Urban Airflow Field Prediction Using Sparse Sensors: A Hybrid Approach with Reduced-Order Model, Temporal Forecasting, and Multi-Task Learning

Arash Kamaliha

A Thesis

In the Department

of

Building, Civil, and Environmental Engineering

Presented in Partial Fulfilment of the Requirements

For the Degree of

Master of Applied Science in (Building Engineering) at

Concordia University

Montreal, Quebec, Canada

December 2025

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Arash Kamaliha**

Entitled: **Urban Airflow Field Prediction Using Sparse Sensors: A Hybrid Approach with Reduced-Order Models, Temporal Forecasting, and Multi-Task Learning**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Building, Civil, and Environmental Engineering)

Signed by the final Examining Committee:

_____Chair

Dr. F. Haghighat

_____Examiner

Dr. M. Ouf

_____Examiner

Dr. F. Haghighat

_____Supervisor

Dr. F. Nasiri

Approved by_____

Dr. Chunjiang An and Dr. Po-Han Chen , Graduate Program Director

Abstract

Urban Airflow Field Prediction Using Sparse Sensors: A Hybrid Approach with Reduced-Order Modeling, Temporal Forecasting, and Multi-Task Learning

Arash Kamaliha

Urban centers worldwide are becoming ever denser, increasing the need for rapid and accurate wind modeling to ensure pedestrian comfort, optimize air quality management, and enable safe and efficient urban air mobility (UAM) operations. Traditional methods for modeling wind flow, such as computational fluid dynamics (CFD) simulations and field measurements, are computationally intensive and typically depend on dense sensor networks, which limits their practicality for real-time wind monitoring across large urban areas. The current thesis proposes an integrated approach for the fast reconstruction of the wind velocity fields in urban environments using only online sensor signals. The model development phase involves three main components: First, a bidirectional long short-term memory (BILSTM) network is trained by using the offline sensors signals to capture the temporal evolution of the sensor readings. Second, the sparsity-promoting dynamic mode decomposition (SP-DMD) algorithm is employed to extract the important spatial pattern (modes) and their associated dynamic coefficients from the high-dimensional wind velocity data. Finally, a multi-task learning (MTL) network, is developed to map sensor signals to the real (growth/decay) and imaginary (oscillatory) components of these dynamic coefficients at each time step. During the prediction phase, real-time and out-of-sample sensor signals from the same locations are fed into the BILSTM model to forecast future measurements. The predicted signals are then input into the trained MTL network to infer the corresponding dynamic coefficients. Finally, the wind field is reconstructed in real-time via a linear combination

of the SP-DMD modes and their coefficients. The proposed framework was validated using high-fidelity CFD simulations for wind velocity flow around an isolated high-rise building. Results demonstrated that the proposed method has a robust capability for online reconstruction of precise urban flow fields. Additionally, during the online prediction phase, the model performance is evaluated for different level of noise in sensor signals, highlighting the model's robustness and generalizability. These findings suggest that the integrated modeling strategy can substantially lower operational costs and holds promise as a surrogate model for applications in next-generation urban planning and autonomous aerial navigation systems.

Key words: Urban Wind Flow Prediction, Reduced Order Modeling, Multi-task Learning, Dynamic Mode Decomposition, Computational Fluid Dynamics, Bidirectional Long Short-Term Memory

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my supervisor, Professor Fuzhan Nasiri, for their invaluable guidance, continuous support, and encouragement throughout the course of this research. Their expertise, patience, and constructive feedback were instrumental in shaping the direction and quality of this thesis.

I would also like to extend my sincere appreciation to the faculty and staff of the Department of Building, Civil, and Environmental Engineering (BCEE) at Concordia University for fostering a supportive and intellectually stimulating academic environment during my studies.

My heartfelt thanks go to my colleagues and friends in the research group, whose insightful discussions and camaraderie made this research journey both productive and enjoyable.

I would like to express special appreciation to the Next-Generation Cities Institute and Shahin Masoumi-Verki for providing the wind simulation dataset used in the case study for this research.

Finally, I am profoundly grateful to my family for their unwavering love, patience, and encouragement. To my parents, thank you for your endless support and for the countless sacrifices that made this achievement possible.

Content

List of Figures

List of Tables

List of Abbreviations

ADMM – Alternating Direction Method of Multipliers

BILSTM – Bidirectional Long Short-Term Memory

CAEs – Convolutional Autoencoders

CFD – Computational Fluid Dynamics

CNNs – Convolutional Neural Networks

CS – Compressed Sensing

DA – Data Assimilation

DDNN-ROM – Data-Driven Neural Network Reduced-Order Model

DMD – Dynamic Mode Decomposition

E-LES – Embedded Large Eddy Simulation

EDMD – Extended Dynamic Mode Decomposition

ESPOD – Ensemble Sparsity POD

FCNNs – Fully Connected Neural Networks

FFT – Fast Fourier Transform

FOM – Full-Order Model

HODMD – Higher-Order Dynamic Mode Decomposition

IROM – Intrusive Reduced-Order Model

LES – Large-Eddy Simulation

LSTM – Long Short-Term Memory

LiDAR – Light Detection and Ranging

MAE – Mean Absolute Error

MRDMD – Multi-Resolution Dynamic Mode Decomposition

MSE – Mean Squared Error

MTL – Multi-Task Learning

NIROM – Non-Intrusive Reduced-Order Model

NZ – Number of None-Zero Mode Amplitudes

PCA – Principal Component Analysis

PDE – Partial Differential Equation

PIDMD – Physics-Informed Dynamic Mode Decomposition

PIGNNs – Physics-Informed Graph Neural Networks

PINNs – Physics-Informed Neural Networks

PLW – Pedestrian-Level Wind

POD – Proper Orthogonal Decomposition

RANS – Reynolds-Averaged Navier-Stokes

RMSE – Root Mean Square Error

RNN – Recurrent Neural Network

ROM – Reduced-Order Model

SGMS-GNN – Sub-Graph-Partitioned Multi-Scale Graph Neuron Network

SGS – Subgrid-Scale

SP-DMD – Sparsity-Promoting Dynamic Mode Decomposition

SVD – Singular Value Decomposition

TKE – Turbulent Kinetic Energy

UAM – Urban Air Mobility

UAV – Unmanned Aerial Vehicle

UHI – Urban Heat Island

URANS – Unsteady Reynolds-Averaged Navier-Stokes

WRF – Weather Research and Forecasting

eVTOL – Electric Vertical Take-Off and Landing

optDMD – Optimized Dynamic Mode Decomposition

Chapter 1: Introduction

1.1 Background

The rapid trend of urbanization has transformed cities into densely populated areas, leading to an increase in tightly packed high-rise structures to accommodate the growing population. Urban wind flow refers to the movement and behavior of wind within city environments, shaped primarily by the presence of buildings, streets, and other structural elements [1]. In densely populated urban areas with narrow street canyons, this phenomenon significantly affects turbulent flow and modifies flow field conditions, which can obstruct the natural ventilation of the city, contributing to the urban heat island (UHI) effect and exacerbating urban air pollution [2]. These issues may influence factors such as building energy consumption and residents' comfort and well-being [3]. Such effects play a critical role in shaping important planning and design choices within urban environments. Moreover, the development of urban air mobility (UAM) concepts has sparked significant investment from both research and industry in real-time urban wind prediction [4]. This effort aims to enhance air traffic management and ensure the safety of urban flight operations. Among the methods proposed in the literature, the field measurement enables online monitoring of wind conditions by collecting wind data using sensors. However, deploying many sensors to monitor an entire urban wind flow field is often impractical. In this context, a primary goal is to achieve real-time reconstruction of urban wind flow fields using a limited number of sparsely distributed wind sensors throughout the city. This approach not only advances UAM design but also enables timely warnings to mitigate potential urban wind-related hazards.

1.2 Problem Statement

Urban wind flow modeling is crucial for enhancing the planning and design to improve pedestrian comfort and foster more sustainable living environments. Specifically, in time-sensitive urban emergency management and the enhancement of UAM operations, real-time reconstructing and predicting urban wind fields at the micro-scale becomes even more critical. Currently, three methods are used for modeling urban wind flow namely Computational Fluid Dynamics (CFD), field measurements, and wind tunnel testing. However, there are challenges associated with each approach.

CFD models offer high-fidelity simulations; but their high computational demands often hinder real-time and long-term analysis, limiting their practicality for precise online monitoring. Field measurements and wind tunnel experiments, on the other hand, provide real-time wind flow data through sensors. Their application in large-scale urban wind studies is often limited by high costs, logistical challenges, and time constraints. Moreover, deploying extensive sensor networks across vast urban areas is generally impractical, making comprehensive wind pattern monitoring difficult.

In response to the limitations of conventional high-fidelity simulations and data-intensive models, researchers have increasingly explored compressed sensing (CS) and sparse reconstruction techniques, which exploit the inherent sparsity of wind field data in specific transform domains [5]. By integrating distributed sensor networks, reduced-order modeling (ROM), and data-driven frameworks, these methods aim to facilitate near real-time reconstruction and monitoring of urban wind fields. However, despite their promise, these approaches face several key challenges, including sensitivity to sensor noise, dependence on optimal sensor placement,

difficulty in maintaining accuracy under varying flow conditions, and computational trade-offs between reconstruction speed and fidelity [6].

Considering the above-mentioned challenges, the current thesis introduces a sparse reconstruction framework designed to reconstruct urban wind fields in real time from a limited number of sensor measurements.

1.3 Research Objectives and Contributions

The primary contribution of this study lies in advancing fast, high-resolution techniques for reconstructing and predicting urban wind flow fields. Notably, the proposed approach is initially validated using synthetic data generated from CFD simulations around an isolated high-rise building, rather than real-world wind velocity measurements. The key contributions of this thesis are as follows:

(1) **Implementation of the ROM technique**: Sparsity-Promoting Dynamic Mode Decomposition (SP-DMD) [7] is employed to full-state wind velocity data to extract the most significant spatial modes and their corresponding temporal dynamic coefficients.

(2) **Development of a deep learning-based framework**: A Multi-Task Learning (MTL) [8] approach is adopted for wind velocity data processing. This method simultaneously maps sparse sensor measurements to the dynamic coefficients (growth/decay rates and oscillation frequencies) of each mode identified through the SP-DMD.

(3) **Enhancing the MTL performance through an encoder-decoder method:** A technique is proposed to enhance the MTL model by reducing the number of features in the network's output layer. This involves filtering out zero values from the sparse dynamic coefficient,

removing one side of mirrored values relative to the real axis, and separating the real and imaginary components to create the separatable task approach.

(4) **Analysis of the temporal evolution of wind sensor measurements** – A time series analysis is conducted on sensor data using a Bidirectional Long Short-Term Memory (BI-LSTM) [9] network to capture and model temporal dependencies in wind sensors signals.

(5) **Evaluation of the proposed framework** – The overall performance of the framework, including the integration of the previous approaches, is assessed based on real-time reconstruction error and prediction accuracy for full-state velocity fields.

(6) **Sensor Location Scenarios and Data Augmentation Techniques for Training a More Robust Time Series Model** – Two different sensor location scenarios are considered to evaluate model performance under varying sensor's locations. Additionally, a data augmentation technique is applied by introducing Gaussian noise to the training dataset to enhance its robustness and improve forecasting accuracy.

1.4 Thesis Outline

The remainder of the thesis is organized as follows:

Chapter 2 provides a literature review, focusing on existing techniques for studying wind flow fields in urban areas, with particular emphasis on data-driven techniques for fast reconstruction of urban wind fields.

Chapter 3 outlines the proposed methodology, detailing the mathematical intuition and the general framework of the integrated approach.

Chapter 4 presents the case study, detailing the CFD domain data, the domain of the sampled data, sensor locations for each scenario, data preprocessing steps, and the hyperparameters used for implementing each model.

Chapter 5 presents the study's results and includes an in-depth discussion of the findings.

Chapter 6 concludes the thesis by summarizing the research, discussing its limitations, and suggesting directions for future work.

Finally, the Appendix section presents the Python implementation codes developed for this thesis.

Chapter 2: Background and Literature Review

This chapter presents a comprehensive review of the literature, highlighting the significance of urban airflow modeling across various sectors and examining the methods explored in previous studies. Section 2.1 presents an application-based review of the literature, analyzing the impact of wind fields on various sectors. Section 2.2 evaluates traditional methodologies for modeling urban wind flow and their applications. Section 2.3 explores data-driven approaches for fast reconstruction and prediction of urban wind flow. Section 2.4 will explore the limitations and gaps in the literature.

2.1 Applications of Urban Wind Flow Modeling

A report by the United Nations Department of Economic and Social Affairs projects that the global urban population will increase from the current 58.3% to 68.4% by 2050, with the most significant growth anticipated in developing countries (Fig. 1) [10]. This implies that cities will see an increase in densely packed high-rise structures to accommodate the growing population.

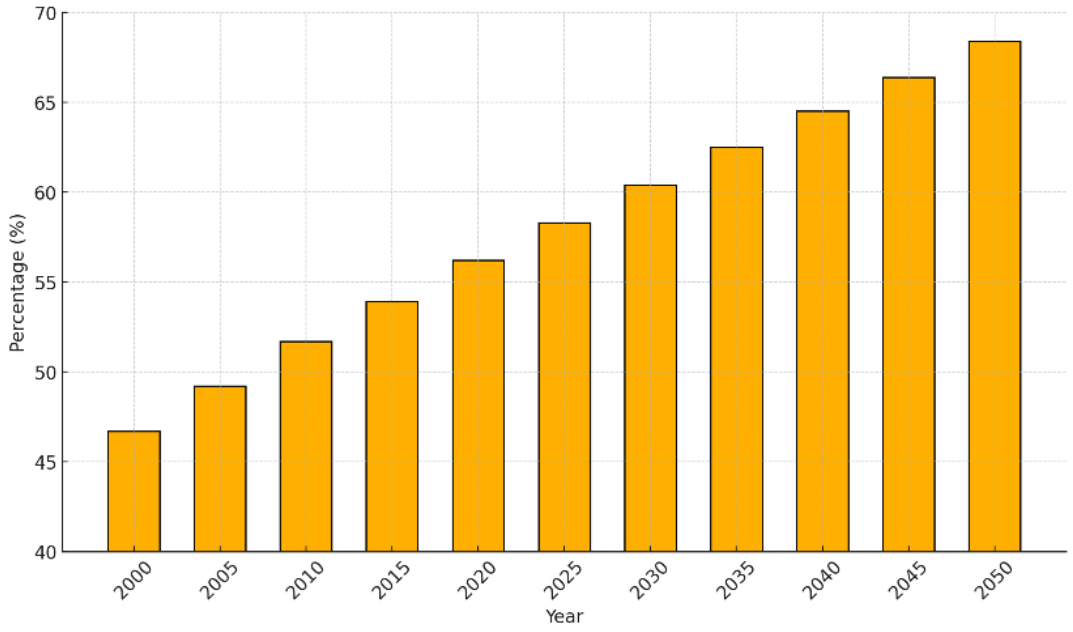


Figure 1: The Share of Global Population Living in Cities [10].

Analyzing wind flow patterns and conducting real-time monitoring of flow fields in densely populated cities are essential for sectors that play a pivotal role in shaping urban planning and design decisions [11]. These analyses are essential for a wide range of applications, including air quality management, enhancement of thermal comfort and safety, optimization of building energy performance, evaluation of renewable energy potential, improvement of urban microclimatic conditions, and the design of more efficient urban air mobility (UAM) systems. In the following, each of these key sectors will be examined based on insights from the existing literature.

Figure 2: Urban Wind Impacts on various sectors.

Wind is a key factor in pollutant dispersion in urban areas, especially in street canyons. Literature highlights that wind speed, direction, building geometry, and urban morphology significantly influence this process [12-14]. Higher wind speeds, can force pollutants upward, limiting their concentration at ground level. However, the direction of the wind is equally important; for instance, in long street canyons, air exchange rates peak at certain angles (e.g., 30°) and drop at others (e.g., 90°) [15]. Efficient monitoring enables 14more accurate prediction and management of air quality, promoting natural air ventilation in cities and fostering healthier living environments [16]. Research indicates that building height, density, and enclosure significantly

influence wind patterns, with high-density urban layouts often creating static zones where pollutants tend to accumulate. Conversely, optimal urban forms such as stepped building heights and interconnected green spaces enhance airflow and improve pollutant dispersion [17].

Understanding wind patterns helps architects and planners design buildings and public spaces that enhance pedestrian comfort and safety. For instance, knowledge of wind flow can prevent the creation of urban street canyons which make pedestrian areas uncomfortable or hazardous [13]. Tall buildings can significantly increase wind speeds at ground level, leading to discomfort or danger, particularly near corners and passages. A study found that buildings over 400 m tall with width modifications at the 1/3 height level have the greatest impact on pedestrian-level wind (PLW) conditions [18]. Modifying building layout, orientation, and tree placement plays a crucial role in shaping outdoor wind conditions and enhancing pedestrian thermal comfort. For instance, features like canopies and podiums can significantly lower both area-averaged and peak pedestrian-level wind (PLW) speeds around high-rise buildings, while permeable floors help improve overall PLW comfort [19]. Research also indicates that wind barriers and porous screens effectively reduce excessive wind speeds while preserving airflow, whereas lift-up building designs enhance pedestrian ventilation but can lead to high wind speeds in open areas [20]. Moreover, buildings with long facades aligned parallel to the prevailing wind direction can generate favorable wind conditions, while configurations featuring a central square space oriented towards the prevailing wind offer optimal thermal comfort [21].

Urban wind environments significantly influence building energy consumption by altering infiltration rates, convective heat transfer, and indirect factors like occupant behavior and pollutant dispersion [22]. Optimized urban planning is essential for improving energy consumption and enhancing building efficiency in urban settings. For example, research indicates that wind speed

has a greater impact on heat dissipation from building walls than wind direction, with maximum dissipation occurring when the walls are positioned perpendicular to the wind [23]. This affects heat transfer, influencing heating loads and thus building energy efficiency, particularly in winter conditions. Another study presents an easy-to-use empirical approach for quickly estimating the viability of wind-driven natural ventilation at the earliest stages of building design to reduce building energy costs [24]. As mentioned, specific wind angles influence pollutant dispersion and natural ventilation in urban areas. Research also shows that certain angles create optimal ventilation flow paths, helping to lower indoor temperatures during hot months while minimizing heat loss in colder periods [25].

Recent investigations on urban wind energy potential, highlights various approaches to enhance small-scale wind utilization in built environments, focusing on both physical and economic aspects. Accurate wind data is essential for optimizing the placement and performance of urban wind turbines, playing a crucial role in advancing sustainable energy solutions within cities [26]. By studying wind flow patterns in urban environments, researchers can pinpoint ideal turbine locations and evaluate their potential for power generation. Additionally, urban planning strategies that account for building shape, spacing, and roof geometry can enhance the long-term effectiveness of wind energy integration in dense urban environments [26, 27]. In addition to locating the optimal location for urban wind turbines to maximize efficiency, Anup Kc et al. [29] investigated the effects of turbulent flows on the small wind turbine performance and reliability. They showed that the lower average wind speeds and elevated turbulence in cities significantly degrade urban turbine performance compared to open-field or rural areas.

Monitoring wind effects aids in assessing and mitigating UHI and other microclimate challenges, enhancing the resilience of urban areas to climate change. Ku et al. [30] similarly used

CFD simulations of multiple hypothetical building configurations, demonstrating that changes in building height, density, and layout can substantially alter ventilation potentials and lead to localized hot spots of turbulent flow.

Air taxis and drone deliveries are poised to become the future of urban transportation and logistics in next-generation cities. Recent advancements in UAM underscore the need for real-time and precise prediction of wind flow around buildings, which can create significant turbulence and dangerous conditions for passenger electric vertical take-off and landing (eVTOL) aircraft and small delivery drones. A variety of studies has addressed this challenge through high-resolution simulations, experimental investigations, and data-driven methods, all aimed at improving safety and reliability.

A key component of prior research on UAM systems involves the use CFD simulations to characterize urban wind environments. For example, Giersch et al. [31] employed a large-eddy simulation (LES) to show how small-scale structures of wind shear and turbulence in urban canyons critically influence low-inertia drone flight and highlight the necessity of high-resolution meteorological databases for planning safe routes. Further study confirmed these microscale complexities by conducting wind-tunnel experiments on a 1:400 city model, indicating that rapid velocity gradients and turbulence intensities arise near building clusters, creating challenging conditions for UAM [32].

Besides pure modeling and experimentation, there is also a strong push toward coupled or hybrid techniques that fuse mesoscale forecasts with finer CFD representations. Research aimed to enhanced wind predictions by integrating the weather research and forecasting (WRF) model for large-scale atmospheric forcing with CFD simulations to refine near-building flow dynamics

[33]. Such an approach was validated against local sensor data and found to capture gustiness and building-induced flow accelerations more accurately than standalone mesoscale simulations.

On the operational side, researchers have explored ways to incorporate these refined wind estimates into mission planning and flight management. For instance, a study formulated a robust optimization strategy for drones delivering parcels to remote islands, showing that factoring in uncertain wind speed and direction yields schedules that minimize delivery risks and costs [34]. For UAM traffic in urban corridors, Balazova et al. [35] proposed low-cost on-board measurement strategies using sonic anemometers and eddy-dissipation-rate algorithms, which identify dangerous vortices and building-induced eddies in near-real time. Additionally, another operationally oriented technique leverages machine learning and data assimilation (DA) methods. In this regard, researchers aimed to integrate the high-resolution simulations with a long short-term memory (LSTM) neural network, producing short-range (up to six hours) nowcasts of wind speeds and gusts, essential for dynamic route planning [36]. Moreover, same group demonstrated how variational DA with a ROM can blend observational sensor inputs into CFD wind fields, thereby improving hazard identification in the turbulent wakes around buildings [4].

Across these varied approaches researchers converge on the same core finding that urban wind fields are highly localized and dynamically complex, requiring high-resolution tools to ensure safety for urban flights. Successful UAM operations and certification frameworks will likely depend on combining reliable computational models with real-time DA and carefully placed meteorological sensors, so that micro-scale wind phenomena, such as recirculation or rooftop vortices, are captured and integrated into flight operations [37]. Collectively, these studies underscore the critical role of high-fidelity modeling, experimental validation, and operational data

synthesis in defining safe corridors, vertiport locations, and flight procedures for next-generation air mobility within the urban environment.

2.2 Methods of Reduced-Order Modeling in Urban Wind Field Analysis

The increasing complexity of systems modeled in modern scientific and engineering domains has led to a significant reliance on high-fidelity numerical simulations. These simulations are capable of capturing intricate details of physical phenomena but often demand substantial computational resources and time. This creates a practical limitation for many applications, particularly those requiring iterative analyses or real-time predictions. Particularly relevant to the present study are online urban-scale wind modeling scenarios, with applications in UAM systems and emergency response operations serving as critical use cases [38].

ROM is a method used to simplify complex, high-dimensional and high-fidelity models by reducing their computational complexity while preserving essential system behavior. This approach is essential for enabling real-time simulations, optimization and control in fields such as CFD, structural mechanics, and control theory, where full-order model (FOM) can be prohibitively expensive due to their high dimensionality and computational demands [39]. By reducing the model order, ROM achieves substantial computational savings while maintaining sufficient accuracy for practical applications, making it indispensable for addressing modern engineering challenges. In general, ROM methods are categorized into model-based and data-driven approaches [40].

Model-based methods, often referred to as intrusive ROM (IROM) techniques, require direct access to the governing equations or the solver of the FOM, such as the discretized residual of the partial differential equations [41]. These methods are considered physics-informed, as they

explicitly integrate known physical laws, governing equations, and structural properties of the system into the model reduction process. By projecting the governing equations of a high-fidelity FOM onto a carefully selected low-dimensional subspace, these techniques achieve significant dimensionality reduction while preserving the system's dominant dynamics. As a result, they reduce reliance on extensive FOM data, enabling more efficient simulations without sacrificing essential physical accuracy.

A prominent example of model-based ROM is the Galerkin projection method. The accuracy of Galerkin-based ROMs is highly dependent on the quality of the reduced basis, which is typically constructed using proper orthogonal decomposition (POD) to extract the most energetic modes from a set of solution snapshots [42]. Alternatively, greedy algorithms can be employed to iteratively enrich the reduced basis, guided by rigorous a posteriori error estimator to ensure convergence and reliability. Despite its advantages, the Galerkin projection method faces limitations in handling highly nonlinear or multiscale systems, where error bounds may not be readily available and convergence can be slow [40].

Data-driven model order reduction methods often referred to as non-intrusive ROM (NIROM), conversely, do not require access to the governing equations of the system. Instead, they rely purely on data from the full model (input/output data), such as high-fidelity simulation data or experimental measurements, and they function as black-box or grey-box models [43]. In the context of urban-scale wind field modeling, data-driven ROM approaches have gained considerable traction. Techniques like POD and Dynamic Mode Decomposition (DMD) variants, both employ linear approximation to extract low-dimensional representations from high-dimensional flow data [44-47]. More recently, advanced methods incorporating various deep learning architectures techniques have emerged, offering enhanced flexibility and scalability

[33,48-51]. Hybrid approaches that combine these methodologies leveraging the strengths of both linear projection based and deep learning paradigms are also being actively explored to improve reconstruction accuracy and computational efficiency [52,53].

