresh()
      mesh1.Edit()
      mesh1.SendCommand(Command="""WB.AppletList.Applet
          ("DSApplet").App.Script.doToolsRunMacro('.../mesh.py');""")

      mesh_filename = geometry_file_path.split('.')[0] + ".msh"
      command_to_export_mesh = """var DS = WB.AppletList.Applet("DSApplet").App; SC =
          DS.Script; SC.doFileExport(FilePath="{}");""".format(mesh_filename)
      mesh1.SendCommand(Command=command_to_export_mesh)
      mesh1.Exit()
      meshComponent1.Update()
    except:
   pass
```

4 Ansys PyFluent

```
%matplotlib inline
import ansys.fluent.core as pyfluent
solver = pyfluent.launch_fluent(product_version="23.2", mode='solver', version='2d',
    precision='single', processor_count=1, show_gui = False)
for i in range(1, 1028):
    try:
        mesh_path = f".../mshs/msh{i}.msh"
        solver.file.read_mesh(file_name = mesh_path)

        # setting the velocity magnitude as 10 m/s
        solver.setup.boundary_conditions.velocity_inlet['inlet'].vmag = 5
        # Set the turbulence model
        solver.setup.models.viscous = {"model": "k-epsilon"}
        # Set the solver settings
        solver.solution.methods.p_v_coupling.flow_scheme = 'SIMPLE'
        solver.solution.initialization.hybrid_initialize()
        solver.solution.run_calculation.iter_count = 2000
        solver.solution.run_calculation.reporting_interval = 2000
        # Run the simulation
        solver.solution.run_calculation()

        solution_path = f".../winds/wind{i}.txt"
        # Export rake data to csv
        solver.file.export.ascii(name = solution_path,
                                 surface_name_list = [],
                                 delimiter = "comma",
                                 cell_func_domain = ["x-velocity", "y-velocity",
                                     "velocity-magnitude", ],
                                 location = "node")
        print(f"Mesh {i} is done")
    except:
        print(f"Error in mesh {i}")
        continue
```

5 Converting point data to raster in ArcMap

```
import arcpy
from arcpy.sa import *
arcpy.CheckOutExtension("Spatial")
arcpy.env.overwriteOutput = True
arcpy.env.addOutputsToMap = False
arcpy.env.workspace = ".../PointToRaster"
arcpy.env.extent = arcpy.Extent(0, 0, 100, 100)
start = 801
end = 900
for i in range(start, end+1):
    try:
        input_txt = ".../winds/wind" + str(i) + ".txt"
```

```
        arcpy.MakeXYEventLayer_management(input_txt, "x-coordinate", "y-coordinate",
            "XYLayer_temp")
        print("XYLayer_temp created successfully")
        point_name = "point" + str(i)+ ".shp"
        arcpy.CopyFeatures_management("XYLayer_temp", point_name)
        x_wind_output_raster = ".../x_winds/x_wind" + str(i) + ".tif"
        x_wind_name = "x_wind" + str(i) + ".tif"
        x_wind = arcpy.PointToRaster_conversion(point_name, "x_velocity", x_wind_name,
            cell_assignment="MINIMUM",priority_field = "", cellsize = 1)
        arcpy.CopyRaster_management(x_wind, x_wind_output_raster)
        print("x_wind" + str(i) + ".tif created successfully")
        y_wind_output_raster = ".../y_winds/y_wind" + str(i) + ".tif"
        y_wind_name = "y_wind" + str(i) + ".tif"
        y_wind = arcpy.PointToRaster_conversion(point_name, "y_velocity", y_wind_name,
            cell_assignment="MINIMUM",priority_field = "", cellsize = 1)
        arcpy.CopyRaster_management(y_wind, y_wind_output_raster)
        print("y_wind" + str(i) + ".tif created successfully")
    except:
        print("Error on wind" + str(i) + ".tif")
```

Colophon

This document was typeset using LaTeX, using the KOMA-Script class `scrbook`. The main font is Palatino.