Considering the inherent complexity and high dimensionality of urban wind dynamics for both experimental measurements and CFD simulations developing efficient and accurate ROM has become essential, particularly for applications in densely built urban environments. These problems often involve a large number of degrees of freedom due to complex boundary conditions, fine spatial discretization, and highly unsteady flow dynamics. In this thesis, a particular focus is placed on data-driven ROM techniques for the reconstruction and prediction of fluid flow characteristic. Specifically, the applicability and performance of these methods in the context of urban wind modeling are critically examined. The following sections explore various data-driven ROM approaches designed for dynamical systems, highlighting their advantages, limitations, and relevance to urban wind flow analysis.

2.2.1 POD for High-Speed Urban Wind Flow Modeling

A classic ROM approach is a linear approximation method known as POD, also known as principal component analysis (PCA) in statistics. In fluid dynamic application, this method can compute an orthogonal basis of modes (spatial patterns) from snapshot data that optimally ranked by the flow energy content [54]. The procedure below follows the formulation introduced by Sirovich (1987) and refined in subsequent texts (Holmes et al., 2012) [55,56] .

Let $\{q_i\}_{i=1}^{p}$ be p instantaneous snapshots of the velocity (or another state) obtained at uniform time intervals Δt from a high-fidelity CFD simulation or a time-resolved experiment. Each

snapshot is reshaped into a column vector $q_i \in R^N$, where N equals the number of spatial degrees of freedom (grid nodes × components). Collecting all snapshots yields the snapshot matrix:

$$\mathrm{X} = \left[\mathrm{q}_1,\ \mathrm{q}_2,\ \ldots\ \mathrm{q}_\mathrm{p}\right] \in R^{N\times p} \tag{1}$$

Optional centering can be applied through mean subtraction, wherein the temporal mean field is computed and removed from the dataset to focus the decomposition on fluctuating structures. The temporal mean field $\bar{q}$ can be retrieved using the following expression:

$$\bar{q} = \frac{1}{p}\sum_{i=1}^{p} q_i \tag{2}$$

Retaining $\bar{q}$ separately later permits exact reconstruction of the full field.

$$x_i = q_i - \bar{q}, \quad X_c = \left[x_1,\ x_2,\ \ldots\ x_p\right] \tag{3}$$

Formulation of the correlation operator cane be done by method of snapshots to construct the Gram matrix for $p \ll N$ and then solve the reduced eigenproblem:

$$C = \frac{1}{p-1} X_c^T X_c \in R^{p\times p}, \quad C\, v_j = \lambda_j\, v_j \tag{4}$$

This is yielding eigenvalues $\lambda 1 \geq \lambda 2 \geq \cdots \geq \lambda p \geq 0$. The spatial modes follow from:

$$\phi_j = \frac{X_c\, v_j}{\sqrt{\lambda_j\,(p-1)}} \in R^N \tag{5}$$

Alternatively, by performing singular-value decomposition (SVD):

$$X_c = U\,\Sigma\,V^T \tag{6}$$

Where $\Sigma = diag(\sigma_1, \ldots, \sigma_m)$ and the left singular vectors and $U = [\phi_1 \ldots \phi_m]$ are the POD modes, orthonormal under the chosen inner product:

$$\langle \phi_i, \phi_j \rangle = \delta_{ij} \tag{7}$$

The cumulative captured kinetic-energy ratio after retaining *r* modes is:

$$E(r) = \frac{\sum_{j=1}^{r} \lambda_j}{\sum_{j=1}^{m} \lambda_j}, \ m = min(N, p) \tag{8}$$

Temporal (modal) coefficients are obtained via orthogonal projection under the first *r* truncated rank:

$$a_j(t_i) = \langle q_i, \phi_j \rangle = \phi_j^T q_i, \ j = 1, \dots, r, i = 1, \dots, p. \tag{9}$$

Flow-field reconstruction with *r* modes is:

$$\tilde{q}(t_i) = \bar{q} + \sum_{j=1}^{r} a_j(t_i) \ \phi_j \tag{10}$$

POD has been widely used in application of urban wind modeling, with the purpose to extract dominant wind flow patterns or coherent structures from either simulation or experimental data [57]. Among the literatures, the combination of classic spectral methods with modern interpolation schemes has been notable for efficiently simulating nonstationary wind fields. For example, the spectral representation method can be advanced by introducing a POD-based interpolation procedure by using only a small number of time-frequency points [58]. This approach significantly accelerates nonstationary wind-field simulations while retaining strong accuracy. More recent endeavors into spectral decomposition concentrate on wind flow and structure interactions around buildings, such as the extended spectral POD (ESPOD) method, which clarifies the relationship between flow patterns and wind load components on tall structures [59].

LES data over real urban terrain used by Liu et al. [60] to examine the energetic coherence structures by adopting POD method, highlighting that a small subset of modes (on the order of 80–100) can capture most of the turbulence energy. Such insights underpin rapid environment assessments that become ever more critical in large-scale cities. Furthermore, research showcased that a "gappy POD" reconstruction from sparse sensor data yields wind-field estimates sufficiently accurate to inform path planning algorithms for energy-efficient unmanned aerial vehicle (UAV) trajectories [61]. Moreover, it was verified that while the ROM can accurately reproduce the overall flight trajectories of advanced aerial mobility craft, certain extreme maneuvers, such as a small drone encountering sudden turbulence, still require high-fidelity details from full LES [57].

2.2.2 Exploring DMD Variants and Their Role in Urban Wind Flow Modeling

DMD was first introduced by Peter J. Schmid [62] as a data-driven technique for analyzing complex fluid flow data. In this work, DMD was proposed as an equation-free method to extract spatial-temporal patterns from time-resolved measurements, without requiring governing equations. This method excels at extracting physically interpretable spatial modes from time-resolved flow-field data and provides a linear model describing how the amplitudes of these modes evolve over time [63]. The method was inspired by the need to identify coherent structures and dynamic behaviors in fluid flows, building on prior modal analysis techniques (like POD) but aiming to incorporate temporal dynamics into the decomposition as well. Therefore, this method not only has the capability to extract the coherent spatial structures (modes) in the dynamical system but also have the ability to reveal the temporal evaluation (growth/decay and frequency) of each spatial mode.

A fundamental aspect of DMD is its strong connection to Koopman operator theory [64]. This theoretical link allows DMD to effectively linearize and analyze the dynamics of highly nonlinear

systems, such as those governed by the Navier-Stokes equations, in an infinite-dimensional function space. This capability is particularly relevant for the inherently nonlinear nature of urban wind flows. The DMD method separates high-dimensional data into spatial and temporal components. The spatial structures are represented by DMD modes, the initial conditions are encoded in the mode amplitudes, and the temporal evolution is described by the dynamic coefficients. As illustrated in Fig. 3 the procedure begins with system snapshots, where the velocity fields are reshaped into a two-dimensional spatiotemporal grid.

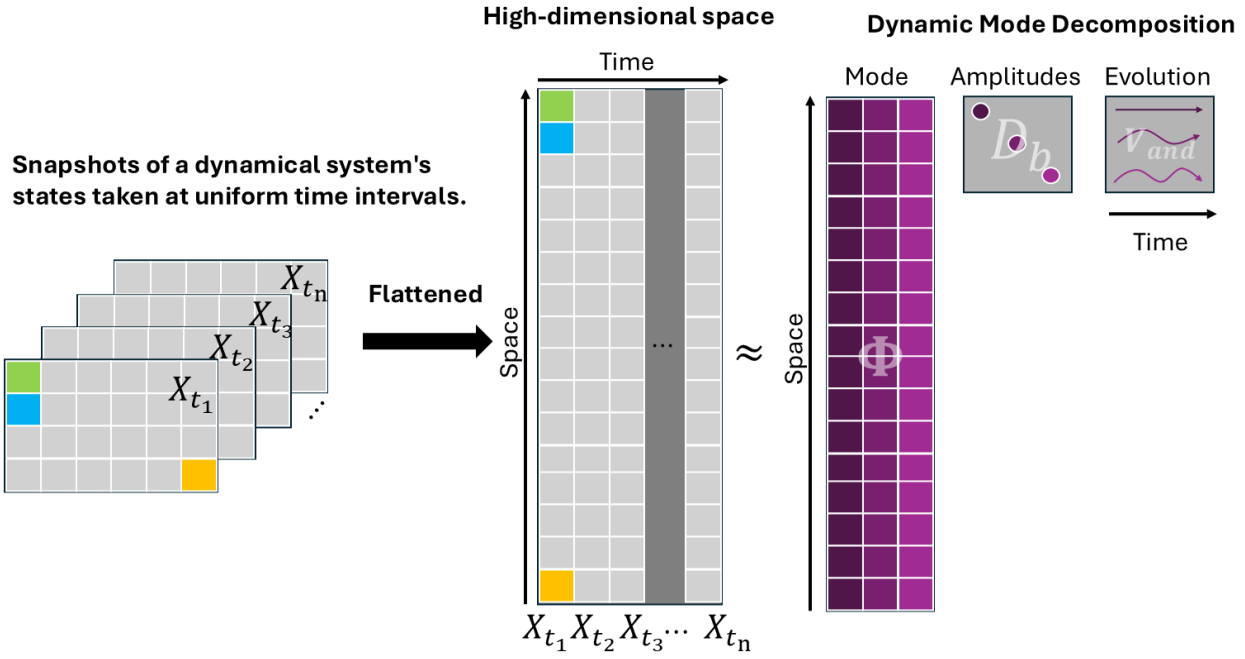


Figure 3: Illustration of the DMD method.

Experimental validation further demonstrates the strengths of DMD that shows these methods has the capability to extract coherent flow structures and their associated frequencies directly from measurement snapshots [65]. The versatility and robustness of DMD have been significantly enhanced through the development of numerous variants, each addressing specific challenges or expanding its applicability. The continuous development and refinement of these variants represent

a crucial evolution in the applicability of DMD to real-world data, particularly for complex systems like urban wind fields.

Urban environments are characterized by strong nonlinearities, multiscale phenomena, and data that can be noisy, sparse, or non-uniformly sampled. The specialized DMD variants directly address these complexities, making the technique more powerful and reliable for modeling flow around buildings, dispersion of pollutants in city environments, and urban microclimates. This trend suggests that the methodological tools within the DMD framework are maturing to a point where they can effectively meet the demanding requirements of urban climate science, paving the way for more sophisticated, accurate, and practical analyses.

A principal development is the Extended DMD (EDMD), where a dictionary of scalar observables (e.g., polynomials, Fourier functions) is used to capture richer nonlinear behaviors than standard DMD [66]. EDMD recovers approximate Koopman eigenvalues and eigenfunctions, demonstrating convergence to a Galerkin solution of the Koopman operator given suitably large data and a well-chosen dictionary.

To effectively capture general periodic, quasi-periodic dynamics, and transient behaviors that decay toward attractors, the higher order DMD (HODMD) method has been developed [67]. By leveraging time-lagged snapshots, HODMD extends the capabilities of standard DMD, enabling the identification of more intricate and subtle dynamic features. This enhanced capacity is particularly valuable for modeling quasi-periodic phenomena such as vortex shedding and other unsteady flow behaviors commonly observed in complex urban environments.

HODMD was applied to a high-fidelity LES of flow between two buildings, revealing the formation and breakdown of an arch vortex which is a key structure affecting pollutant dispersion

in urban street canyons [68]. The study showed that the presence of this vortex increases pollutant concentration in the leeward region, offering valuable insights into urban ventilation and air quality. These findings underscore the importance of dynamic flow analysis using DMD for advancing urban sustainability and environmental wind engineering.

To address challenges in selecting a parsimonious subset of DMD modes, the SP-DMD method is proposed [7]. This method aims to find the trade off between the reconstruction error and the number of modes retained by using an L1-regularized formulation for mode amplitudes. Their study demonstrated that the number of active modes can be optimally reduced in original DMD method. Moreover, the findings showed that, in the standard DMD algorithm, the set of active modes can be optimally reduced to retain only the most dynamically significant ones that still ensuring accurate data reconstruction. They suggested that the associated optimization problem can be addressed by the Alternating Direction Method of Multipliers (ADMM) [69].

Considering that complex data often contain dynamics operating at distinct temporal scales, multiresolution DMD (MRDMD) was introduced [70]. This approach is a recursive extension of DMD that hierarchically filters snapshots to isolate slow-background and fast-foreground modes; the benchmark tests shows that the method cleanly disentangles multiscale structures that defeat traditional DMD or POD. Most recently, the MRDMD mode library is used as the input to a greedy QR-pivot routine, yielding physics-aware sensor locations that capture intermittent phenomena with far fewer measurements that can outperform the POD-based sampling strategies [71].

This method was subsequently extended to urban wind applications, addressing both wind flow dynamics and pressure distributions on urban structures. By integrating a signed distance function with an optimization algorithm, the approach enables strategic sensor placement using a minimal number of sampling points, while still achieving high-fidelity reconstruction of complex

urban wind fields [72]. Similarly focusing on air quality, MRDMD has been utilized to identify multiscale patterns in pollutant concentration, specifically targeting PM2.5, to inform optimal sensor placement strategies. This approach demonstrates how an enhanced sensor network can effectively capture both short- and long-term variations in air quality, thereby improving monitoring and management in urban environments [73].

To ensure reliable and sampling-independent DMD results in urban wind engineering, recent studies emphasize the importance of proper data preprocessing and acquisition. Key practices include mean flow removal, statistical stationarity, sufficient sampling duration, and high temporal resolution [45]. These measures help stabilize DMD eigenvalues and ensure that extracted modes accurately capture the physical flow dynamics, making them essential for effective application of DMD to real-world urban wind data.

Optimized DMD (optDMD) reformulates DMD as a variable-projection least-squares problem which can enable the selection of a specified number of modes, while preserving accurate growth and decay rates which is an advantage for ROM of a complex flows [74]. This method highlights optDMD as a lightweight, interpretable, and mathematically robust tool suitable for real-time urban air quality modeling. When applied to global atmospheric chemistry and enhanced with time-shifting and ensemble bagging, optDMD demonstrated the capability to stably reconstruct and forecast pollutant fields over a 20-day period, outperforming classical DMD [44].

Emerging research extends DMD to incorporate physical constraints and handle uncertain data through Physics-Informed DMD (PIDMD) [75]. This approach integrates known symmetries, boundary conditions, and conservation laws as manifold constraints on the linear operator, resulting in models that are both physically consistent and robust to noise. By leveraging prior

physical knowledge, PIDMD can yield more accurate and efficient models, requiring less data and computational effort than standard DMD, particularly in complex urban flow scenarios.

Although these methodological advancements are significant, DMD remains highly sensitive to data resolution, noise, and waveform shape, which has led to thorough examination. A large-scale error analysis is conducted in canonical shear flows, demonstrating that non-sinusoidal instabilities (such as square waves) lead to substantially higher measurement errors in growth-rate estimations [76]. They emphasize that signal-to-noise ratio, snapshot density, and ensemble averaging are crucial factors for reliability.

The ability of DMD and its variants to extract coherent spatiotemporal patterns from complex data makes them particularly well-suited for urban flow analysis. Researchers are increasingly leveraging these methods to gain deeper insights into urban wind phenomena and to develop efficient reduced-order models for city-scale simulations.

2.2.3 Deep Learning Based ROMs for fast modeling urban wind fields

Data-driven neural network ROMs (DDNN-ROM), classified under the NIROM category, represent a significant advancement in computational modeling. By approximating FOMs through projection onto a low-dimensional manifold, these models enable accelerated simulations suitable for real-time applications or many-query tasks such as optimization, control, and uncertainty quantification. A key advantage of DDNN-ROM approaches over traditional data driven ROMs (e.g., POD, DMD) lies in overcoming the limitations of linear approximation subspaces, which often struggle to capture highly nonlinear dynamics and can suffer from stability issues. These models are trained on historical simulation data or real-world measurements to predict dynamics in real time.

By leveraging architectures such as convolutional neural networks (CNNs) [77], Fully connected neural networks (FCNNs) [51], convolutional autoencoders (CAEs) [78,79], physics-informed neural networks (PINNs) [80,81], researchers have achieved significant improvements in both prediction speed and accuracy. In addition, online learning algorithms facilitate continuous model updates as new sensor data become available, thereby refining predictions and adapting to changing urban conditions. In parallel, recent developments in transformer-based architectures have begun to gain traction in the modeling of physical systems, offering particular promise for forecasting urban wind dynamics [49,50,82-84].

One line of work embeds LES snapshots in a non-linear latent space learned by three-dimensional CAEs and then predicts latent dynamics with gradient-boosting regressors [48]. Using this strategy, this study achieved four-week predictions of a 150 km² Beijing domain at 10 m resolution in minutes. Recent studies show that NIROMs can further reduce the computational cost of LES for high-rise building airflow prediction by learning non-linear latent spaces with CAE variants (standard, multiscale, and self-attention) and modeling their temporal dynamics using LSTM networks [78]. Among them, the self-attention CAE achieves the highest accuracy, capturing over 94% of turbulent kinetic energy, reducing error by half, and preserving key flow features, all while cutting computational time. Moreover, an autoencoder-based ROM can be combined with a separate state-estimation network to map sparse data into the autoencoder's latent space, then rapidly decode the full wake flow based on the sparse measurements [51]. This approach achieved speedups of over 70% compared to standard shallow decoders, while retaining high robustness to sensor noise.

Collectively, these studies converge on hybrid CFD and deep learning or reduced-order frameworks that promise practical turnaround times, but they also expose persistent gaps in cross

climatic generalisability, turbulence-model dependence, and data demands issues that future work must address through harmonised benchmarks and rigorous cross-validation across scales and sites.

2.2.4 Physics-Informed ROMs Approaches

Recent advances in PINNs and DA techniques have shown significant potential in improving the fidelity and efficiency of urban wind flow reconstruction and forecasting. PINNs are a class of deep learning frameworks that embed physical laws (typically PDEs) such as the Navier-Stokes equations into the loss function of neural networks [80]. Unlike traditional black-box models, PINNs do not rely solely on data which they incorporate governing equations as soft constraints during training, enabling robust generalization from sparse data and adherence to physical principles. This is particularly advantageous in urban environments where sensor data are often limited and unevenly distributed. PINNs have been successfully applied to laminar and turbulent flow problems, and recent work has extended their applicability to complex domains, such as flows around buildings [85,86,87]. A promising development in urban wind modeling is the use of physics-informed graph neural networks (PIGNNs), such as the PIGNN-CFD model [85]. This model is developed to incorporate RANS equations into a graph neural network trained on unstructured CFD mesh data. This method achieves speed-ups of one to two orders of magnitude while preserving CFD-level accuracy. Another research proposes a physics-informed, graph-assisted auto-encoder that reconstructs high-resolution urban wind fields from sparse sensor measurements [86]. The model couples a deep graph neural network (well-suited to irregular sensor point clouds) with a loss term enforcing mass-conservation, thereby embedding the fluid-continuity equation directly into training. This hybrid treatment of spatial relationships and physical laws yields roughly a 50 % reduction in root-mean-square error relative to a purely data-

driven generative baseline across multiple wind directions, underscoring the gains in stability and accuracy obtained by integrating governing equations into learning architectures for urban flow prediction.

In summary, PINNs provide a promising synergy for urban wind modeling by combining physical consistency with adaptability to real-world data. Ongoing research is focused on improving scalability, robustness, and integration with sensor networks, aiming toward real-time, high-fidelity digital twins of urban airflow environments.

2.2.5 Hybrid Methods for ROMs in Urban Wind Flow Applications

The application of hybrid ROMs, primarily combining deep learning frameworks with linear approximation methods such as POD and DMD, is increasingly emerging for predictive tasks, aiming to leverage the strengths of both deep learning and traditional ROM strategies.

A common hybrid strategy combines recurrent neural networks (RNNs) with the latent space of linear reduced-order models (e.g., POD, DMD). In this approach, dimensionality reduction is first performed, after which an RNN-based model predicts the temporal evolution in the reduced space. Long Short-Term Memory (LSTM) networks are frequently employed for this purpose, as they are specifically designed for sequence and time-series data [88]. Unlike conventional RNNs, LSTMs effectively address the vanishing and exploding gradient problems that hinder long-sequence learning.

For example, a study utilized a framework based on the POD algorithm and the LSTM networks, to predict the high-fidelity LES wind shear data in complex mountainous terrain [52]. Their method has the capability to effectively capture the spatiotemporal fluctuations of wind shear within a reduced-dimensional space. In addition, POD-based sensor selection, when integrated

with a CNN, enables accurate flow reconstruction using minimal data; in the case of lid-driven cavity flow, just 1000 optimally placed sensors achieved 95% reconstruction fidelity, demonstrating the effectiveness of data-driven measurement strategies. [53]. A CS reduced-order modeling framework is developed combining LSTM networks, SP-DMD, and deep neural networks to predict unsteady flow fields from sparse and noisy sensor data [89]. Applied to both laminar and turbulent cylinder flows, the method accurately reconstructs key flow features and maintains robustness under high noise levels. The approach shows strong potential for flow estimation and control, though its reliance on linear DMD limits its ability to capture nonlinear turbulence. A complementary strategy is proposed through the use of snapshot-based POD and Tucker tensor decomposition, which efficiently compress CFD snapshots, enabling the reconstruction of a 3D unsteady flow field using only 20 sparsely distributed sensors [90]. However, this method remains limited to flow states captured within the offline library, thereby restricting its prognostic capabilities.

2.3 Gaps and Limitations

Summary of the literature review indicates the following gaps in the existing studies on online reconstruction of urban wind flow fields from sparse real-time measurements:

Current reconstruction frameworks assume carefully optimized sensor layouts, yet many ideal locations are physically inaccessible in real cities. A robust approach that tolerates random or sub-optimal sensor placement (and remains stable under real-world signal fluctuations) is still lacking.

High-fidelity CFD delivers excellent accuracy but is incompatible with real-time needs, whereas most reduced-order and machine-learning models sacrifice fine-scale detail for speed.

Achieving near-CFD fidelity at operational latencies remains an open challenge, especially for transient phenomena and extreme events.

Small-scale, highly nonlinear turbulence in the “roughness zone” around buildings is poorly captured by linear ROMs. Even physics-informed enhancements have yet to reproduce the chaotic, multi-scale nature of urban flow with sufficient accuracy for safety-critical platforms such as lightweight drones.

Many models are trained on synthetic CFD data or a single city layout and show limited validation on diverse, unseen geometries or weather extremes. Performance under drastically different block arrangements, roughness distributions, or climatic conditions remains uncertain, hindering broad deployment.

For UAM systems, particularly lightly wing-loaded delivery drones, ROM-based wind fields can mis-predict control saturation events and underestimate risk envelopes. Similar issues arise when mean RANS fields replace unsteady wind inputs, leading to potentially unsafe flight-dynamic assessments. Rigorous, domain-specific validation protocols and transparent error bounds are still absent.

Collectively, these gaps highlight the need for sensor-agnostic, noise-resilient models that couple turbulence-resolving physics with real-time efficiency and demonstrate verifiable performance across heterogeneous urban and meteorological scenarios—especially where safety or regulatory compliance is at stake.

Chapter 3: Methodology

This section introduces a modeling framework developed to predict unsteady turbulent wind velocity fields based on limited sensor measurements collected downstream of a building. The chapter starts with a clear explanation of the main building blocks of the framework, preparing for the implementation sections that come in next chapter.

3.1 Framework Overview and Key Components

Similar to the most ROM approaches, this framework comprises two main phases including the offline model development phase and the online prediction phase. In the offline phase, five distinct stages are considered using offline datasets. These stages are then integrated to form a comprehensive surrogate model designed for high-precision wind fidelity prediction. The steps involved in the offline model development phase include:

Stage One: Sensor signals at the target coordinates are collected and used to train a BILSTM model that learns the temporal dynamics of the selected signals.

Stage Two: SP-DMD is employed to extract dominant flow structures and construct a reduced-order space from the high-dimensional, unsteady, and turbulent velocity field, enabling the identification of the most significant dynamic modes and their corresponding complexed-value dynamic coefficients.

Stage Three: By analyzing the characteristics of the dynamic coefficients, this latent space can be further reduced through encode-decode preprocessing. This process includes filtering out zero values from the dynamic coefficients, eliminating redundant mirrored values relative to the real

axis, and separating the real and imaginary components to establish a more structured and separable task approach.

Stage Four: The MTL network has been developed to establish the correlation between sensor measurements and the real and imaginary components of the dynamic coefficients (latent space), at each time step. This model enables accurate mapping of sensor data to dynamic coefficients, facilitating a comprehensive reconstruction of the velocity field. Fig. 4 illustrates a schematic representation of the proposed model.

During the online prediction phase, the offline-trained framework functions as a surrogate model to deliver rapid and high-fidelity urban wind forecasts. Real-time sensor measurements, which collected from the same locations as used in the offline training but representing out-of-sample data, are retrieved for this purpose. Depending on whether the goal is online monitoring or future prediction, two approaches are followed:

- For online monitoring, the real-time sensor readings are directly input into the trained MTL network to generate the dynamic coefficients, which reconstructs the flow fields at the corresponding time steps.
- For forecasting, the BILSTM network first estimates the future evolution of the sensor signals. These predicted signals are then passed to the MTL network to generate the associated dynamic coefficients, which in turn allow for the reconstruction of the complete velocity field.

Fig. 5 provides a schematic overview of the framework's prediction phase.

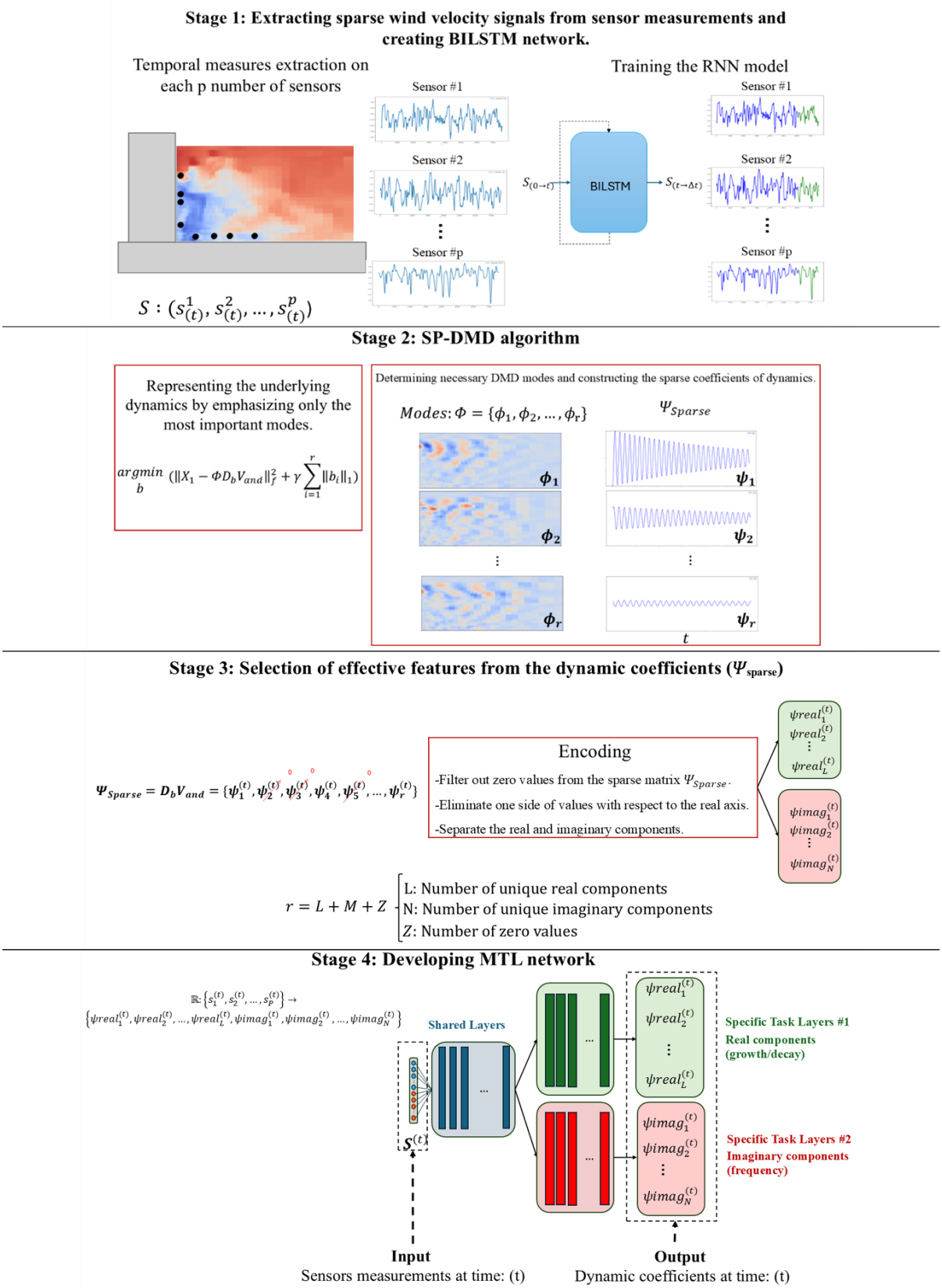


Figure 4: Architecture of the model development phase.

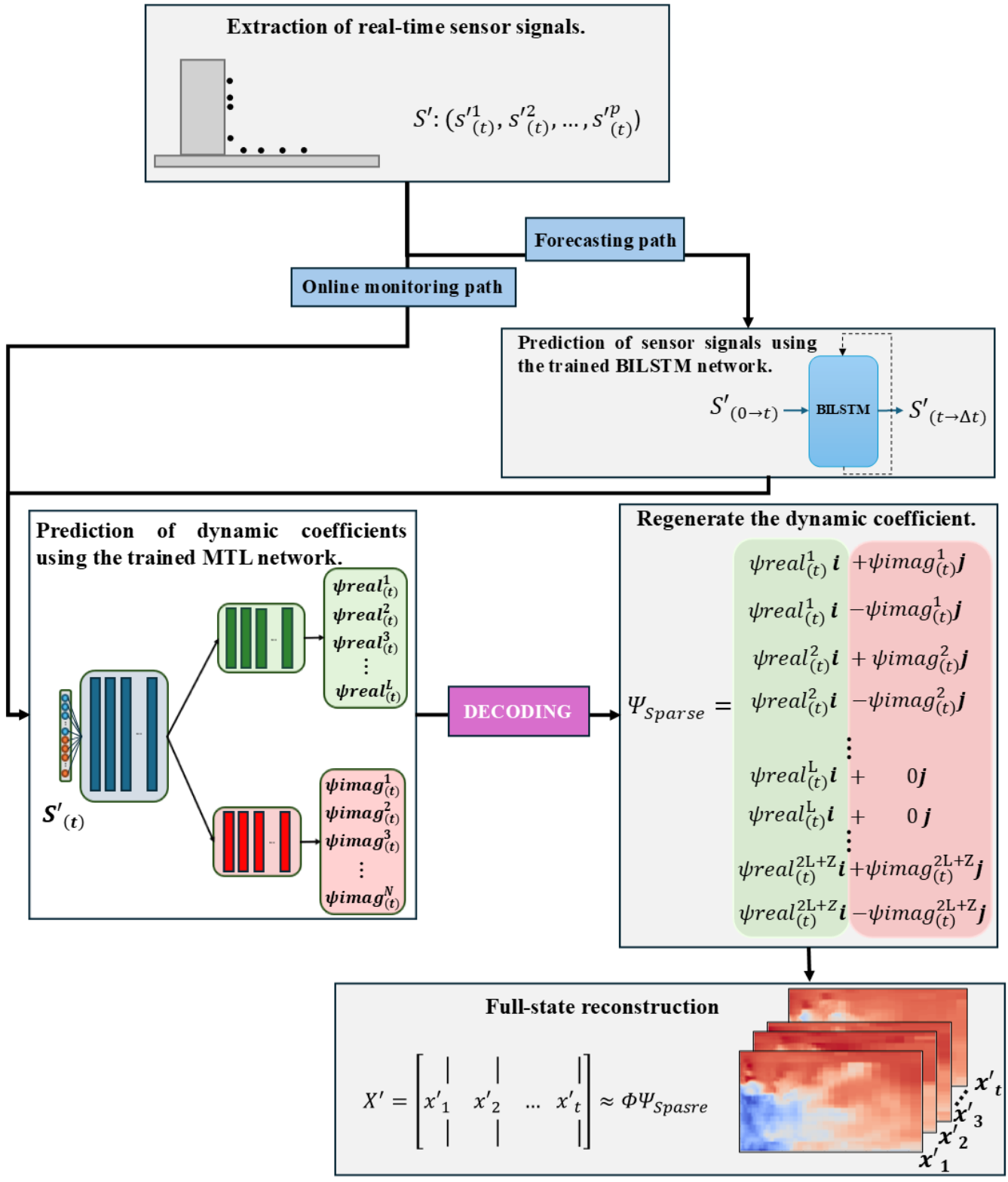


Figure 5: Architecture of the online phase for predicting the complete velocity field using out of sample sensor signals.

3.2 Predicting Sensor Signals Using a BILSTM Network

To understand the mechanism of the BILSTM network, fist the LSTM should be reviewed. At each time-step, the LSTM receives the current input vector, the previous hidden state and a persistent cell state, allowing information and gradients to flow almost linearly across hundreds of steps. Four trainable gates (forget, input, candidate memory and output) regulate this flow.

These gates decide what to keep, what to update, and what to forget, making the network much better at learning and remembering important details. The forget gate, input gate, and output gate (each utilizing a sigmoid activation function) control the flow of information in an LSTM cell by determining how much of the previous cell state is retained, how much new information is added, and how much of the internal memory is exposed to the next layer, respectively (see Eq. 11, Eq. 12 and Eq. 13).

The candidate memory gate with a tanh activation function decides what candidate values could enter memory (Eq. 14). Their element-wise combination updates the cell state through an additive operation (scaled by forget gate) that circumvents repeated multiplication and thus preserves gradient magnitude (Eq. 15). Finally, the element-wise multiplication of the cell state passed through tanh and the output gate, forms the new hidden state that the next layer will consume as shown in Eq. 16. Because the model can learn to set the forget gate near one when information must be memorised and near zero when it should be discarded, it dynamically tunes its memory span, capturing both short and long-range dependencies in data such sensor signal streams. Fig. 6 show the schematic of one LSTM cell.

$$f^{(t)} = \sigma\left(W_f x^{(t)} + R_f h^{(t-1)} + b_f\right) \tag{11}$$

$$\mathrm{i}^{(t)} = \sigma\left(W_i x^{(t)} + R_i h^{(t-1)} + b_i\right) \tag{12}$$

$$o^{(t)} = \sigma\left(W_o x^{(t)} + R_o h^{(t-1)} + b_o\right) \tag{13}$$

$$z^{(t)} = tanh\left(W_z x^{(t)} + R_z h^{(t-1)} + b_z\right) \tag{14}$$

$$c^{(t)} = z^{(t)} \odot \mathrm{i}^{(t)} + c^{(t-1)} \odot f^{(t)} \tag{15}$$

$$h^{(t)} = o^{(t)} \odot tanh\left(c^{(t)}\right) \tag{16}$$

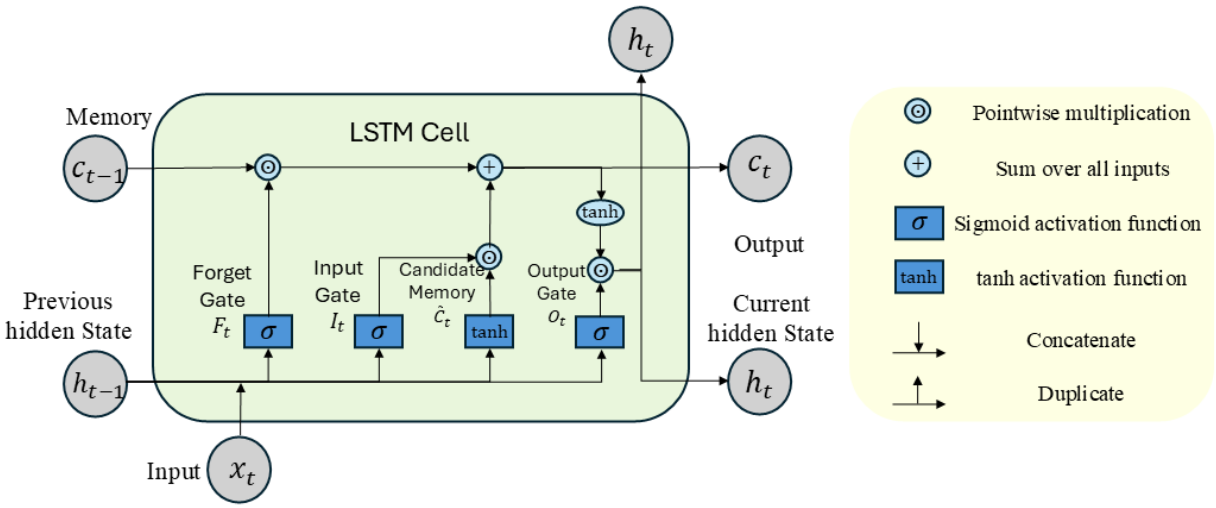


Figure 6: Illustration of an LSTM cell.

BILSTM builds on the traditional LSTM by improving the model's ability to understand context in sequence prediction tasks. While standard LSTMs process data in a single direction (typically from past to future), BILSTMs process data both forward and backward. This dual-directional approach provides the network with twice the information at any given point, leading to more accurate and nuanced predictions. Fig. 7 shows the schematic of the desired BI-LSTM network. Let $x^{(1:T)}$ be an input sequence of length T, therefore:

Forward: $$\overrightarrow{h^{(t)}}, \overrightarrow{c^{(t)}} = \mathrm{LSTM_f}\left(x^{(t)}, \overrightarrow{h^{(t-1)}}, \overrightarrow{c^{(t-1)}}\right), \quad t = 1, \ldots, T \tag{17}$$

Backward: $$\overleftarrow{h^{(t)}}, \overleftarrow{c^{(t)}} = \mathrm{LSTM_b}\left(x^{(t)}, \overleftarrow{h^{(t+1)}}, \overleftarrow{c^{(t+1)}}\right), \quad t = T, \ldots, 1 \tag{18}$$

At every position t, the two hidden states are merged by concatenation:

$$h\_bi^{(t)} = concat[\overrightarrow{h^{(t)}}, \overleftarrow{h^{(t)}}] \in R^{2H} \tag{19}$$

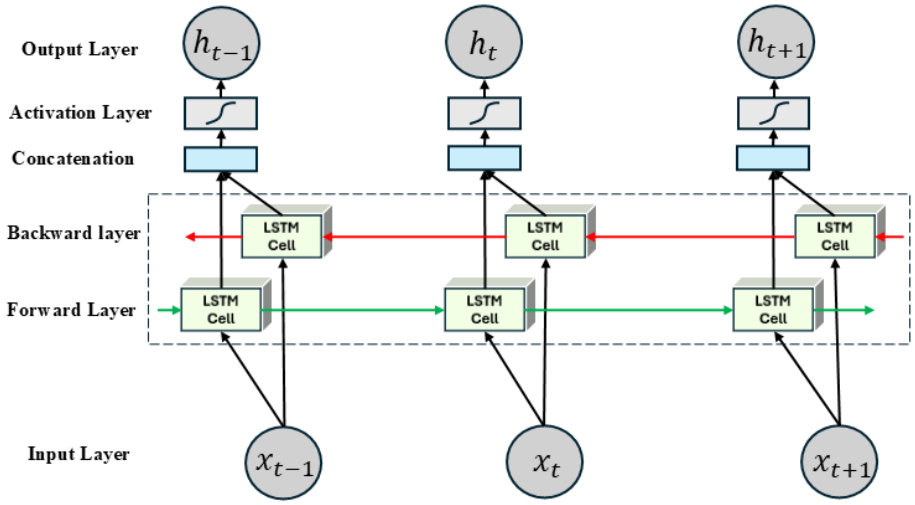


Figure 7: Bidirectional long short-term memory (BILSTM) architecture.

3.3 Sparsity-Promoting DMD

This section will discuss SP-DMD, but first, the Exact DMD will be reviewed. As discussed, DMD is a data-driven and equation-free method, designed to identify spatiotemporal coherent structures within a complex dynamical system. This approach can be used with sequential data, whether generated from numerical simulations or obtained through experimental measurements.

3.3.1 Exact DMD Algorithm

The first step involves gathering several pairs of snapshots as the system evolves over time. These snapshots capture the state of the system; for example, in this study, they represent a two-dimensional fluid velocity field measured at specific discrete points. The time intervals between each consecutive pair are consistent and small enough to capture the highest frequencies in the system's dynamics.

This field is typically transformed into a high-dimensional column vector sequence, referred to as $X \epsilon \mathbb{R}^{nxm}$:

$$X = \begin{bmatrix} | & | & & | \\ x(t_1) & x(t_2) & \dots & x(t_m) \\ | & | & & | \end{bmatrix}. \quad (20)$$

These snapshots, are subsequently organized into two data matrices, denoted as X_1 and X_2:

$$X_1 = \begin{bmatrix} | & | & & | \\ x(t_1) & x(t_2) & \dots & x(t_{m-1}) \\ | & | & & | \end{bmatrix}. \quad (21)$$

$$X_2 = \begin{bmatrix} | & | & & | \\ x(t_2) & x(t_3) & \dots & x(t_m) \\ | & | & & | \end{bmatrix}. \quad (22)$$

The goal of the DMD algorithm is to estimate the leading spectral decomposition, focusing on the eigenvalues (λ_k) and their associated eigenvectors (φ_k) of the optimal linear operator A, which captures the relationship between two snapshot matrices over time:

$$X_2 \approx AX_1. \quad (23)$$

Although these snapshots typically come from a nonlinear dynamical system, DMD provides an optimal local linear approximation. The optimal matrix A is defined as:

$$A = \underset{A}{argmin} \; \|X_2 - AX_1\|_F = X_2 X_1^{\dagger}. \quad (24)$$

Where $||\cdot||_F$ is the Frobenius norm that sums the squared residuals of every entry of the snapshot matrix and † denotes the pseudo-inverse operation. Given the potential for X to represent an extremely large dataset, which could complicate the regression task, the DMD algorithm proposes an approximation of the higher order for constructing A, focusing on selecting a reduced rank.

By performing the SVD of X_1 and selecting the first 'r' truncated singular values:

$$X_1 \approx \widetilde{U}\tilde{\Sigma}\tilde{V}^*. \tag{25}$$

Since only the leading r eigenvalues and eigenvectors of A are of interest, A can be projected onto the POD modes in $\widetilde{U}$:

$$\tilde{A} = \widetilde{U}^* A \widetilde{U} = \widetilde{U}^* X_2 \tilde{V} \tilde{\Sigma}^{-1}. \tag{26}$$

By computing the eigen decomposition of $\tilde{A}$:

$$\tilde{A}W = W\Lambda. \tag{27}$$

The entries of the diagonal matrix are the DMD eigenvalues, which also correspond to eigenvalues of the full A matrix. The high-dimensional DMD modes are reconstructed using the eigenvectors W of the reduced system and the time-shifted snapshot matrix X_2 according to:

$$\Phi = X_2 \tilde{V} \tilde{\Sigma}^{-1} W. \tag{28}$$

The columns φ_i of Φ (DMD modes) are notably the eigenvectors of the high-dimensional matrix A, associated with the eigenvalues λ_i in Λ:

$$A\Phi = \Phi\Lambda. \tag{29}$$

The state reconstruction at each time snapshot t_i can be written as:

$$\hat{x}(t_i) \approx \sum_{k=1}^{r} \varphi_k \, b_k \, \lambda_k^{t_i}. \tag{30}$$

Where r is the number of dominant DMD modes retained, φ_k are the DMD modes, λ_k are the DMD eigenvalues, b_k are the amplitudes determined by projecting the initial state (or another reference state) onto the DMD modes.

By considering the initial snapshot X_1 at time $t_i = 0$, from Eq. 25:

$$b = \Phi^{\dagger} x(t_1). \tag{31}$$

The full state approximation of flow field X_1 at all time steps can be reconstructed by:

$$\hat{X}_1 = \Phi D_b V_{and}. \tag{32}$$

Where Φ is the matrix, whose columns φ_k represent the DMD modes, capturing the coherent spatial patterns, while D_b is a diagonal matrix containing the mode amplitudes, and V_{and} is the Vandermonde matrix governs the temporal evolution of each mode. This can be written in the form:

$$\hat{X}_1 = \begin{bmatrix} | & | & & | \\ x(t_1) & x(t_2) & \dots & x(t_{m-1}) \\ | & | & & | \end{bmatrix} \approx \begin{bmatrix} | & | & & | \\ \varphi_1 & \varphi_2 & \dots & \varphi_r \\ | & | & & | \end{bmatrix} \begin{bmatrix} b_1 & 0 & \cdots & 0 \\ 0 & b_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & b_r \end{bmatrix} \begin{bmatrix} 1 & \lambda_1 & \cdots & (\lambda_1)^{m-2} \\ 1 & \lambda_2 & \cdots & (\lambda_2)^{m-2} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \lambda_r & \dots & (\lambda_r)^{m-2} \end{bmatrix}. \tag{33}$$

3.3.2 SP-DMD Method

DMD can extract low-rank spatiotemporal dominant patterns from snapshots generated by complex dynamical systems. However, many DMD modes are often required to accurately reconstruct complex turbulent flows, making it still challenging for real-world applications.

SP-DMD designed to enhance the estimation of mode amplitudes by promoting sparsity and selecting the most important modes. This method balances the number of modes and reconstruction error by applying a Lasso penalty on mode amplitudes, which enforces sparsity by shrinking some amplitudes to zero, effectively eliminating those features from the model. In this case, the reconstructed snapshots using the DMD modes, unknown amplitudes b_i, and the known time evolution carried by each eigenvalue can be written as:

$$\hat{X}_1(b) \;=\; \Phi D_b V_{and}. \tag{34}$$

Reducing the cost function is interested by solving the:

$$J(b) = \left\| X_1 - \hat{X}_1(b) \right\|_F^2 = \sum_{j=1}^{m-1} \sum_{i=1}^{n} [X_1(i,j) - \hat{X}_{1(b)}(i,j)]^2. \tag{35}$$

By solving this convex optimization problem, the sparse solution can be derived as:

$$\underset{b}{argmin}\left(J(b)+\gamma\sum_{i=1}^{r}\|b_i\|_1\right). \tag{36}$$

Here, γ is a positive regularization parameter that enforces sparsity in the vector b. Higher values of γ place greater emphasis on reducing the number of non-zero elements in b. The term $||\cdot||_1$ represents the l_1-norm penalty, which promotes sparsity in b (encouraging more zero values in D_b), which is a desired outcome. This optimization problem can be solved conveniently by using ADMM algorithm. For a detailed explanation of the settings used to solve this convex optimization problem via ADMM, the reader is referred to [7].

3.4 Preprocessing Dynamic Coefficients for Efficient MTL Network

The DMD algorithm may also be seen as connecting the favorable aspects of the SVD for spatial dimensionality reduction and the fast Fourier transform FFT for temporal frequency identification. Thus, each DMD mode is associated with a particular eigenvalue in a form of:

$$\Lambda_r = [1, (\lambda_r)^1, (\lambda_r)^2, \ldots, (\lambda_r)^{m-2}] = [1, (\alpha_r)^1 + \boldsymbol{i}(\beta_r)^1, (\alpha_r)^2 + \boldsymbol{i}(\beta_r)^2, \ldots, (\alpha_r)^{m-2} + \boldsymbol{i}(\beta_r)^{m-2}]. \tag{37}$$

Where β represents the specific oscillation frequency, and α denotes the growth or decay rate for each mode.

The dynamic mode coefficients (time-dependent amplitudes) can be obtained by multiplying the diagonal amplitude matrix with the Vandermonde matrix:

$$\Psi_{Sp} = D_{b(sparse)}V_{and} = \begin{bmatrix} b_1 & b_1\lambda_1 & \cdots & b_1(\lambda_1)^{m-2} \\ b_2 & b_2\lambda_2 & \cdots & b_2(\lambda_2)^{m-2} \\ \vdots & \vdots & \ddots & \vdots \\ b_r & b_r & \cdots & b_r(\lambda_r)^{m-2} \end{bmatrix}. \tag{38}$$

Since D_b is a diagonal matrix primarily consists of zero values, Ψ_{Sp} represents the sparse dynamic coefficients. As a result, Ψ_{Sp} contains mostly zero rows, emphasizing the temporal evolution of each selected mode.

To reduce the number of features more efficiently, this research considers the behavior of the sparse dynamic coefficient matrix to create the desired output mapping for the MTL network, with the goal of capturing the unique frequency (imaginary component) and growth/decay (real component) of Ψ_{Sp} . Specifically, the process involves:

1. Filtering out zero values from the sparse matrix Ψ_{Sp} dynamic coefficients (removing zero-valued rows).
2. Eliminating redundant mirrored values relative to the real axis (since each pair's frequency is both positive and negative of the same magnitude, and the growth/decay rate is constant for each pair).
3. Separating the real and imaginary components to establish a more structured and separable task approach for the MTL network.

Fig. 8 provides a schematic representation of the process. In Fig. 8(a), the blue points represent Ψ at specific time, obtained from the original DMD method. The red crosses indicate the selected Ψ values at initial system state after applying the SP-DMD method, yielding Ψ_{Sp}. Fig. 8(b) illustrates Ψ_{Sp} after the imposition of sparsity, and Fig. 8(c) shows the elimination of redundant mirrored values relative to the real axis, regulating the unique values for prediction task.

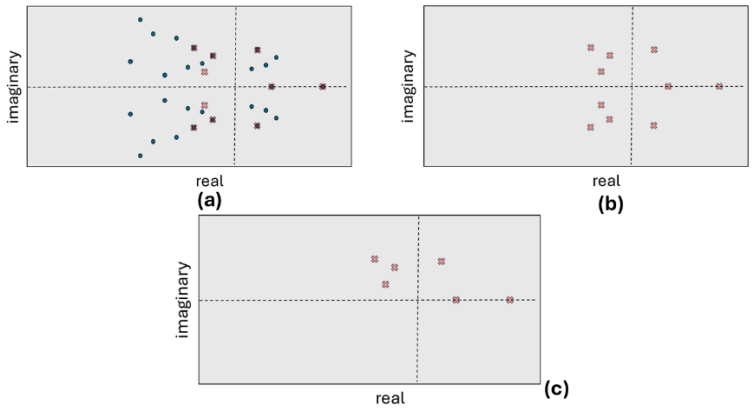


Figure 8: Illustration of the proposed sampling method for the dynamic coefficient: (a) Comparing the original DMD dynamic coefficient or Ψ (blue points) at a specific time step with the SP-DMD-derived Ψ_{Sp} (red crosses). (b) The Ψ_{Sp} values obtained after applying sparsity. (c) Elimination of mirrored values across the real axis.

3.5 Multi-Task Learning Network

Multi-Task Learning (MTL) is a machine learning paradigm where multiple related tasks are learned simultaneously using a shared model. This approach leverages the information from different tasks to improve the generalization performance of all tasks involved. By training on multiple tasks together, MTL increases data efficiency and reduces overfitting through shared representations, allowing the model to capture more robust and generalizable features. In this thesis, the goal is to find the correlation between sparse sensor measurements and the latent space of dynamic coefficients Ψ_{Sp} retrieved from the SP-DMD method, as shown in Eq. 39:

$$\mathbb{R}: \{s^1_{(t)}, s^2_{(t)}, \dots, s^P_{(t)}\} \rightarrow \{\psi real^1_{(t)}, \psi real^2_{(t)}, \dots, \psi real^L_{(t)}, \psi imag^1_{(t)}, \psi imag^2_{(t)}, \ \dots, \psi imag^N_{(t)}\} \quad (39)$$

Generally, this network is designed to handle T different tasks using a combination of shared and task-specific parameters. The shared parameters, θ_s, form a common representation layer that captures the underlying features and patterns shared across all tasks. In addition, each task t has its own set of task-specific parameters, θ_t, to capture unique features and nuances relevant to that particular task. Fig. 9 shows the suggested network.

The overall model's function, f_t can be expressed as:

$$\hat{y}_t = f_t(x; (\theta_s, \theta_t)) = h_t(g(x; \theta_s); \theta_t) \quad (40)$$

Where x is the input data, $g(x; \theta_s)$ is the shared representation layer with shared parameters θ_s, h_t is the task-specific function for task t with parameters θ_t.

For each task t (where $t \in 1,2,..,T$), the model calculates the loss based on the true values y_t, and the predicted values $\hat{y}_t$:

$$\mathcal{L}oss_t = \frac{1}{n}\sum_{j=1}^{n} \mathcal{L}(y_{t,j}, \hat{y}_{t,j}) \tag{41}$$

Here $\mathcal{L}$ is representative of the desired loss function. The total loss is computed as the weighted sum of the individual losses across all T tasks and is given by:

$$\mathcal{L}oss_{Total} = \sum_{t=1}^{T} \alpha_t \cdot \mathcal{L}oss_t \tag{42}$$

Where α_t is the weight for task t, and $\mathcal{L}oss_t$ is the individual loss for task t.

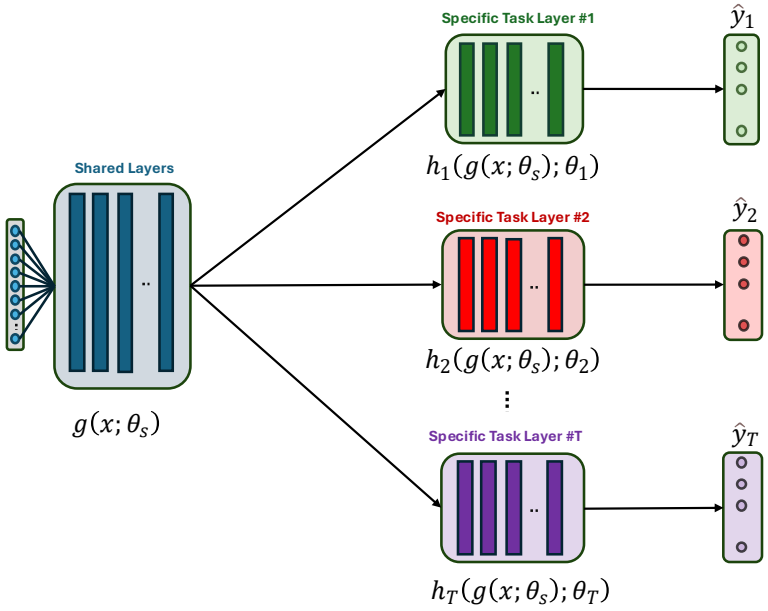


Figure 9: Schematic of the multi-task learning (MTL) network.

Chapter 4: Case Study

The proposed methodology is applied to a dataset of snapshots obtained from a three-dimensional CFD simulation, focusing on the wake flow behind an isolated high-rise building [91]. After discarding the initial transient phase, a total of 6,084 snapshots of the streamwise (U) and crossflow (W) velocity components were extracted. The snapshots are separated by a time interval of $\Delta t = 1 \times 10^{-3} s$. The data were collected on a vertical plane within the spatial domain defined by (x/H, z/H) ∈ (0, 160) × (0, 60), as illustrated in Fig. 10. The spatial grid consists uniformly spaced points with a resolution of 4 mm, capturing both U and W velocity components. For model development, the first 66% of the data (4,000 snapshots) were used in the offline training phase, while the remaining 2,083 snapshots were reserved for online prediction.

In this study, two sensor placement strategies are employed to evaluate the method's performance based on sensor locations. In both scenarios, 25 virtual sensors are used to capture streamwise and crossflow velocity signals at specific locations including a sparse sensor distribution (S1) and a random placement near the building wall and ground (S2) as shown in Fig. 11(a) and Fig. 11(b) respectively. In random sensor placement scenario, six sensors are randomly placed near the building wall at a distance of 4 mm, while 19 sensors are positioned near the ground at a distance of 12 mm.

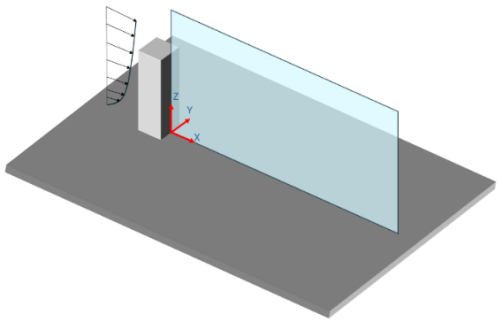
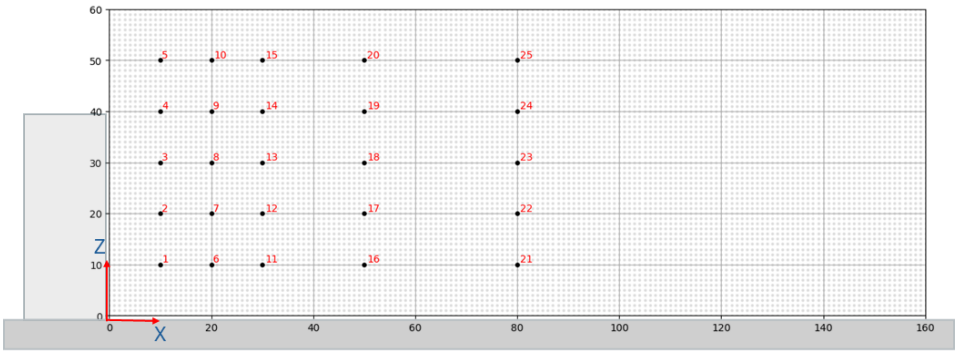
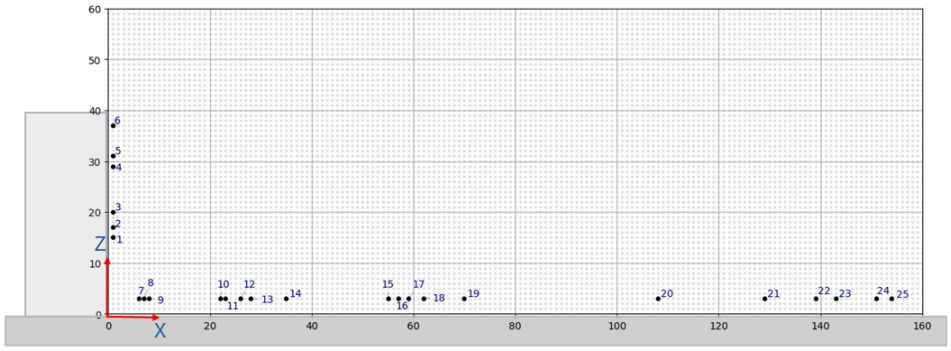


Figure 10: Vertical plane position.



(a)



(b)

Figure 11: Sensor placement strategies include: (a) Sparse domain distribution (S1), and (b) Random near-wall distribution (S2).

4.1 Overview of the Case Study Domain and CFD simulation settings

The case study models an isolated cubical building ($H = 0.16\,m$, $B \approx 0.5\,H$) placed on a flat floor and exposed to a thermally stratified atmospheric boundary layer. The velocity profile gives a roof-level speed of $U_H = 1.37\mathrm{ms}^{-1}$,and temperature drops from 45.3 °C at the ground to 11.3 °C at roof height, yielding a bulk stratification $\Delta\theta \approx 30.4\,°C$. Fig. 12 illustrates the domain and boundary conditions.

In the numerical setup, an Embedded LES (E-LES) method is used. A high-resolution LES zone is placed around the obstacle to accurately capture detailed flow features, while the area farther away is modeled using a coarser mesh with a URANS (realizable k–ε turbulence model) approach to reduce computational cost. The dynamic Smagorinsky SGS model drives the LES,

and synthetic vortex forcing transfers modeled turbulence to resolved scales across the interface. The hexahedral mesh consists of a total of 541,590 cells, and the simulation time step is adopted at $\Delta t = 1 \times 10^{-4} s$. This configuration captures coherent shear-layer eddies while cutting computation cost by roughly a factor of two compared with a full-domain LES. For a more detailed overview of the CFD configuration, refer to the reference [91].

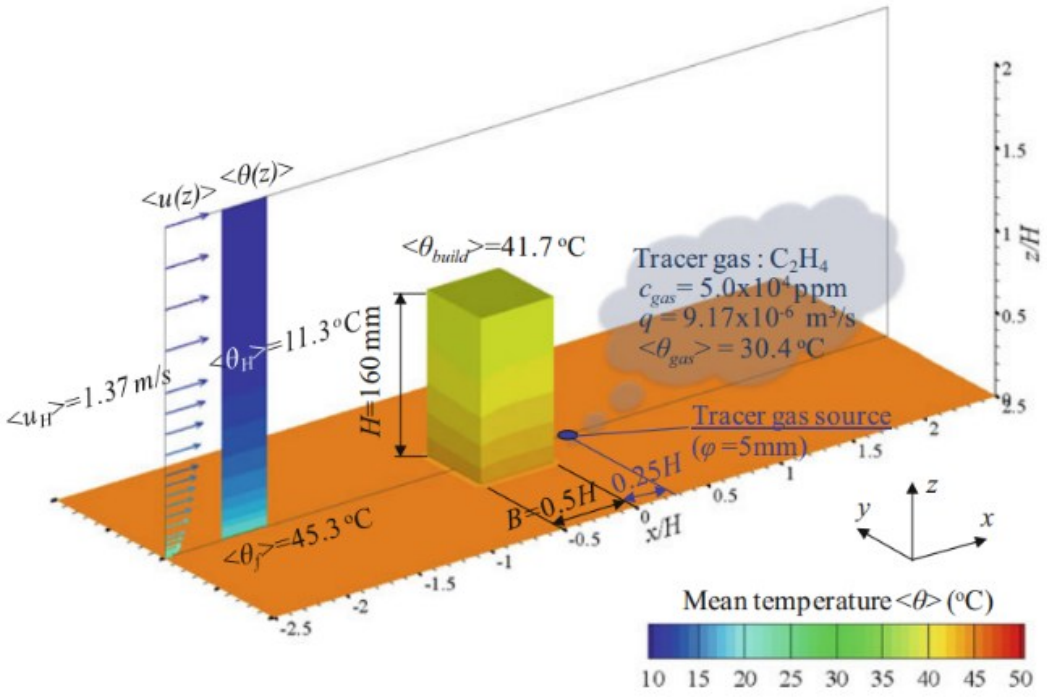


Figure 12: A schematic view of the isolated building model and the boundary conditions, picture from [91], [92].

4.2 Implementation the Method on the Case Study

4.2.1 Noise Augmentation of Sensor Signals During Offline Training

Gaussian (or normal) noise is a stochastic perturbation whose samples n follow the probability-density function [93]:

$$p(n) = \frac{1}{\sqrt{2\pi\sigma^2}} exp\left[\frac{(n-\mu)^2}{2\sigma^2}\right] \tag{43}$$

Where μ and σ denote the mean and standard deviation, respectively. In most signal-processing machine-learning applications the mean is set to zero (μ=0), ensuring an unbiased

perturbation, while σ controls the noise level. Because many naturally occurring disturbances are well approximated by this distribution and the central-limit theorem guarantees its emergence in aggregated errors, Gaussian noise is the canonical choice for stress-testing models under realistic measurement variability [94]. In the present study, each raw virtual sensor $S_{(t)}$ was augmented by an additive Gaussian component:

$$\tilde{S}_{(t)} = S_{(t)} + \varepsilon_{(t)}, \qquad \varepsilon_{(t)} \sim N(0, \sigma^2) \tag{44}$$

With the standard deviation fixed at $\sigma = p\,\mu_s$, where μ_s is the mean absolute amplitude of the corresponding sensor signal and p introduces a controlled noise-to-signal ratio across all 50 input signals, striking a balance between regularisation strength and signal fidelity.

Augmentation was applied exclusively to the inputs of the training partition, and target sequences and all validation/test data remained noise-free. Exposing the model to these perturbed observations compels it to infer the underlying clean dynamics, thereby mitigating over-fitting and markedly improving robustness to measurement noise in real-world deployments. Fig. 13 illustrates an example of additive Gaussian noise with a noise level of $\sigma = 0.05\,\mu_{s6}$ applied to the streamwise velocity signals recorded by sensor 6 in the S2 sensor configuration scenario.

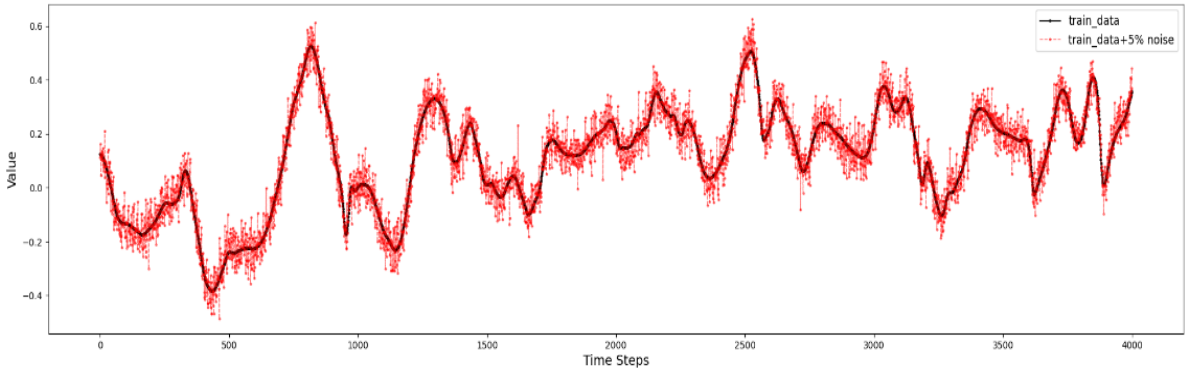


Figure 13: Comparing the original and noisy signals from sensor number 6 in S2 sensor settings during the offline training phase.

4.2.2 Architecture of the BILSTM Network and Hyperparameters Selection

The BILSTM model described is designed for time series prediction task, leveraging the temporal relationships within the sensor signals. It consists of one BILSTM layer with 128 LSTM units in each direction, which process sequences in both forward and backward directions to capture past and future dependencies effectively. The model handles streamwise and crossflow velocity signals for each 25 sensors (50 features), ensuring the representation of complex patterns.

For each sensors configuration (S1 and S2) a separate BILSTM network is trained. To prepare the input data, first 4,000 sequential samples are allocated for training, while the remaining 2,084 samples are reserved for validation/test sets (online prediction phase). A look-back window sequence length of 5 is used, enabling the model to analyze 5 previous time steps for making predictions, with a prediction sequence length of 1.

During training, white noise was injected into the model's input data. Experimental results showed that a noise level of $\sigma = 0.05\,\mu_s$ provided the best balance between bias and variance. Higher noise levels (greater than 10%) hindered the learning convergence, while lower levels (less than 2%) had minimal regularization effect.

The training process employs a dropout rate of 0.025, preventing overfitting by randomly deactivating some neurons during training. Moreover, Batch Normalization [95] is incorporated into the network architecture to enhance training speed and stability. By normalizing the input of each layer to have a mean of 0 and a standard deviation of 1, it mitigates the internal covariate shift (the change in the distribution of layer activations caused by weight updates during training) thereby facilitating more efficient and reliable learning process.

The model is trained over 300 epochs with a batch size of 256, allowing for efficient learning with large datasets. A learning rate decay strategy is implemented to adaptively regulate the learning rate over training epochs, thereby facilitating more stable convergence as the model progresses through training iterations. This process begins with an initial learning rate of 0.01, which is then multiplied by a decay factor of 0.99 at the end of each training iteration. This gradual reduction helps to stabilize the training process by allowing larger updates in the early stages and finer adjustments as the model converges, ultimately improving overall performance and reducing the risk of overshooting the minimum.

For optimization, the model utilizes the Adam optimizer [96] and the dense layers of the model use the *tanh* activation function, ensuring outputs remain within the preprocessing bounded range. This architecture provides a robust framework for sequence modeling, making it suitable for applications requiring accurate temporal predictions. Fig. 14 illustrates the network architecture, and Table. 1 summarizes the selected hyperparameters for the BILSTM network.

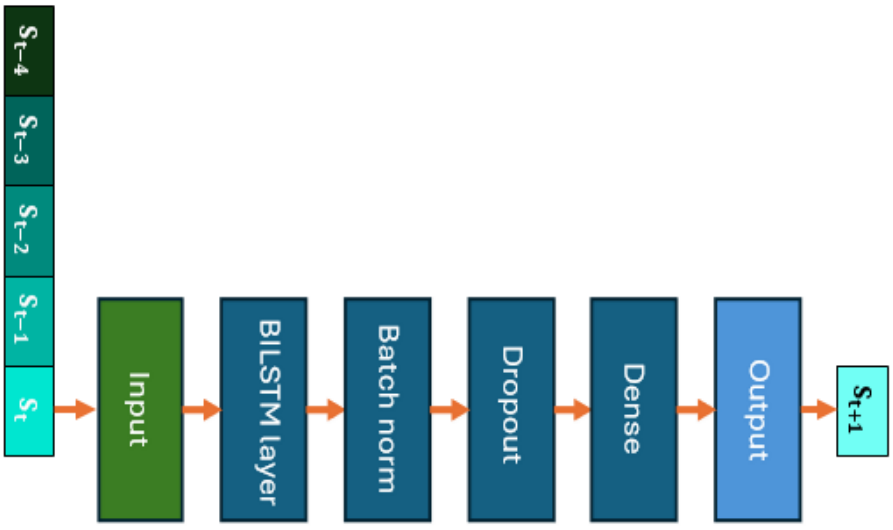


Figure 14: Schematic representation of the proposed BI-LSTM model architecture.

Table 1: Hyperparameters selected for the proposed BI-LSTM network.

Parameter	Value
Number of BI-LSTM layers	1
Number of input and output features	50
Number of LSTM units	(2*64)128
Look-back window sequence length	5
Prediction sequence length	1
Dense layer activation function	tanh
Loss function	MSE
Dropout rate	0.025
Number of epochs	300
Batch size	256
Initial learning rate (lr)	0.01
Learning rate schedule per epoch	lr*0.99
Optimizer	Adam
Gaussian noise level (additive)	$\sigma = 0.05\mu_s$

4.2.3 Preliminary Assessment of the SP-DMD and POD analysis on Case Study

In this section, the SP-DMD algorithm is applied to the high-order wind velocity snapshots. To facilitate the implementation of SP-DMD technique, the full-state velocity data must first be reshaped. As mentioned, the dataset for the model development phase consists of 4,000 snapshots, each containing 9,600 spatial points with streamwise and crossflow velocity components. These snapshots are initially structured as (4000, 60, 160, 2) and need to be flattened into a shape of (4000, 19200) for processing (see Fig. 15).

Before performing the SP-DMD analysis, it is necessary to determine the appropriate number of modes for the exact DMD. To do this, POD is first applied, followed by the evaluation of two reconstruction error metrics. As previously explained, POD is a technique that extracts the most energetic and coherent structures (modes) from high-dimensional data (such as velocity fields in CFD) by maximizing variance (equivalently, kinetic energy). The objective is to identify a set of

orthogonal spatial modes that efficiently represent the data while capturing as much of its kinetic energy as possible with the fewest number of modes

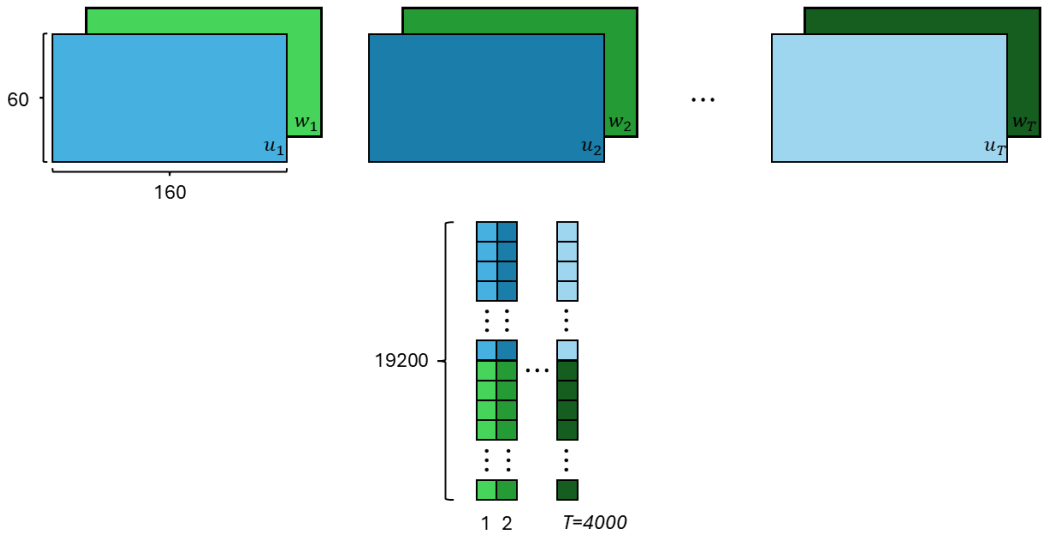


Figure 15: Data flattening process for SP-DMD algorithm.

POD is implemented via SVD of the snapshot matrix X_1 (see Eq. 25). This decomposition yields singular values σ_i, each associated with a POD mode. These singular values are the square root of the energy (variance) captured by each POD mode, therefore, ${\sigma_i}^2$ represents the kinetic energy associated with POD mode i.

The total kinetic energy (TKE) contained in the velocity dataset is thus given by the sum of the squared singular values:

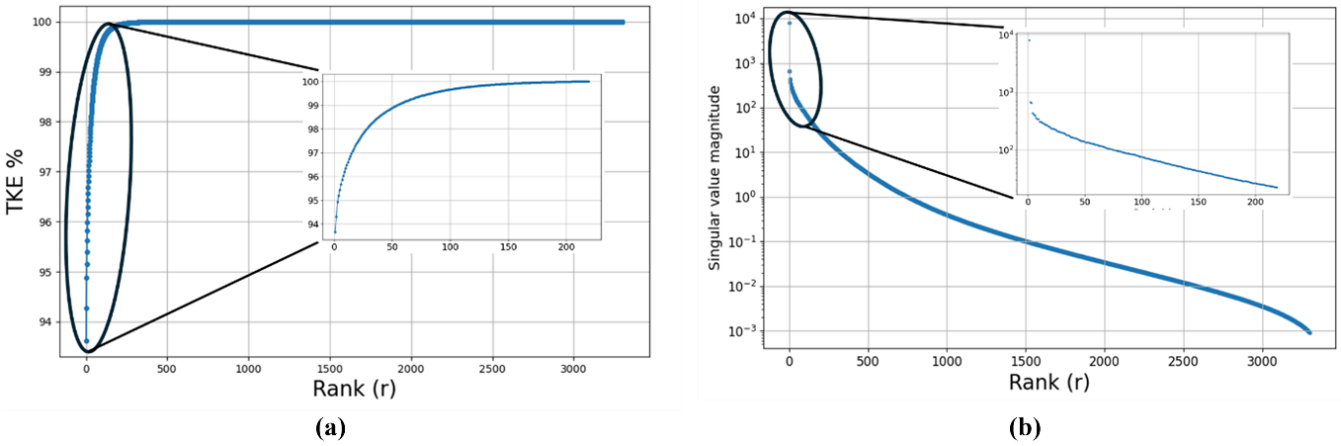
$$\mathrm{TKE} = \sum_{i=1}^{r} \sigma_i^2 \tag{45}$$

This provides a physically meaningful interpretation that the energy content of each POD mode is explicitly given by the square of its corresponding singular value. Truncating the decomposition after k number of POD modes allows for a low-rank approximation that preserves

most of the total energy, with minimal loss of information. The percentage of total energy captured by the first k modes is given by:

$$\mathrm{TKE(k)\%} = \frac{\sum_{i=1}^{k} \sigma_i^2}{\mathrm{TKE}} \times 100 \tag{46}$$

Total of 3,300 velocity field snapshots are used in this analysis. Selecting the first $k = 50$ number of POD modes captures 98.83% of the TKE, while using the first 219 modes captures 99.95%. Fig. 16(a) and Fig. 16(b) illustrates the percentage of TKE retained as a function of the number of POD modes selected, and the decay of the singular values, respectively.



(a) (b)

Figure 16: TKE% retained for the different number of selected POD modes (a), Singular value decay of the velocity data (b).

An experiment is conducted to evaluate the performance of the exact DMD under varying truncation ranks. Two error metrics are employed for this purpose, namely the root mean squared error (RMSE) over all channels, grids, and time steps from Eq. 48 and the performance loss (π-Loss) from Eq. 47 [7]. The results indicate that selecting the first 219 modes yields a π-Loss of 0.9% and an RMSE of 0.074. These findings are illustrated in Fig. 17.

$$\pi - \mathrm{Loss}\ \% = 100 \frac{\left\| \mathrm{X}_1 - \widehat{\mathrm{X}}_1 \right\|_{\mathrm{F}}}{\left\| \mathrm{X}_1 \right\|_{\mathrm{F}}} \tag{47}$$

$$\mathrm{RMSE} = \sqrt{\frac{1}{T \cdot C \cdot \mathrm{G}} \sum_{t=1}^{T} \sum_{c=1}^{C} \sum_{g=1}^{G} \left(\mathrm{X}_{1_{t,c,g}} - \widehat{\mathrm{X}}_{1_{t,c,g}} \right)^2} \tag{48}$$

Here, T represents the total number of snapshots in offline phase (in this case, 4,000), C denotes the number of velocity channels (2 in this study), and G is the total number of spatial grid points (9,600). X_1 refers to the ground truth data obtained from the CFD simulation case study, while $\widehat{\mathrm{X}}_1$ corresponds to the reconstructed data.

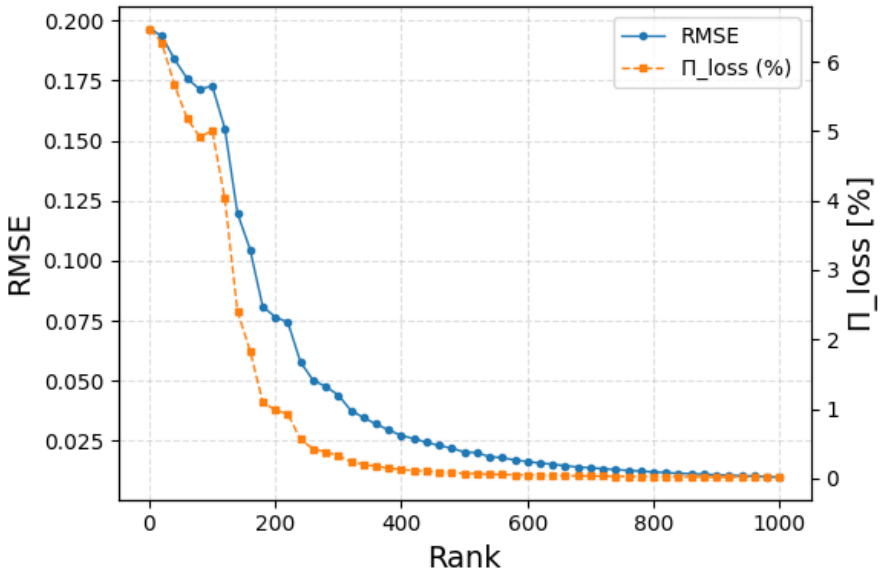


Figure 17: Reconstruction error of exact DMD with different truncation ranks.

Each DMD mode has an associated amplitude that reflects its contribution to the initial condition. Modes with higher amplitudes have a more substantial impact on the system's behavior. By calculating these amplitudes, the modes can be ranked accordingly. Fig. 18 shows the first 6 dominant modes sorted in descending order based on the absolute values of the corresponding mode amplitudes $|b_i|$, for streamwise and crossflow velocity fields.

By applying the SP-DMD algorithm, the goal is to identify the optimal trade-off between the number of selected modes out of a total available mode (219 in this study) and the reconstruction

error. To evaluate the performance loss at varying levels of sparsity, the π-Loss metric proposed in the original SP-DMD study is employed.

An experiment was conducted by evaluating 300 logarithmically spaced values of the penalization parameter γ, ranging from 0.05 to 100,000, to investigate its impact on the π-Loss and the number of non-zero mode amplitudes (NZ). Fig. 19 shows the results for this experiment, as can be expected, increasing the penalization parameter γ leads to greater π-Loss in the reconstructed velocity fields. Moreover, as the γ parameter increases, the number of NZ (corresponding to the number of active modes) decreases. It is shown that the combined velocity fields can be effectively reconstructed using 120 DMD modes, with $\gamma = 708.7$ resulting in a π-Loss of 1.61%.

$$\pi - \text{Loss } \% = 100\sqrt{\frac{\mathrm{J(b)}}{\mathrm{J(0)}}} = 100\frac{\|\mathrm{X_1} - \Phi \mathrm{D_b V_{and}}\|_\mathrm{F}}{\|\mathrm{X_1}\|_\mathrm{F}} = 100\frac{\left\|\mathrm{X_1} - \widehat{\mathrm{X}}_1(b)\right\|_\mathrm{F}}{\|\mathrm{X_1}\|_\mathrm{F}} \tag{49}$$

4.2.4 Creating the Library of Filtered Dynamic Coefficients by Selecting the Effective Features from Ψ_{sparse}

This thesis presents a novel method for extracting unique and meaningful values from the sparse dynamic coefficient set (Ψ_{sp}) which is essential for full-state reconstruction. The main objective of this method is to reduce the number of output features required by the MTL network, thereby decreasing prediction error and enhancing the model's robustness.

The matrix Ψ_{sp} has a dimension of 4000×219, where 4000 corresponds to the number of time steps and 219 represents the dynamic coefficients associated with each mode. Notably, many entries across the same column indices are zeros, indicating sparse temporal activity for certain modes. To reduce dimensionality and focus on the active components, the process begins by

removing columns consisting entirely of zero-valued entries, reducing the matrix size to 4000×120 as expected. Subsequently, real-valued components with zero imaginary parts (typically corresponding to mean flow dynamics) are excluded. As this in this study, they include only two modes, the final matrix is reduced further to a size of 4000×118.

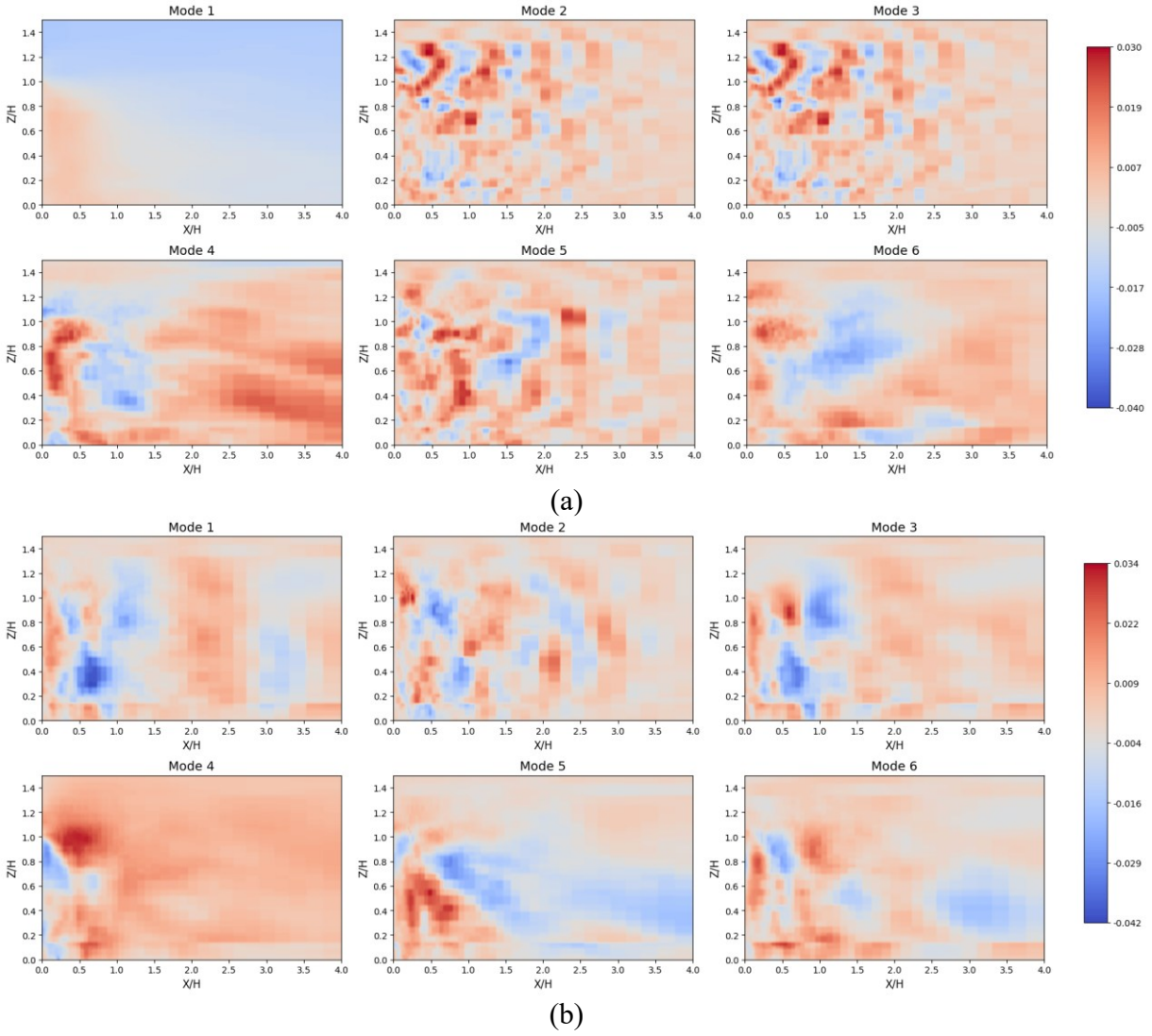
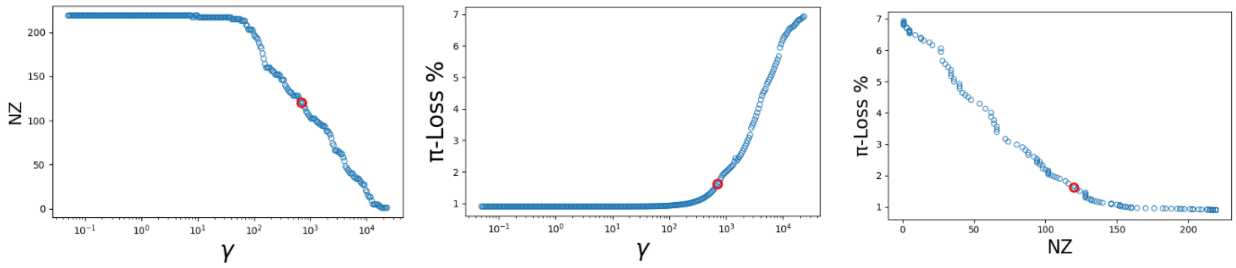


Figure 18: The first six most dominant spatial modes, sorted in descending order of the energy based on the absolute values of the corresponding mode amplitudes $|b_i|$, for (a) streamwise and (b) crossflow components



(a) (b) (c)

Figure 19: (a) Relationship between the sparsity level γ and the number of non-zero mode amplitudes. (b) π-Loss values correspond to different sparsity levels γ. (c) Correlation between π-Loss and the number of non-zero mode amplitudes. The red circle indicates the selected sparsity level used in this work.

Following this, redundant mirrored values across the real axis (due to conjugate symmetry) are discarded. The real and imaginary parts of the remaining complex coefficients are then separated, producing two matrices: one real-valued and one imaginary-valued, each of size 4000×59. Finally, the previously removed isolated real-only components are reintroduced to the real-valued matrix, resulting in a final shape of 4000×61. As a result, the pipeline defines two output tasks for the MTL network:

- **Task 1**: Predict a real-valued matrix formed by combining the real parts of the complex coefficients with the reinserted real-only components, yielding 61 features.
- **Task 2**: Predict an imaginary-valued matrix comprising only the imaginary parts of the complex coefficients, yielding 59 features.

Overall, this preprocessing strategy (1) removes non-informative zero-padding, (2) isolates purely real coefficients, (3) reduces redundancy due to conjugate symmetry, and (4) transforms the prediction task into a numerically stable and well-structured dual-output regression problem.

4.2.5 Implementation of the MTL Model

The MTL model is employed to predict the unique real and imaginary components of the SP-DMD dynamic coefficients through a shared and task-specific architecture. This architecture is

trained using sensor signals collected during the offline model development phase. As previously discussed, the objective of the MTL model is to predict the growth/decay rates (real components with 61 features) and the corresponding frequencies (imaginary components with 59 features) based on the sensor readings.

4.2.5.1 Data Pre-processing for MTL Network

This model will process 50 input features, which include 25 streamwise velocity signals and 25 crossflow velocity signals. The dataset consists of 4000 samples, which are split randomly into 90% for training, 10% for validation/test set.

To enhance generalization and mitigate overfitting, additive relative Gaussian noise is applied to the input signals during training, while noise-free input signals are used for validation. Specifically, Gaussian noise is injected into the training data (sensor measurements) to simulate variability and improve the model's robustness to signal disturbances.

4.2.5.2 MTL Model Architecture and Hyper Parameters

The model employs four shared dense layers to extract common representations beneficial for both tasks, complemented by task-specific dense layers. The real component prediction task includes five hidden layers, while the imaginary component prediction task also utilizes five hidden layers. Dropout with a rate of 0.002 is applied across all layers to prevent overfitting.

The activation function used in the hidden layers is Leaky ReLU, which enables learning from both positive and negative input values, while the output layers use a Linear activation for accurate regression. The model is optimized using the ADAM optimizer [96], with a learning rate scheduling strategy similar to that employed during the training of the BILSTM network. The MTL network is trained using a batch size of 256 over 800 epochs, ensuring robust learning.

To minimize prediction errors, the model employs the Huber Loss function [97], which is particularly effective at handling outliers. Commonly used in regression tasks, Huber Loss combines the advantages of both Mean Squared Error (MSE) and Mean Absolute Error (MAE). It is less sensitive to outliers than MSE while maintaining a smooth and differentiable structure, transitioning between MSE and MAE based on a tunable hyperparameter δ. For each task t (where $t \in 1,2$), the loss is computed using the true values y_t, and the predicted values $\hat{y}_t$ as defined by the Huber Loss formula.

$$\mathcal{L}oss_t = \frac{1}{n}\sum_{j=1}^{n} Huber\left(y_{t,j}, \hat{y}_{t,j}\right) \tag{50}$$

$$Huber_\delta(y, \hat{y}) = \begin{cases} \frac{1}{2}(y - \hat{y})^2 & for\ |y - \hat{y}| \le \delta \\ \delta \cdot \left(|y - \hat{y}| - \frac{1}{2}\delta\right) & for\ |y - \hat{y}| > \delta \end{cases} \tag{51}$$

The total loss is calculated as the weighted sum of the individual losses across all tasks. Equal loss weights, $\alpha_1 = \alpha_2 = 1$, are assigned to both tasks to ensure balanced importance between them:

$$\mathcal{L}oss_{total} = \sum_{t=1}^{2} \alpha_t \cdot \mathcal{L}oss_t = \mathcal{L}oss_1 + \mathcal{L}oss_2 \tag{52}$$

With a total of 636,000 trainable parameters, the model efficiently combines shared and task-specific learning to enhance predictive accuracy. This design allows the model to simultaneously predict 61 real components and 59 imaginary components of SP-DMD dynamic coefficients using

the sensor signals. The specific details of the model's structure and layer organization can be visualized in Fig. 20. The selected model hyperparameters are summarized in Table 2.

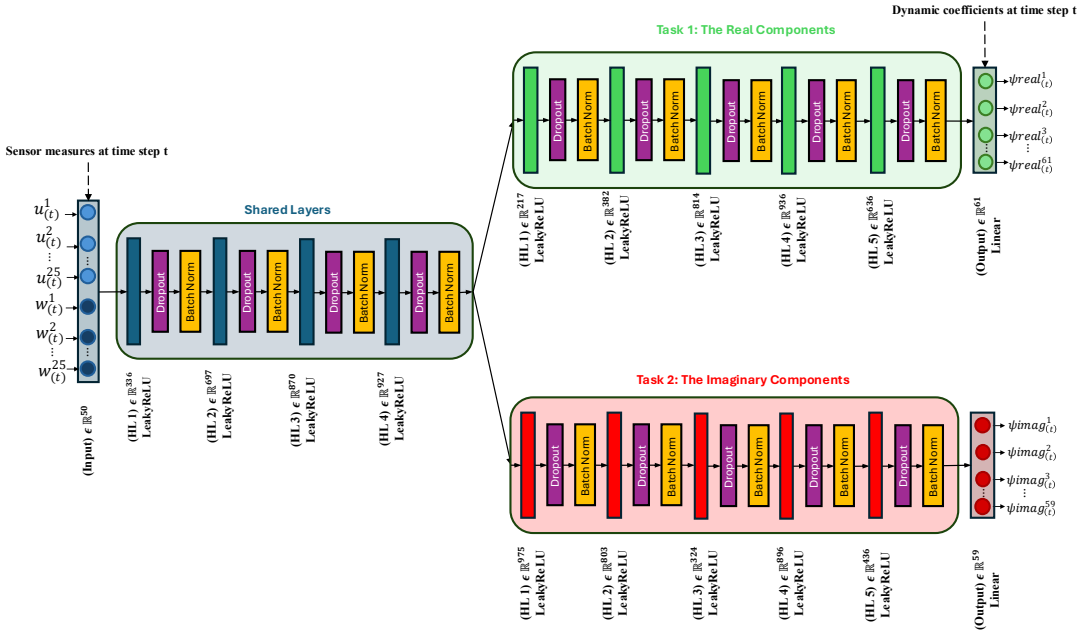


Figure 20: Schematic representation of the proposed MTL model architecture.

Table 2: Hyperparameters selected for the proposed MTL network.

Parameter	**Value**
Number Shared layers	4
Number of hidden layers for task 1	5
Number of output features for task 1	61
Number of hidden layers for task 2	5
Number of output features for task 2	59
Dropout	0.02
Epochs	800
Batch size	256
Initial learning rate	0.01
Learning rate schedule per epoch	lr*0.99
Optimizer	Adam
Final dense layer activation function	Linear
Loss function	Huber ($\delta = 1$)
Gaussian noise level (additive)	$\sigma = 0.05\mu_s$

Chapter 5: Results and Discussion

In this section, the performance of each individual building block of the proposed framework is first evaluated separately, followed by an assessment of the overall performance of the integrated system. Section 5.1 investigates the reconstruction error of velocity fields using the selected modes from SP-DMD, along with the corresponding eigenvalues, mode amplitudes, and error distribution across different sparsity levels. Section 5.2 examines the temporal forecasting framework during the online phase for two sensors' configurations (S1 and S2) under varying levels of noise. Section 5.3 evaluates the mapping performance of the MTL framework, assuming direct access to sensor signals for reconstructing the corresponding full-state fields. Finally, Section 5.4 assesses the overall performance of the integrated framework and conducts model verification.

5.1 Evaluation of the SP-DMD Method on the Case Study

In this section, the results obtained from applying the SP-DMD to wind velocity field data are evaluated. As discussed in the previous chapter, increasing the sparsity level in the mode amplitude vector b_i leads to a sparser optimal solution. This enforced sparsity, however, comes at the cost of increased reconstruction error, as fewer modes are retained to represent the full dynamics of the system.

Fig. 21 and Fig. 22 illustrate the reconstructed SP-DMD velocity fields at $t = 1s, t = 2.5s$ and $t = 3s$ for varying sparsity levels in both the streamwise and crossflow directions, respectively. As the penalization parameter increases, promoting higher sparsity, reconstructions using 195 and 120 active modes remain largely indistinguishable from each other and from the reference field. This suggests that a moderate reduction in the number of modes does not significantly compromise the reconstruction quality.

However, when the sparsity is pushed further (resulting in only a single active mode) the reconstruction degrades considerably. In this extreme case, the velocity field captures only the mean flow pattern, failing to reproduce the high-resolution dynamics in both the streamwise and crossflow components. This highlights the inherent trade-off between model compactness and reconstruction fidelity; that is, while aggressive sparsity enhances interpretability and reduces computational complexity, it simultaneously limits the model's ability to accurately capture complex and high-frequency flow features.

To assess the spatial distribution of SP-DMD reconstruction error under varying levels of sparsity during the offline phase, the MSE is computed and averaged over 4000 snapshot samples. The results are presented separately for the streamwise Fig. 23(a) and crossflow Fig. 23(b) velocity components. Each visualization displays the spatial variation of reconstruction error for selected number of none-zero mode amplitudes: 195, 120, 100, 48, 21, and 1. As the number of retained modes decreases (as sparsity increases) the reconstruction error becomes increasingly significant, particularly in regions characterized by complex flow dynamics or high turbulence intensity. In the extreme case with only a single active mode, the reconstruction predominantly fails to represent the dynamic fluctuations of the flow, resulting in substantial localized errors. Conversely, reconstructions utilizing 100 or more modes demonstrate low MSE values and closely approximate the reference flow fields, emphasizing the trade-off between sparsity and reconstruction fidelity.

These visualizations underscore the impact of sparsity on the spatial accuracy of SP-DMD reconstructions and highlight the importance of selecting an appropriate number of active modes for reliable velocity field recovery in both flow directions.

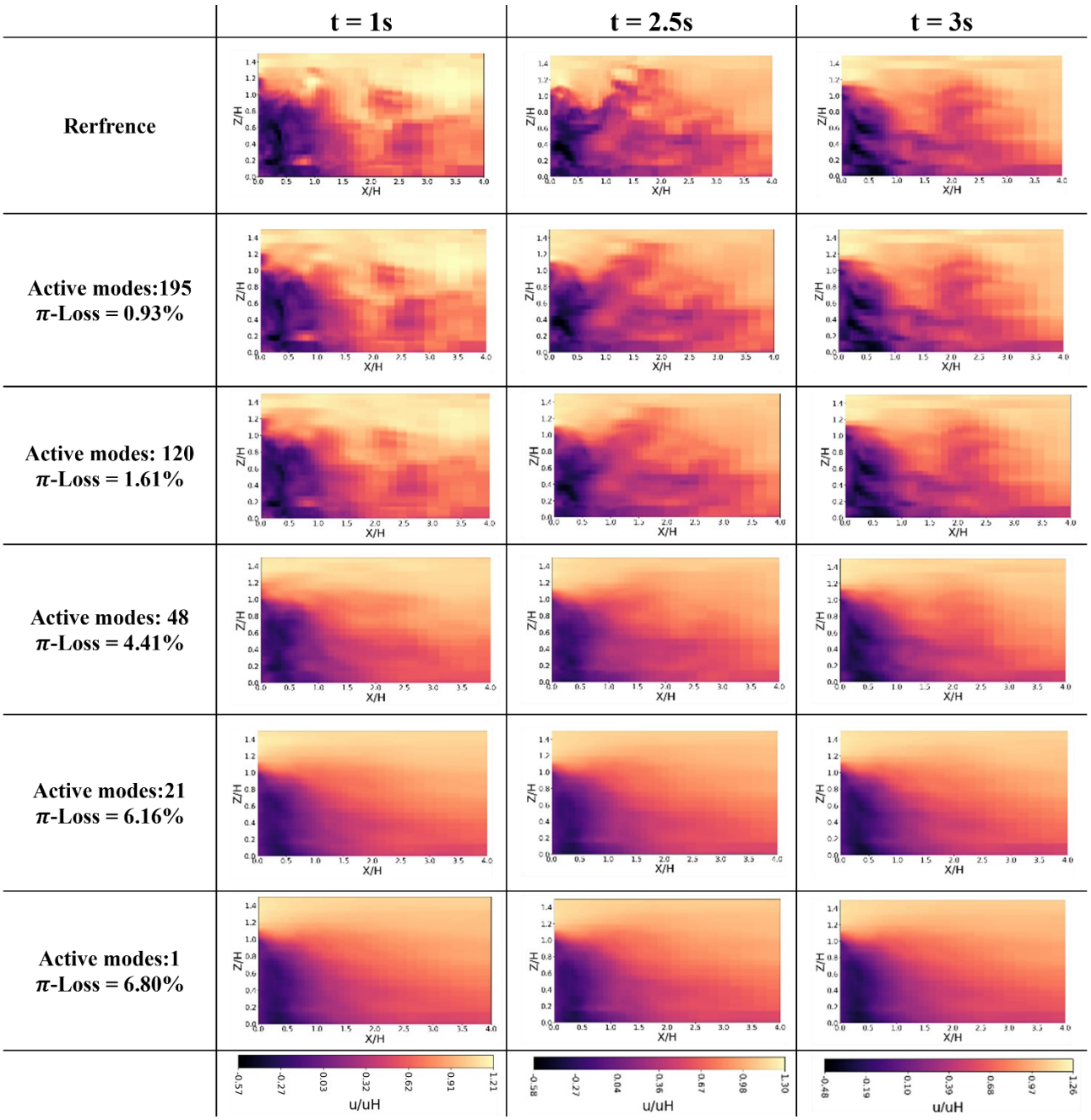


Figure 21: Evaluation of the streamwise velocity field reconstruction using SP-DMD by inducing different levels of sparsity at t=1s, t=2.5s and t=3s during the offline phase.

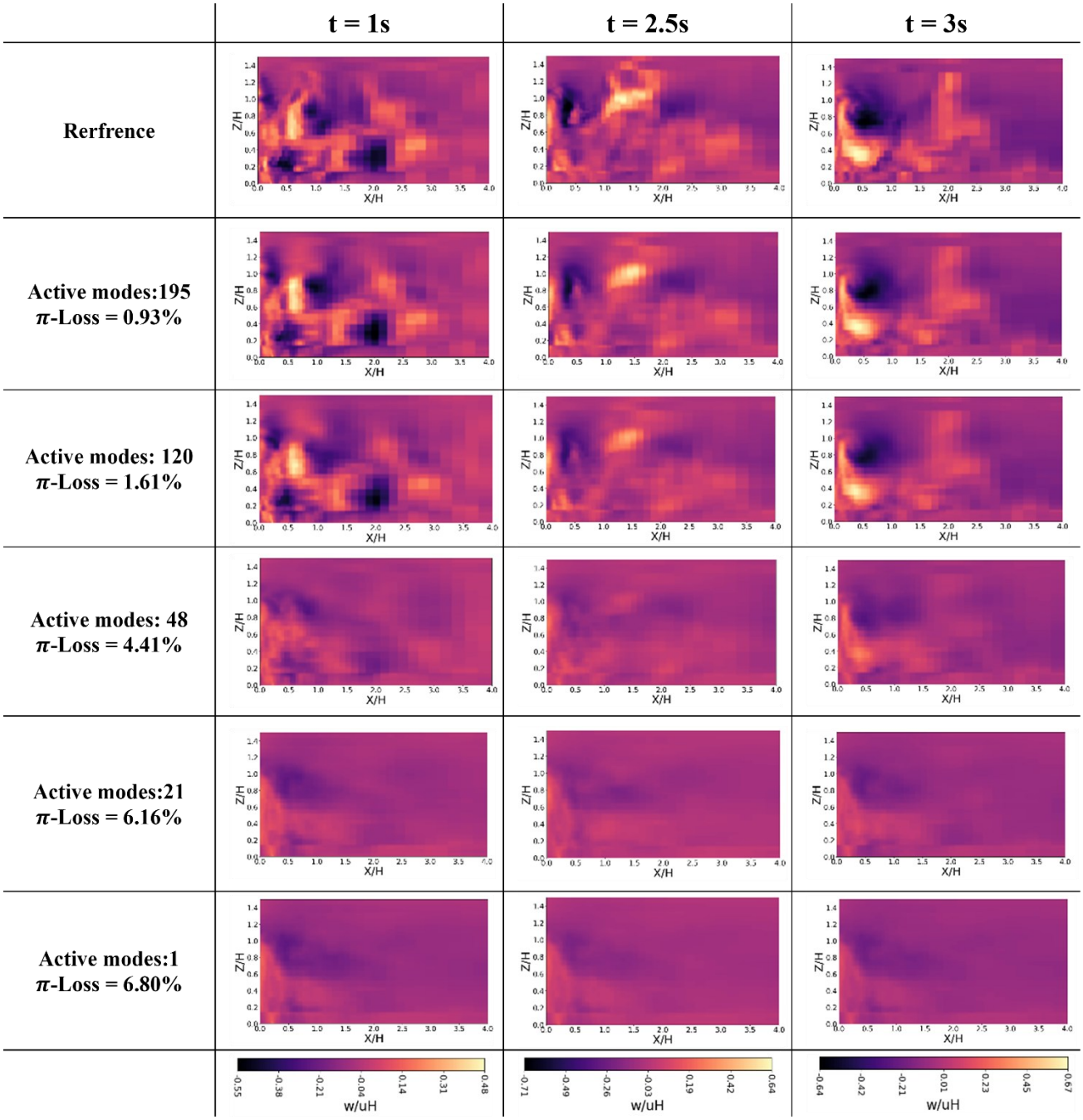


Figure 22: Evaluation of the crossflow wind velocity fields reconstruction using SP-DMD by inducing different levels of sparsity at t=1s, t=2.5s and t=3s during the offline phase.

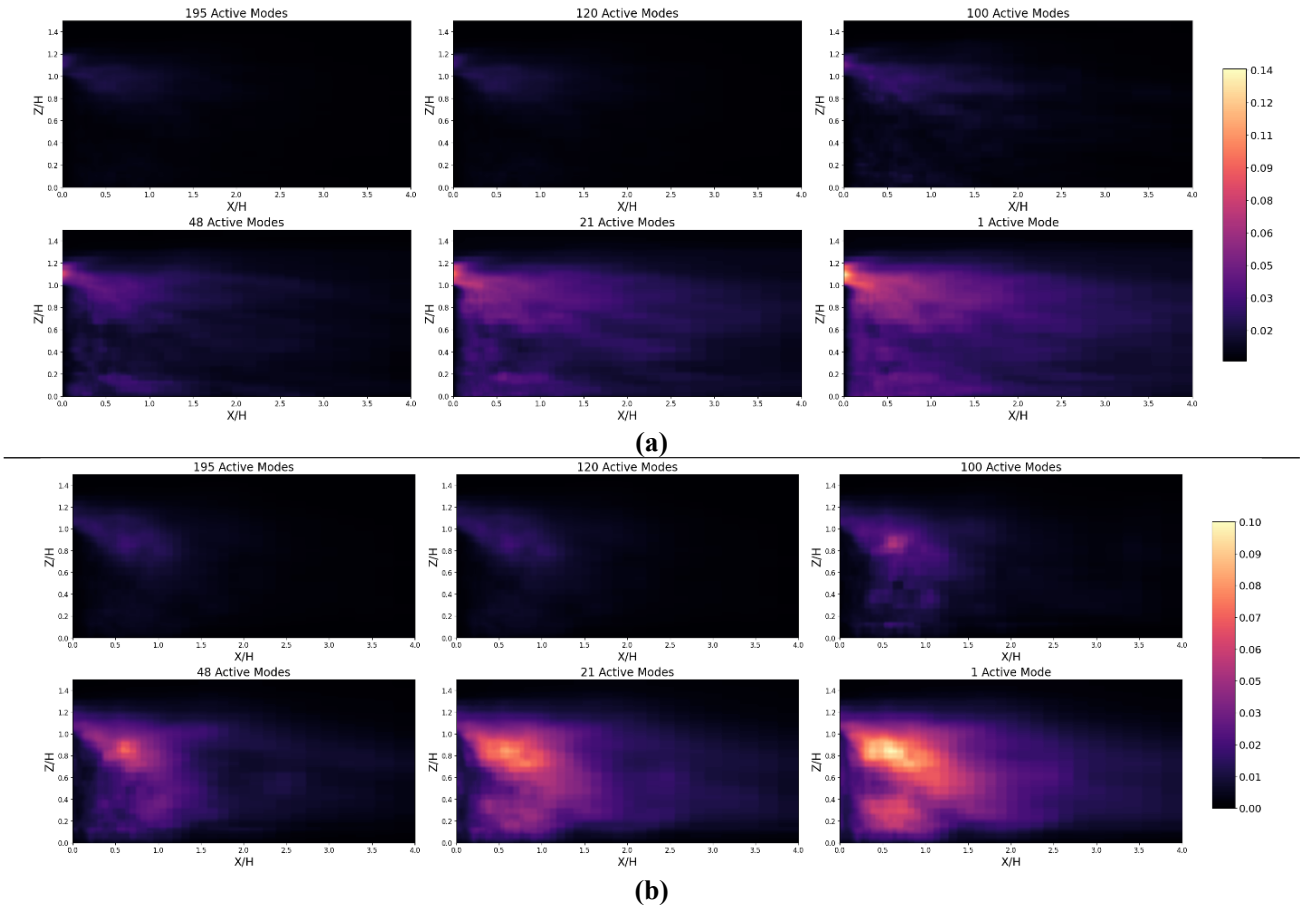


Figure 23: Comparison of MSE distribution in velocity field reconstruction across 4000 offline training phase snapshots at varying sparsity levels (different numbers of active modes), evaluated for: (a) streamwise (u/uH) and (b) crossflow (w/uH) components.

Fig. 24(a) compares the complete discrete-time DMD eigenvalues (blue circles) with the subsets selected by the SP-DMD (red crosses) for progressively smaller selected active modes. as mentioned, in the complex plane, the real part of each eigenvalue represents exponential growth or decay, while the imaginary part corresponds to the oscillation frequency. Eigenvalues near the unit circle signify neutral or weakly damped dynamics (typically associated with coherent structures that dominate long-term flow behavior). In contrast, eigenvalues located deep within the unit circle indicate rapidly decaying modes, often attributed to noise or transient phenomena.

In all cases, SP-DMD prioritizes the retention of low-frequency, weakly damped modes and consistently eliminates highly damped interior modes. As the sparsity level decreases, the selected modes converge to a minimal set of one real mode and a single complex-conjugate pair. This minimal configuration is sufficient to capture the dominant dynamics of the wind velocity around the building with negligible loss in reconstruction accuracy.

Fig. 24(b) illustrates the relationship between the absolute value of mode amplitude $|b_i|$ and its corresponding frequency $Imag(\lambda_i)$. The exact DMD results are shown as blue circles, while the modes selected by the SP-DMD are indicated by red crosses for various sparsity levels. The SP-DMD preferentially retains low-frequency modes associated with higher mode amplitudes (higher energy), meaning it effectively filtering out high-frequency, low-energy modes that are prevalent in the full DMD spectrum. This filtering behavior is further demonstrated, where the level of sparsity progressively narrows the selection to a single dominant low-frequency mode (mean flow).

This underscore two important insights. First, the most energetic flow structures are concentrated in the low-frequency range. Second, relying solely on the exact DMD amplitude spectrum is insufficient for identifying a compact and dynamically meaningful modal basis. This explain that imposing a sparsity constraint, as done in SP-DMD, is essential for extracting the dominant coherent structures effectively.

The absolute values of the mode amplitudes $|b_i|$, plotted with respect to the mode index in the diagonal matrix $diag(b_i)$ for various levels of sparsity, are shown in Fig. 25. This analysis reveals that the mode corresponding to the mean flow (characterized by high energy and low frequency) is located at index 177 in the $diag(b_i)$ matrix. Moreover, as the sparsity level increases, this dominant mode is the only one retained, indicating that aggressive sparsity prioritizes the

preservation of low-frequency, energy-dominant structures while discarding higher-frequency, lower-energy components.

5.2 Online Reconstruction of Full Wind Field States from Sparse Sensor Signals using direct sensor readings

In this section, wind flow fields are reconstructed directly from sensor readings at each corresponding time step. This implies that only two components of the framework (MTL and SP-DMD) are utilized. The model sequentially processes the real-time sensor readings at each time step to predict the corresponding full-state velocity fields.

Fig. 26 and Fig. 27 present the MSE of the reconstructed flow fields during the online phase for sensor placements S1 and S2, respectively, compared to the reference data. Fig. 26(a) and Fig. 27(a) show the model's overall performance when the MTL network is trained on noise-free signals for the S1 and S2 sensor settings, respectively. For both sensor placement scenario, the overall reconstruction error increases with higher noise levels. Conversely, when the MTL model is trained on noisy data, it becomes less sensitive to noise, resulting in a more robust and reliable framework (Fig. 26(b) and Fig. 27(b)).

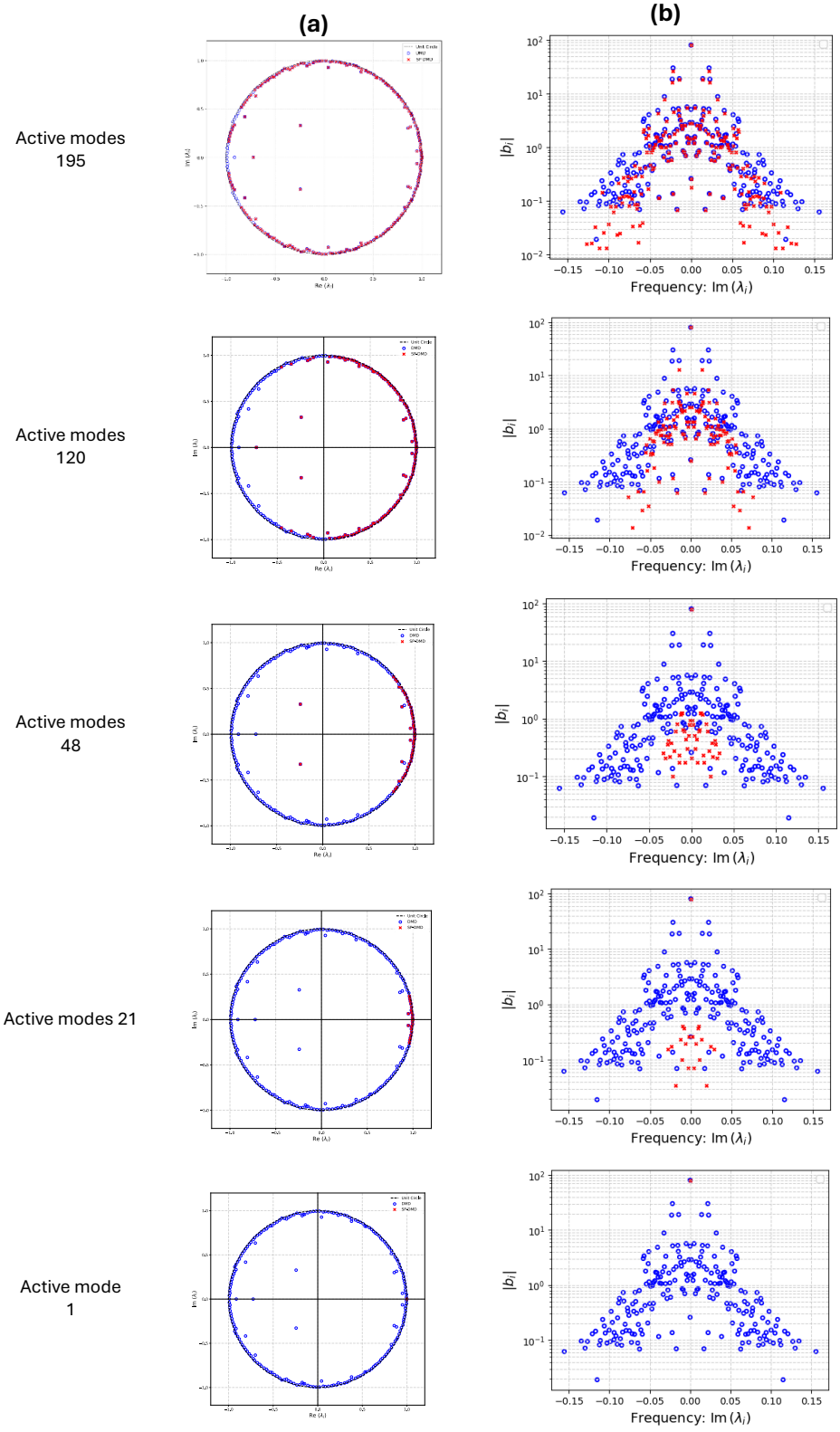


Figure 24: (a) Eigenvalues from the exact DMD algorithm (blue circles) and the non-zero eigenvalues selected by the SP-DMD algorithm (red crosses) at different levels of sparsity. (b) Absolute values of the mode amplitudes b_i and the corresponding frequencies on the different levels of sparsity

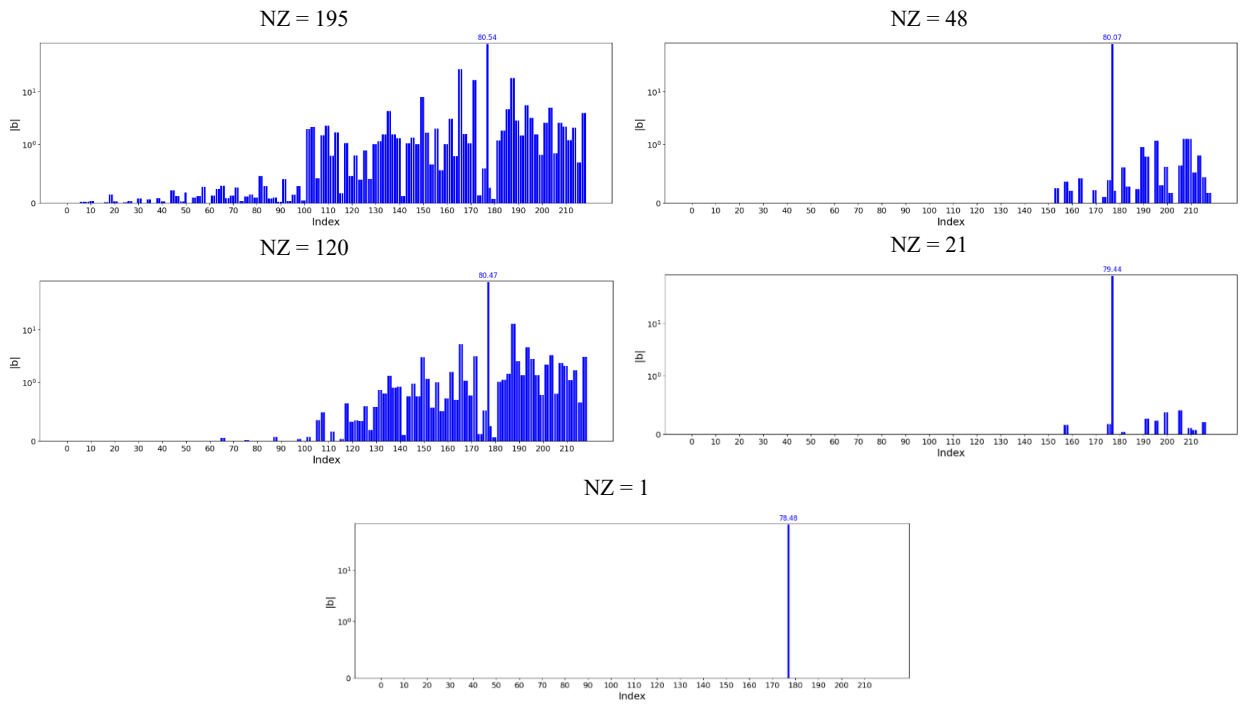


Figure 25: Absolute values of the mode amplitudes b_i on the different levels of sparsity.

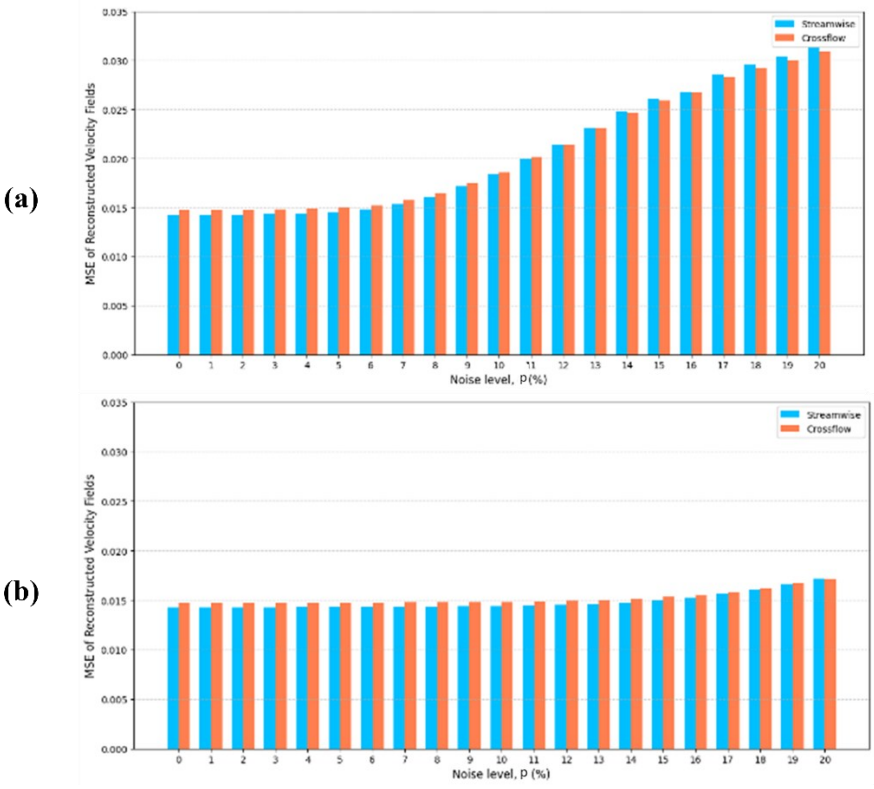


Figure 26: Online reconstruction error of wind velocity fields with respect to the reference data under varying noise levels for S1 sensor placement scenario: (a) model trained on noise-free signals, and (b) model trained on noisy signals.

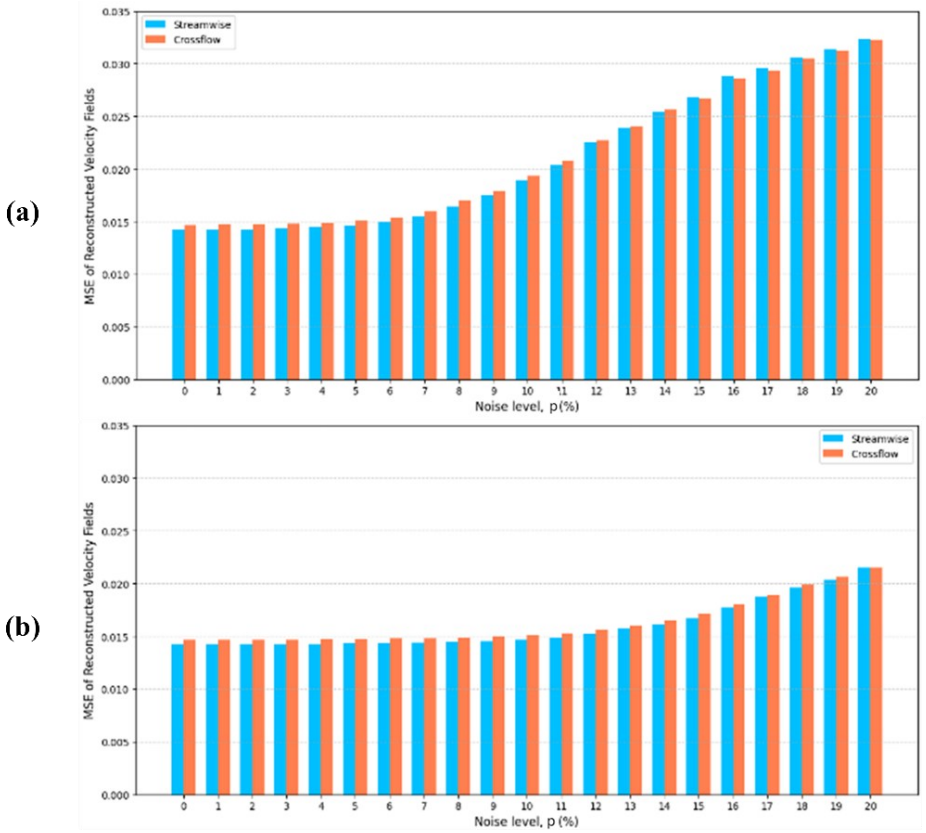


Figure 27: Online reconstruction error of wind velocity fields with respect to the reference data under varying noise levels in S2 sensor placement scenario: (a) model trained on noise-free signals, and (b) model trained on noisy signals.

5.3 Performance Evaluation of the BILSTM for Predicting Sensor Signals

Fig. 28(a) and Fig. 28(b) present the predicted sensor signals over a duration of 2084 timesteps (2.084 seconds) from the online forecasting (test) phase for the S1 and S2 sensor configurations, respectively. The results are shown for the 1st, 6th, 11th, 16th, and 21st sensors, including both streamwise and crossflow velocity components. In each plot, the CFD reference data is represented by a solid black line, while the predicted values are shown as a dotted orange line. These visualizations facilitate a direct comparison between predicted and reference signals, allowing for an intuitive assessment of the network's tracking performance across a range of temporal

dynamics. Across all sensors, the predicted trajectories closely follow the reference data, effectively preserving both amplitude and phase characteristics.

Under a 5 % relative additive Gaussian-noise perturbation, the BILSTM forecasts for both S1 and S2 sensor layouts remain tightly aligned with the CFD reference throughout the 2,084 time-step (2.084 s) online phase. Across the 1st, 6th, 11th, 16th and 21st sensors, the network preserves peak amplitudes and phase relationships, showing only minor attenuation during the most energetic crossflow bursts. This close overlap between predicted and truth signals confirms the model's robustness to measurement noise and its ability to generalise to distinct spatial configurations while sustaining high-fidelity reconstruction. Fig. 29 illustrates these findings.

To evaluate the performance of the time series BILSTM model, a heat map of absolute deviation error was generated on the test run across various noise levels (Fig. 30). The horizontal axis represents the 2,084 timesteps during the prediction phase, while the vertical axis corresponds to 25 sensors signals from S2 settings, used for both streamwise and crossflow velocity predictions.

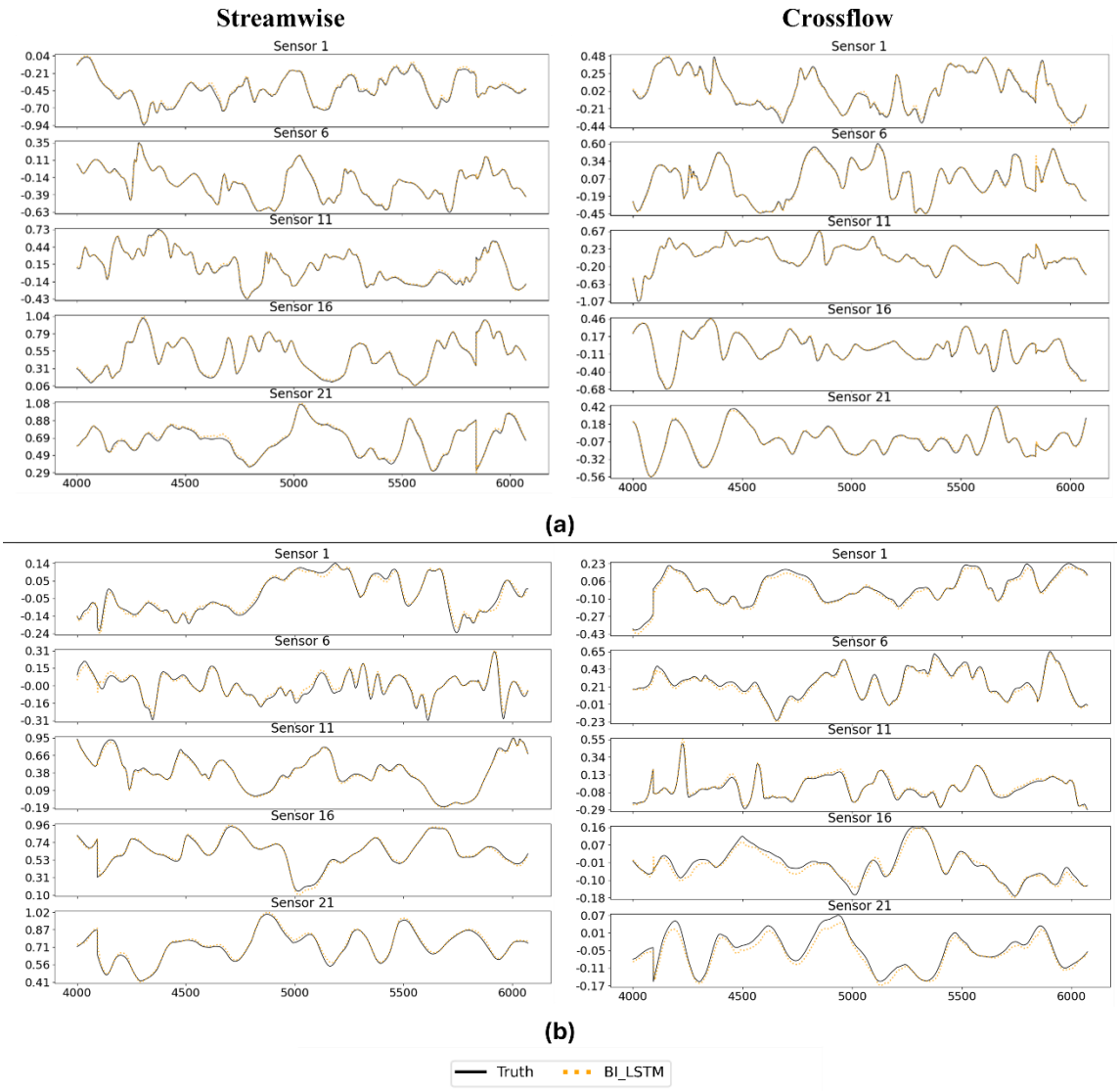


Figure 28: Comparison of BILSTM model predictions with original sensor signals (1st, 6th, 11th, 16th, and 21st) for Streamwise velocity $U(ms^{-1})$ and Crossflow velocity $W(ms^{-1})$ using (a) S1 scenario and (b) S2 scenario at the online forecasting phase.

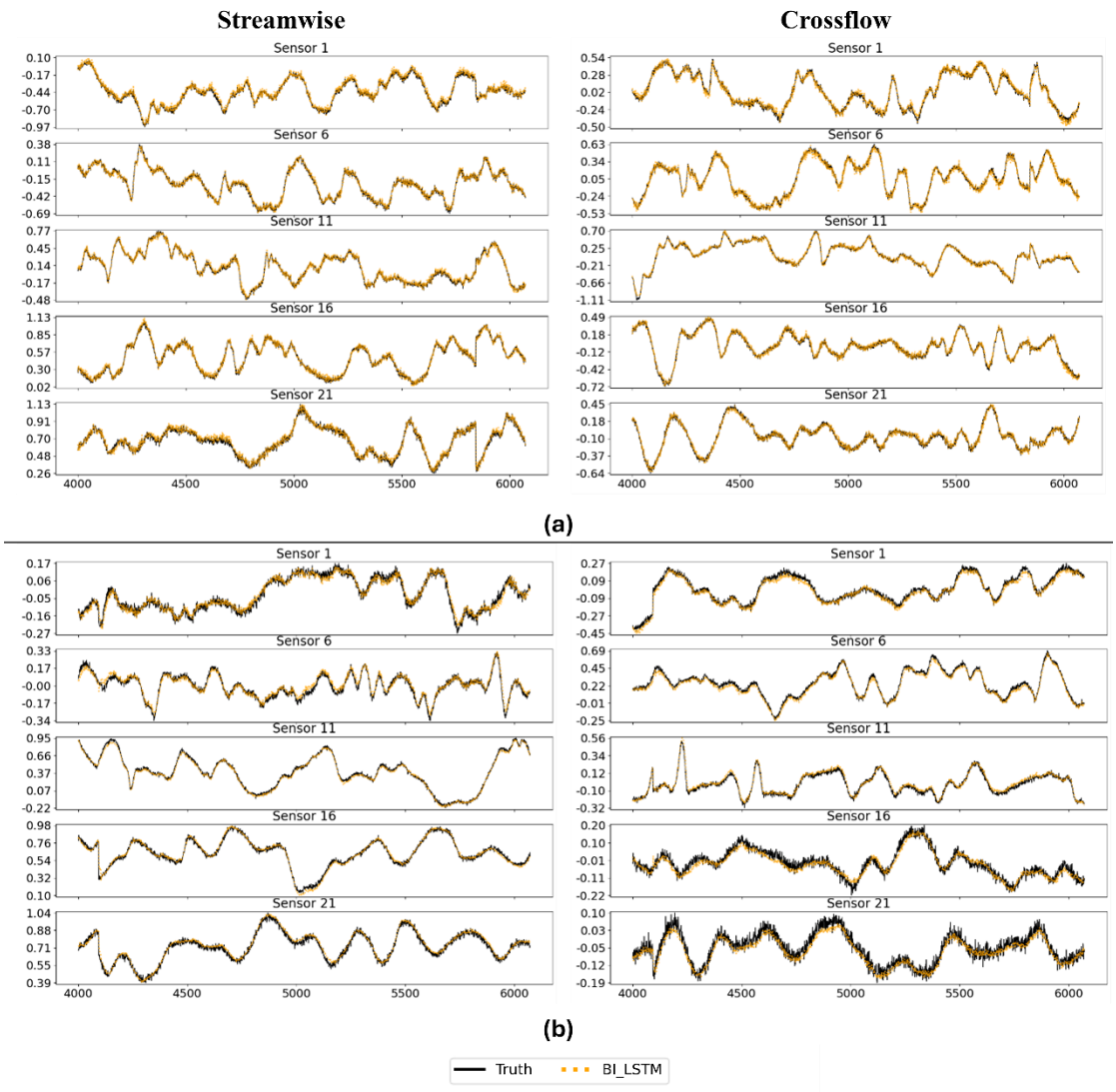


Figure 29: Comparison of BILSTM model predictions for sensors 1st, 6th, 11th, 16th, and 21st under a 5% relative noise level, showing Streamwise velocity $U(ms^{-1})$ and Crossflow velocity $W(ms^{-1})$ during the online forecasting phase using the (a) S1 scenario and (b) S2 scenario.

For most of the predictions, the absolute deviation error remains below 0.1. However, a notable error occurs around time step ≈ 110, where the error exceeds 0.25 for streamwise velocity at sensors 14 to 20. This anomaly is attributed to a sharp drop in streamwise velocity observed at those sensors during that specific time step. The localized error spike indicates that the model encounters difficulties in capturing abrupt transients, although its overall performance remains stable and consistent across the temporal and spatial domains.

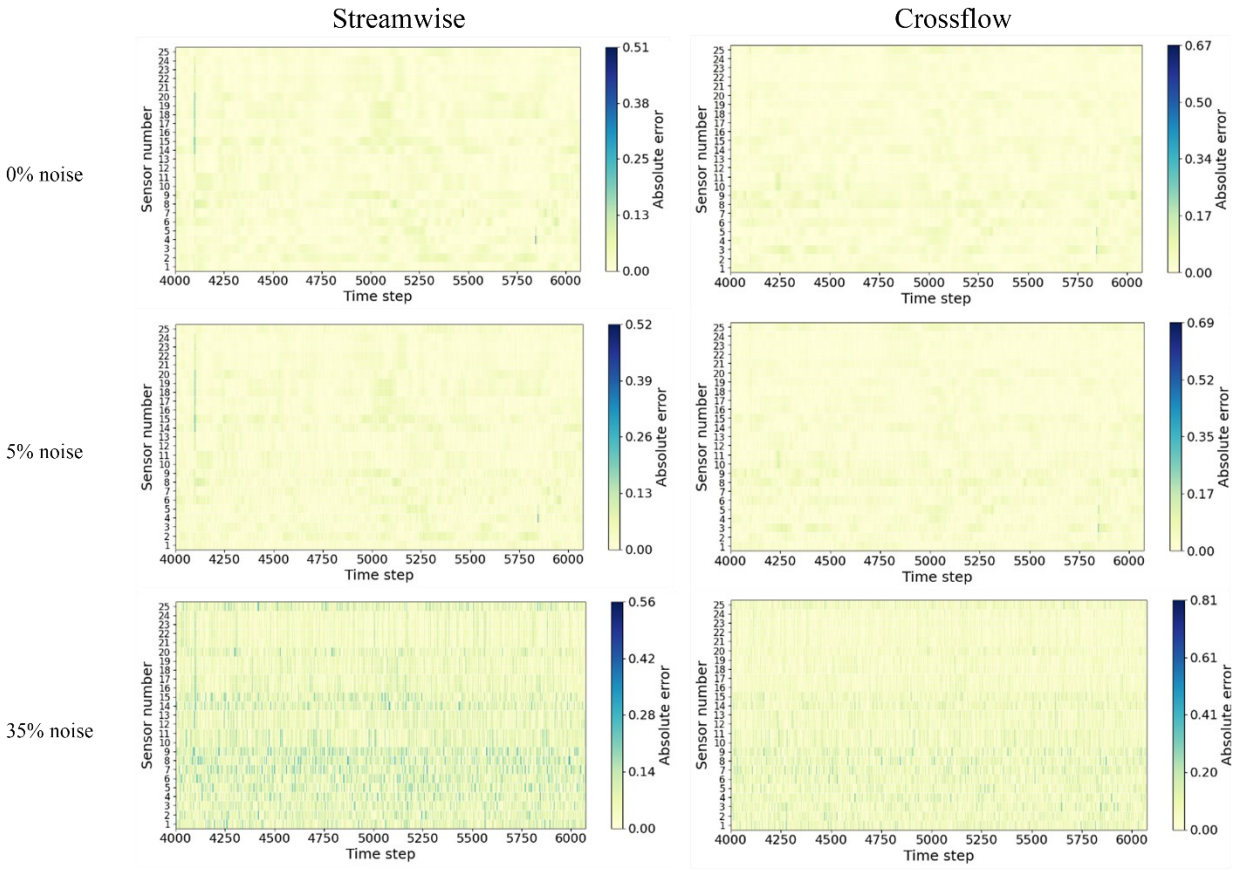


Figure 30: Absolute-error heat map of the BILSTM test run across varying noise levels.

5.4 Reconstruction of Full-State Velocity Fields Using the Integrated Model and Corresponding Verification

In this section, the performance of the integrated framework is assessed under the S2 sensor configuration, in which only randomly distributed near-wall sensors are employed. This arrangement is preferred for real-world applications and has already demonstrated acceptable accuracy for real-time monitoring.

Fig. 31 presents a comparison between the reconstructed turbulent velocity fields and the reference case at $t = 0.5s, 1s$, and 1.5s under three noise conditions: no noise, $\sigma = 0.25\mu_s$ and $\sigma = 0.35\mu_s$. The corresponding spatial distributions of MSE are also shown, highlighting the impact of varying noise levels on reconstruction accuracy across the total predicted snapshots. As

can be expected, the largest reconstruction errors occur in the building's near-wake region due to high turbulence area. Moreover, increasing the relative noise level in the input signals leads to a higher concentration of reconstruction errors in the same regions.

Time-averaged streamwise and crossflow velocity profiles of the reconstructed fields are presented in Fig. 32 for four locations $x/H = 0.25$ (downstream of the building), $x/H = 0.5$, $x/H = 2.5$, and $x/H = 3.5$ (far wake). Moreover, Fig. 33 shows the instantaneous profiles sampled at $t = 1.5\ s$ at the same x/H positions.

The reconstructed fields show strong agreement with the CFD reference at the far wake locations, while minor discrepancies are observed near the building, where turbulence intensity is highest specificity for $z/H = [0.5 \sim 1.25]$. Across varying noise levels, the reconstructed fields maintain consistent qualitative behavior compared to the CFD data, demonstrating the method's robustness in capturing transient flow structures in the near-wake region. It is further observed that increasing noise levels have a more pronounced effect on instantaneous profiles than on time-averaged results, indicating that the time-averaged are less sensitive to input noise.

Fig. 34 presents the time histories of streamwise and crossflow velocity fluctuations $(u'/U_H, w'/U_H)$ obtained from both the CFD reference solution and the proposed predictive framework under varying levels of input noise during the online prediction phase. The streamwise and crossflow velocity fluctuations can be computed using Eq. (43) and (44), respectively:

$$u' = u - \bar{u} \tag{53}$$

$$w' = w - \bar{w} \tag{54}$$

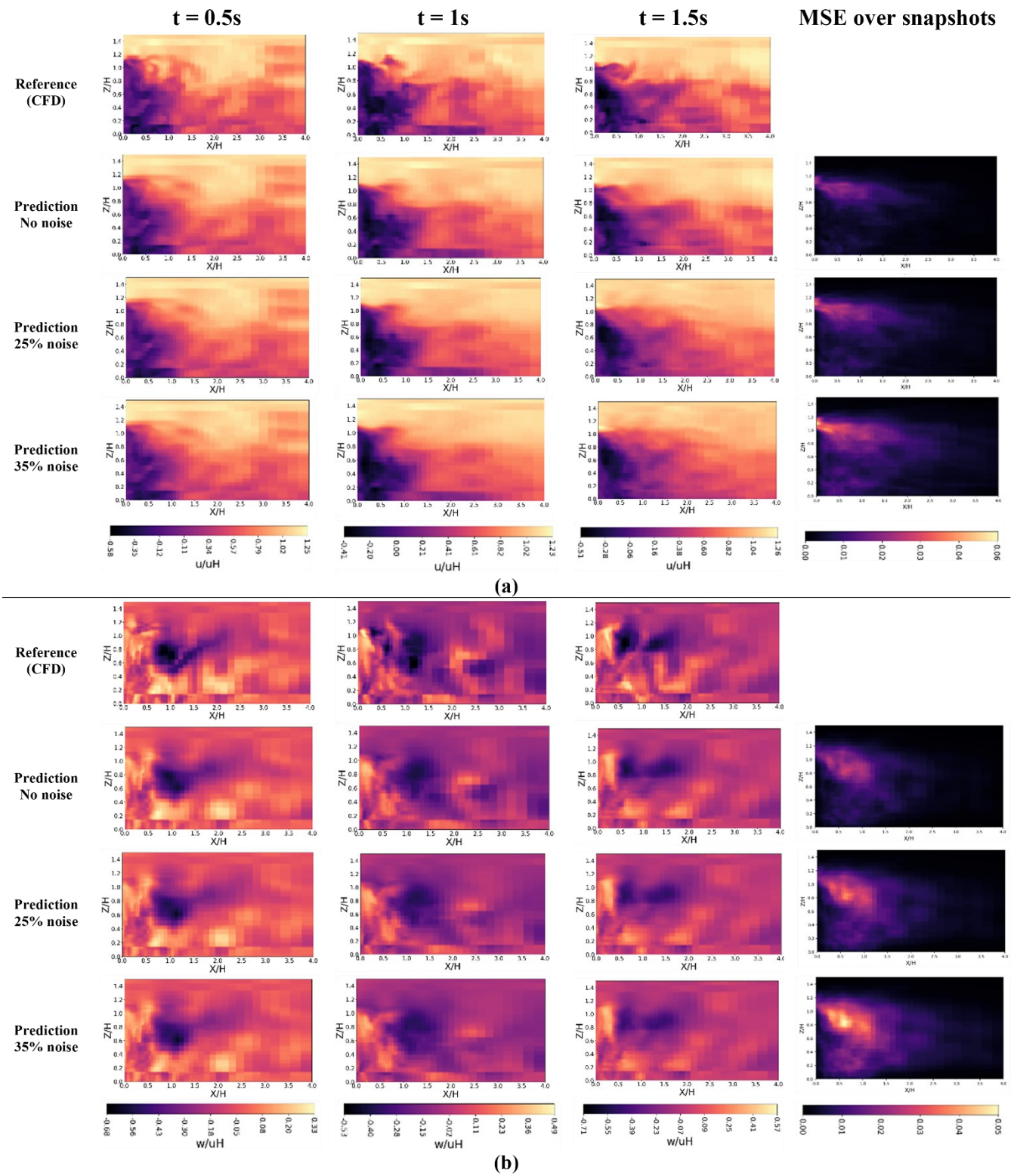


Figure 31: Reconstruction of velocity fields for (a) streamwise and (b) crossflow components based on different noise levels at S2 sensor locations during the online prediction phase.

where $\bar{u}$ and $\bar{w}$ denote the time-averaged streamwise and crossflow velocity components, respectively. The comparisons are performed at three representative sampling locations for $(x/H, z/H) = (0.25, 1)$, $(2, 0.75)$, and $(2.95, 1.45)$, corresponding to the near-wake, mid-wake, and far-wake regions, respectively

At the near-wake location $(x/H, z/H) = (0.25, 1)$, where turbulent fluctuations are strongest, the reconstructed velocity signals qualitatively follow the CFD reference and successfully capturing dominant patterns. However, the effect of noise becomes more evident in this region. As the noise level increases (particularly at $\sigma = 25\%\mu_s$ and $\sigma = 35\%\mu_s$), the predicted signals become smoother, leading to attenuation of high-frequency components and local peaks. Despite this, the overall dynamic behavior remains preserved, indicating that the model maintains a level of robustness even under strong noise perturbations.

In the mid-wake region $(x/H = 2, z/H = 0.75)$, the predictions show improved agreement with the CFD data across all noise levels. Both u' and w' components are well captured in terms of amplitude and phase. This highlights the model's effectiveness in regions where the turbulent structures begin to stabilize, and coherent flow patterns dominate.

In the far-wake region $(x/H = 3.95, z/H = 1.45)$, the velocity fluctuations are significantly weaker and dominated by low-frequency dynamics. The reconstructed signals demonstrate excellent agreement with the CFD reference, with minimal influence from noise, even at the highest levels tested.

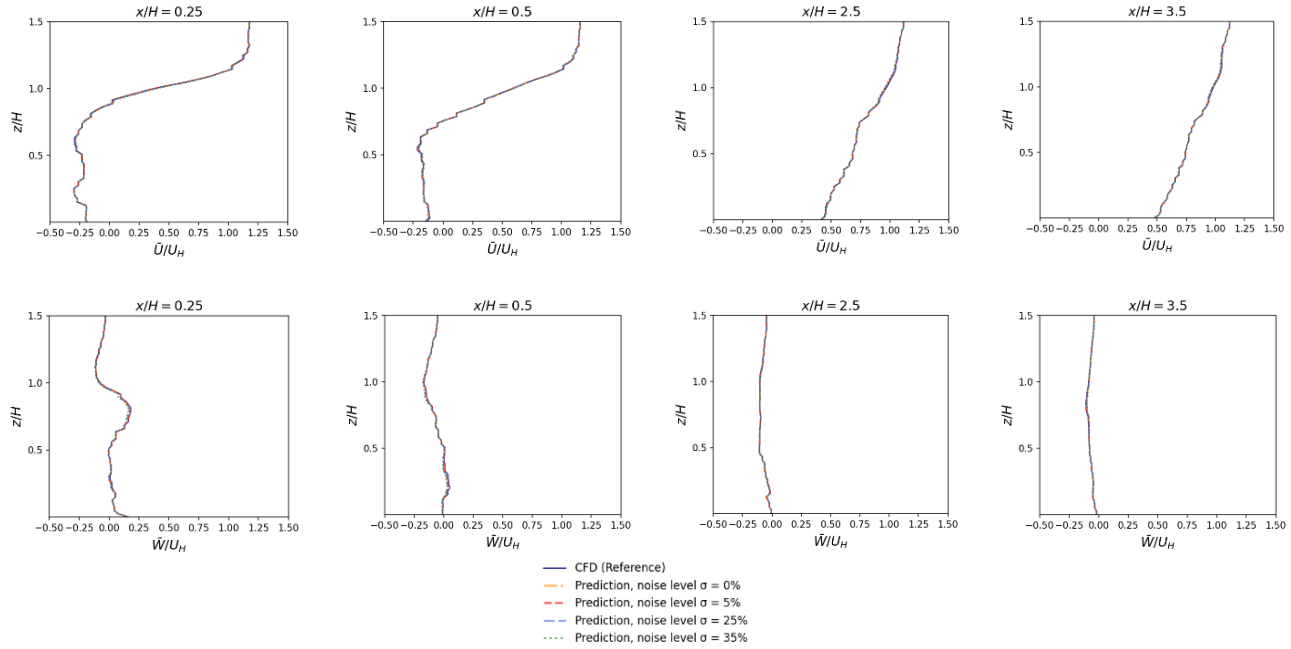


Figure 32: Time-averaged (mean) streamwise and crossflow velocity profiles at positions $x\ /\ H = 0.25$, $x\ /\ H = 0.5$, $x\ /\ H = 2.5$, and $x\ /\ H = 3.5$ under varying level of noises.

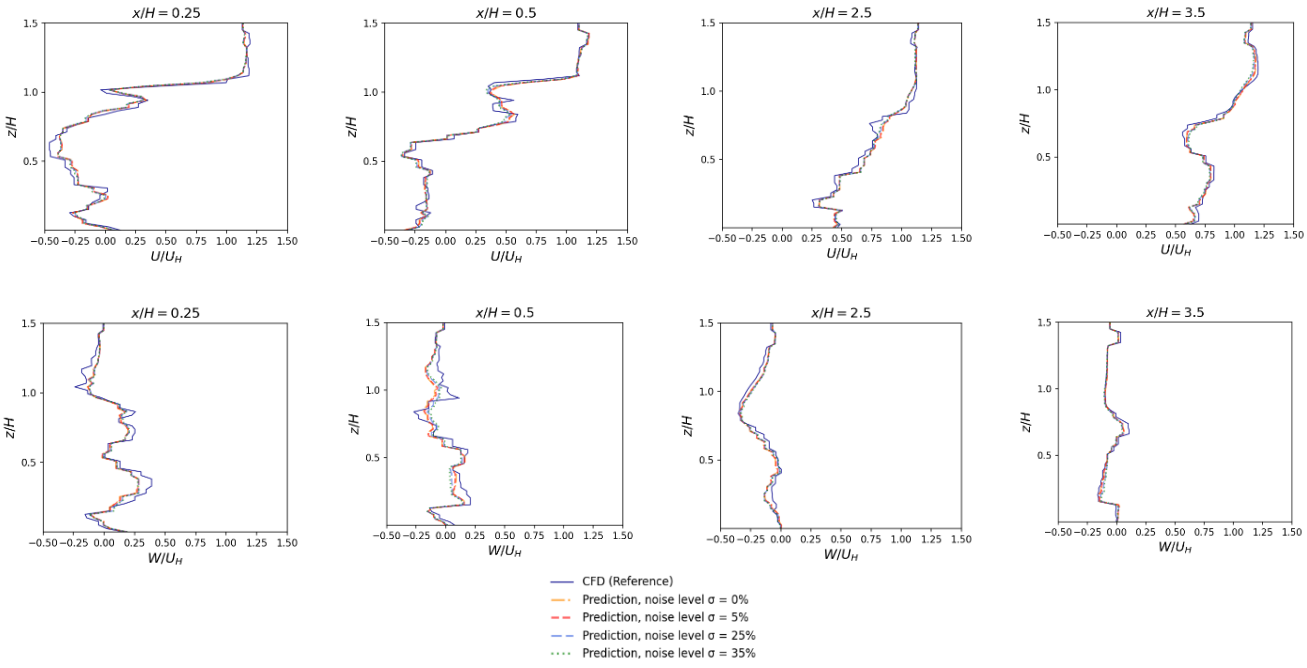


Figure 33: Streamwise and crossflow velocity profiles positions $x/H = 0.25$, $x/H = 0.5$, $x/H = 2.5$, and $x/H = 3.5$ at $t = 1.5s$ under varying level of noises during the online prediction phase.

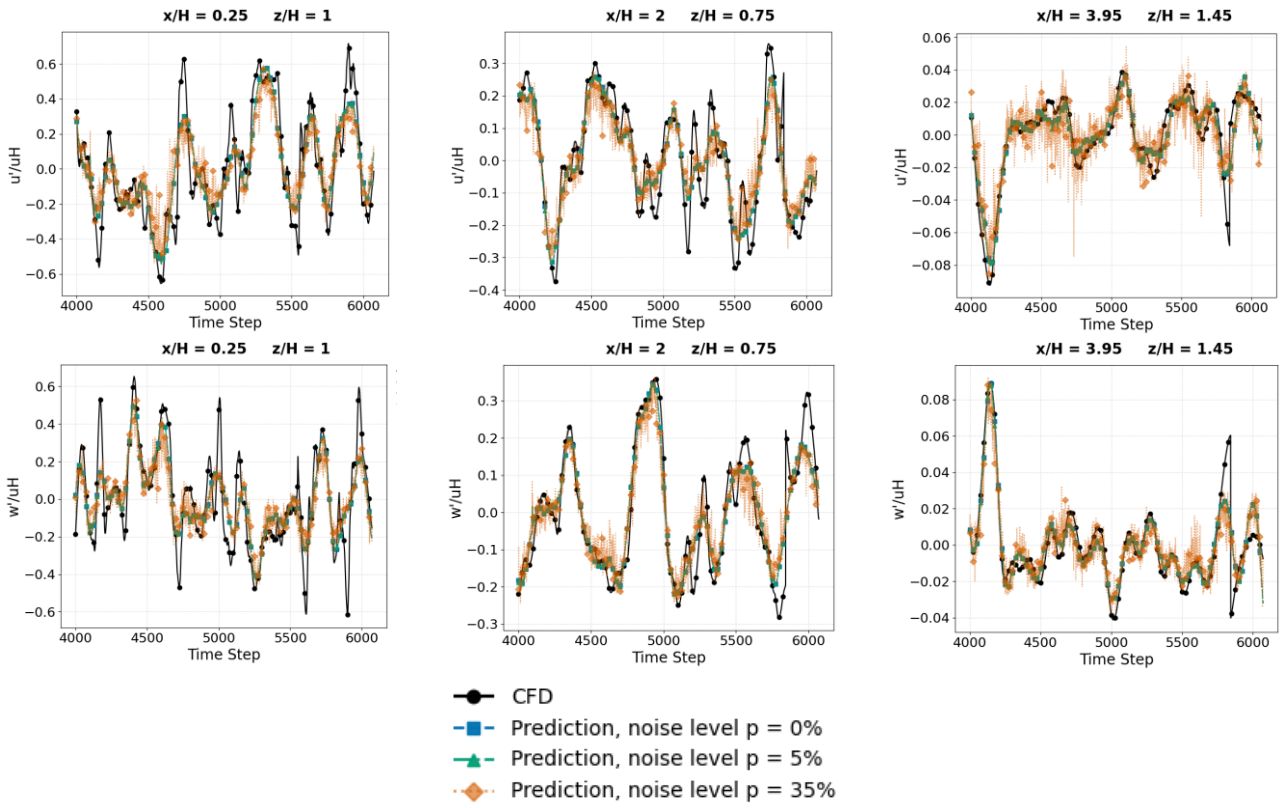


Figure 34: Time histories of streamwise and crossflow velocity fluctuations (u' / U_H and w' / U_H) obtained from the CFD simulation case study and the proposed framework at three sampling locations $(x/H, z/H) = (0.25, 1), (2, 0.75)$ and $(2.95, 1.45)$ under varying level of noises during the online prediction phase.

Chapter 6: Conclusions

6.1 Summary

This thesis presents a hybrid, data-driven framework for real-time reconstruction of high-dimensional velocity fields from sparse sensor measurements. The proposed method operates in two primary stages: an offline phase, where three synergistic components are trained, and an online phase for real-time prediction. The offline phase comprises a BILSTM network for temporal forecasting of sensor signals, a MTL network that maps sparse sensor measurements to the SP-DMD dynamic coefficients, and a SP-DMD algorithm for extracting dominant spatial modes and their associated dynamic coefficients.

The integrated framework is validated using synthetic CFD simulations of wind flow around an isolated high-rise building. Results demonstrate that the method can accurately reconstruct unsteady urban wind fields even under noisy input conditions, showcasing its robustness and generalization capacity. The framework achieves high-fidelity reconstruction across various spatial regions and noise levels, offering particularly strong performance in the mid- and far-wake zones. Although reconstruction accuracy is slightly reduced in the near-wake region (where turbulence intensity and high-frequency fluctuations dominate) the method still captures the essential dynamics.

Importantly, the study finds that time-averaged flow structures remain stable across different noise levels, while instantaneous features tend to be smoothed under increased noise. This trend underscores the model's effectiveness in delivering robust predictions in practical scenarios characterized by sensor uncertainty. Furthermore, it was observed that the majority of

reconstruction error is localized in the near-wake region immediately downstream of the building, correlating with regions of elevated turbulence and velocity fluctuations.

6.2 Contributions

This thesis advances the current state of the art in fast and high-resolution reconstruction and prediction of urban wind fields through the development and validation of an integrated data-driven framework. A key aspect of the work lies in its validation strategy which the complete modeling pipeline is tested on high-fidelity CFD data, thereby allowing for precise evaluation of methodological performance in isolation from real-world measurement noise. The primary contributions of this research are as follows:

- The SP-DMD algorithm was applied to full-state velocity snapshots as model order reduction methods to identify a compact set of dominant spatial modes and their corresponding dynamic coefficients. These coefficients, representing continuous-time growth/decay rates and oscillation frequencies, provide an interpretable and low-rank representation of complex flow dynamics.
- A novel encoder-decoder preprocessing pipeline was introduced as an encoding strategy to isolate and extract the distinct growth/decay rates and oscillation frequencies from the dynamic coefficients. This "separable task" encoding significantly reduces the MTL output dimensionality without sacrificing information content, resulting in faster training convergence and improved model scalability.
- A MTL model was developed to map sparse sensor measurements directly to the SP-DMD dynamic coefficients. This feed-forward architecture effectively acts as a bridge between the

limited sensor signals inputs and full-field flow reconstructions by simultaneously predicting both real and imaginary components of the dynamic coefficients.

- A BILSTM network was employed for temporal forecasting of sensor signals, enabling the capture of temporal dependencies within the data and the prediction of future measurements.
- The end-to-end framework, comprising SP-DMD, MTL, and BILSTM components, was rigorously evaluated to assess both real-time monitoring accuracy and forecasting performance. The integrated system demonstrated the capability to recover low-rank approximations of high-dimensional CFD velocity fields with high. These results underscore the framework's effectiveness and suitability for surrogate modeling in real-time applications in urban air flow prediction.
- The framework was trained under various sensor configurations to evaluate its adaptability to different spatial sampling scenarios. Additionally, the introduction of additive Gaussian noise during training improved the model's generalization capability under noisy or perturbed input conditions. The robustness of the framework was further assessed by testing it across multiple sensor noise levels, revealing its sensitivity to natural measurement noise and its ability to maintain reliable reconstruction performance.

Collectively, these contributions provide a scalable and efficient solution for real-time urban wind field estimation, offering a promising pathway for integration into smart city infrastructure, urban planning tools, and autonomous aerial navigation systems.

6.3 Assumptions and Limitations

While the methodology demonstrates promising performance, the following assumptions and limitations should be considered:

Assumptions:

1. The numerical simulations employed in this study are assumed to reliably capture realistic urban wind flow patterns. The case study, based on CFD simulations around an isolated building with controlled boundary and initial conditions, is considered to provide high-fidelity reference data for model development and validation.
2. The spatial sensor configurations used for sparse sampling are assumed to remain fixed and optimal throughout the study. The model assumes that sensor positions are known and that their readings are accurate and synchronized.
3. The framework assumes that the temporal evolution of the flow is quasi-stationary over the analyzed window, such that the dynamic patterns learned from training data can generalize to future timesteps.
4. For generalization during training, additive relative Gaussian noise is introduced to the input sensor signals. It is assumed that this noise approximates real-world disturbances and sensor inaccuracies.

Limitations:

1. The framework has been validated on synthetic CFD data, which lacks real-world measurement noise, boundary variability, and environmental uncertainties. Hence, model performance may degrade in practical deployments without further calibration.
2. Although noise was synthetically introduced during training, the model has not been tested against real sensor uncertainty or missing data.

3. The model is trained using a fixed temporal resolution (i.e., timestep size). Applying it to datasets with different temporal sampling rates may require retraining or adaptation of the network.

4. The current model operates in a feedforward manner without incorporating mechanisms for online correction or error feedback, which could be crucial for long-term deployment or drifting systems.

5. The urban geometry used in simulation scenarios is relatively simplified and does not account for highly irregular or evolving urban topologies. Therefore, the current model may not generalize well to cities with more complex terrain and architectural diversity.

6.4 Future Works

While this thesis has demonstrated a robust framework for high-resolution reconstruction and short-term prediction of urban wind fields using ROM and deep learning architectures, several directions can be explored to enhance the methodology and expand its applicability:

Future research should validate the proposed framework using real sensor data collected from urban environments. Integrating field-deployable anemometers and sensor arrays will help assess model robustness under realistic atmospheric conditions, including turbulence, measurement noise, and environmental variability.

Developing an adaptive or data-driven sensor placement strategy could improve reconstruction accuracy while minimizing the number of sensors. Optimization techniques, such as genetic algorithms, could be integrated to identify sensor configurations that maximize information gain.

To improve long-term generalization, future work could incorporate online learning frameworks that allow the model to adapt dynamically to new or changing flow conditions. Alternatively, transfer learning approaches could enable pre-trained models to be fine-tuned on different urban layouts or weather scenarios with minimal additional training.

Coupling the current framework with additional physical phenomena (such as pollutant dispersion, thermal effects, or pedestrian comfort indices) can expand the applicability of the model in urban planning, smart cities, and environmental monitoring contexts.

Implementing methods for uncertainty quantification, such as Bayesian deep learning or ensemble methods, would allow the model to report confidence intervals alongside its predictions. This is critical for safety-critical applications and decision-making support systems.

While BILSTM has demonstrated promising results, future studies may explore more advanced architectures such as transformers, attention-based time series models for long term forecasting, or PINNs to capture long-range dependencies and physical constraints more effectively.

By addressing these directions, future research can move closer to enabling real-time, accurate, and interpretable urban wind field estimation systems deployable in practical smart city applications.

Bibliography

[1] P. Mokhasi and D. Rempfer, 'Optimized sensor placement for urban flow measurement', *Physics of Fluids*, vol. 16, no. 5, pp. 1758–1764, 2004, doi: 10.1063/1.1689351.

[2] P. Rajagopalan, K. C. Lim, and E. Jamei, 'Urban heat island and wind flow characteristics of a tropical city', *Solar Energy*, vol. 107, pp. 159–170, 2014, doi: 10.1016/j.solener.2014.05.042.

[3] E. Ng, C. Yuan, L. Chen, C. Ren, and J. C. H. Fung, 'Improving the wind environment in high-density cities by understanding urban morphology and surface roughness: A study in Hong Kong', *Landsc Urban Plan*, vol. 101, no. 1, pp. 59–74, 2011, doi: 10.1016/j.landurbplan.2011.01.004.

[4] M. Chrit, 'Reconstructing urban wind flows for urban air mobility using reduced-order data assimilation', *Theoretical and Applied Mechanics Letters*, vol. 13, no. 4, Jul. 2023, doi: 10.1016/j.taml.2023.100451.

[5] J.-C. Loiseau, B. R. Noack, and S. L. Brunton, 'Sparse reduced-order modeling : Sensor-based dynamics to full-state estimation', Jun. 2017, doi: 10.1017/jfm.2018.147.

[6] L. Li, Y. Fang, L. Liu, H. Peng, J. Kurths, and Y. Yang, 'Overview of compressed sensing: Sensing model, reconstruction algorithm, and its applications', Sep. 01, 2020, *MDPI AG*. doi: 10.3390/app10175909.

[7] M. R. Jovanović, P. J. Schmid, and J. W. Nichols, 'Sparsity-promoting dynamic mode decomposition', *Physics of Fluids*, vol. 26, no. 2, Feb. 2014, doi: 10.1063/1.4863670.

[8] Y. Zhang and Q. Yang, 'An overview of multi-task learning', Jan. 01, 2018, *Oxford University Press*. doi: 10.1093/nsr/nwx105.

[9] S. Brahma, 'Improved Sentence Modeling using Suffix Bidirectional LSTM', May 2018, [Online]. Available: http://arxiv.org/abs/1805.07340

[10] United Nations., 'World Urbanization Prospects - Population Division '.

[11] L. Biao, J. Cunyan, W. Lu, C. Weihua, and L. Jing, 'A parametric study of the effect of building layout on wind flow over an urban area', *Build Environ*, vol. 160, Aug. 2019, doi: 10.1016/j.buildenv.2019.106160.

[12] H. K. Elminir, 'Dependence of urban air pollutants on meteorology', *Science of the Total Environment*, vol. 350, no. 1–3, pp. 225–237, Nov. 2005, doi: 10.1016/j.scitotenv.2005.01.043.

[13] K. Ahmad, M. Khare, and K. K. Chaudhry, 'Wind tunnel simulation studies on dispersion at urban street canyons and intersections - A review', *Journal of Wind Engineering and Industrial Aerodynamics*, vol. 93, no. 9, pp. 697–717, 2005, doi: 10.1016/j.jweia.2005.04.002.

[14] S. Masoumi-Verki, 'Analysis and Reduced-Order Modeling of Urban Airflow and Pollutant Dispersion under Thermal Stratification Conditions', 2022.

[15] Y. D. Huang, R. W. Hou, Z. Y. Liu, Y. Song, P. Y. Cui, and C. N. Kim, 'Effects of wind direction on the airflow and pollutant dispersion inside a long street canyon', *Aerosol Air Qual Res*, vol. 19, no. 5, pp. 1152–1171, May 2019, doi: 10.4209/aaqr.2018.09.0344.

[16] H. Qin, P. Lin, S. S. Y. Lau, and D. Song, ‘Influence of site and tower types on urban natural ventilation performance in high-rise high-density urban environment’, *Build Environ*, vol. 179, Jul. 2020, doi: 10.1016/j.buildenv.2020.106960.

[17] J. Yang, B. Shi, Y. Shi, S. Marvin, Y. Zheng, and G. Xia, ‘Air pollution dispersal in high density urban areas: Research on the triadic relation of wind, air pollution, and urban form’, *Sustain Cities Soc*, vol. 54, Mar. 2020, doi: 10.1016/j.scs.2019.101941.

[18] X. Xu, Q. Yang, A. Yoshida, and Y. Tamura, ‘Characteristics of pedestrian-level wind around super-tall buildings with various configurations’, *Journal of Wind Engineering and Industrial Aerodynamics*, vol. 166, pp. 61–73, Jul. 2017, doi: 10.1016/j.jweia.2017.03.013.

[19] T. van Druenen, T. van Hooff, H. Montazeri, and B. Blocken, ‘CFD evaluation of building geometry modifications to reduce pedestrian-level wind speed’, *Build Environ*, vol. 163, Oct. 2019, doi: 10.1016/j.buildenv.2019.106293.

[20] Y. Du and C. M. Mak, ‘Improving pedestrian level low wind velocity environment in high-density cities: A general framework and case study’, *Sustain Cities Soc*, vol. 42, pp. 314–324, Oct. 2018, doi: 10.1016/j.scs.2018.08.001.

[21] B. Hong and B. Lin, ‘Numerical studies of the outdoor wind environment and thermal comfort at pedestrian level in housing blocks with different building layout patterns and trees arrangement’, *Renew Energy*, vol. 73, pp. 18–27, Jan. 2015, doi: 10.1016/j.renene.2014.05.060.

[22] S. Khoshdel Nikkho, M. Heidarinejad, J. Liu, and J. Srebric, ‘Quantifying the impact of urban wind sheltering on the building energy consumption’, *Appl Therm Eng*, vol. 116, pp. 850–865, 2017, doi: 10.1016/j.applthermaleng.2017.01.044.

[23] *ICPRE 2016 : 2016 IEEE International Conference on Power and Renewable Energy : October 21-23, 2016, Shanghai, China*. IEEE, 2017.

[24] J. Cheng, D. Qi, A. Katal, L. (Leon) Wang, and T. Stathopoulos, ‘Evaluating wind-driven natural ventilation potential for early building design’, *Journal of Wind Engineering and Industrial Aerodynamics*, vol. 182, pp. 160–169, Nov. 2018, doi: 10.1016/j.jweia.2018.09.017.

[25] Z. Huifen, Y. Fuhua, and Z. Qian, ‘Research on the Impact of Wind Angles on the Residential Building Energy Consumption’, *Math Probl Eng*, vol. 2014, 2014, doi: 10.1155/2014/794650.

[26] T. Stathopoulos *et al.*, ‘Urban wind energy: Some views on potential and challenges’, *Journal of Wind Engineering and Industrial Aerodynamics*, vol. 179, pp. 146–157, Aug. 2018, doi: 10.1016/j.jweia.2018.05.018.

[27] F. Toja-Silva, T. Kono, C. Peralta, O. Lopez-Garcia, and J. Chen, ‘A review of computational fluid dynamics (CFD) simulations of the wind flow around buildings for urban wind energy exploitation’, *Journal of Wind Engineering and Industrial Aerodynamics*, vol. 180, pp. 66–87, Sep. 2018, doi: 10.1016/j.jweia.2018.07.010.

[28] K. C. S. Kwok and G. Hu, ‘Wind energy system for buildings in an urban environment’, *Journal of Wind Engineering and Industrial Aerodynamics*, vol. 234, Mar. 2023, doi: 10.1016/j.jweia.2023.105349.

[29] A. KC, J. Whale, and T. Urmee, ‘Urban wind conditions and small wind turbines in the built environment: A review’, Feb. 01, 2019, *Elsevier Ltd*. doi: 10.1016/j.renene.2018.07.050.

[30] C. A. Ku and H. K. Tsai, ‘Evaluating the influence of urban morphology on urban wind environment based on computational fluid dynamics simulation’, *ISPRS Int J Geoinf*, vol. 9, no. 6, Jun. 2020, doi: 10.3390/ijgi9060399.

[31] S. Giersch, O. El Guernaoui, S. Raasch, M. Sauer, and M. Palomar, ‘Atmospheric flow simulation strategies to assess turbulent wind conditions for safe drone operations in urban environments’, *Journal of Wind Engineering and Industrial Aerodynamics*, vol. 229, Oct. 2022, doi: 10.1016/j.jweia.2022.105136.

[32] M. Al Labbad, A. Wall, G. L. Larose, F. Khouli, and H. Barber, ‘Experimental investigations into the effect of urban airflow characteristics on urban air mobility applications’, *Journal of Wind Engineering and Industrial Aerodynamics*, vol. 229, Oct. 2022, doi: 10.1016/j.jweia.2022.105126.

[33] M. Chrit and M. Majdi, ‘Improving Wind Speed Forecasting for Urban Air Mobility Using Coupled Simulations’, *Advances in Meteorology*, vol. 2022, 2022, doi: 10.1155/2022/2629432.

[34] H. Jung and J. Kim, ‘Drone scheduling model for delivering small parcels to remote islands considering wind direction and speed’, *Comput Ind Eng*, vol. 163, Jan. 2022, doi: 10.1016/j.cie.2021.107784.

[35] R. Balazova, J. Hlinka, P. Gabrlik, A. Santus, and S. Ferrari, ‘ENHANCING UAS SAFETY THROUGH BUILDING-INDUCED DANGEROUS ZONES PREDICTION: CONCEPT AND SIMULATIONS’, *Aviation*, vol. 28, no. 4, pp. 279–291, Nov. 2024, doi: 10.3846/aviation.2024.22718.

[36] M. Chrit and M. Majdi, ‘Operational wind and turbulence nowcasting capability for advanced air mobility’, *Neural Comput Appl*, vol. 36, no. 18, pp. 10637–10654, Jun. 2024, doi: 10.1007/s00521-024-09614-0.

[37] D. S. Nithya, G. Quaranta, V. Muscarello, and M. Liang, ‘Review of Wind Flow Modelling in Urban Environments to Support the Development of Urban Air Mobility’, Apr. 01, 2024, *Multidisciplinary Digital Publishing Institute (MDPI)*. doi: 10.3390/drones8040147.

[38] P. Ngae, H. Kouichi, P. Kumar, A. A. Feiz, and A. Chpoun, ‘Optimization of an urban monitoring network for emergency response applications: An approach for characterizing the source of hazardous releases’, *Quarterly Journal of the Royal Meteorological Society*, vol. 145, no. 720, pp. 967–981, Apr. 2019, doi: 10.1002/qj.3471.

[39] S. Masoumi-Verki, F. Haghighat, and U. Eicker, ‘A review of advances towards efficient reduced-order models (ROM) for predicting urban airflow and pollutant dispersion’, May 15, 2022, *Elsevier Ltd*. doi: 10.1016/j.buildenv.2022.108966.

[40] G. Padula, M. Girfoglio, and G. Rozza, ‘A brief review of Reduced Order Models using intrusive and non-intrusive techniques’, Jun. 2024, [Online]. Available: http://arxiv.org/abs/2406.00559

[41] C. Bonneville *et al.*, ‘A Comprehensive Review of Latent Space Dynamics Identification Algorithms for Intrusive and Non-Intrusive Reduced-Order-Modeling’, 2024.

[42] C. W. Rowley, T. Colonius, and R. M. Murray, ‘Model reduction for compressible flows using POD and Galerkin projection’, *Physica D*, vol. 189, no. 1–2, pp. 115–129, Feb. 2004, doi: 10.1016/j.physd.2003.03.001.

[43] J. Yu, C. Yan, and M. Guo, ‘Non-intrusive reduced-order modeling for fluid problems: A brief review’, *Proc Inst Mech Eng G J Aerosp Eng*, vol. 233, no. 16, pp. 5896–5912, Dec. 2019, doi: 10.1177/0954410019890721.

[44] M. Velagar, C. Keller, and J. N. Kutz, ‘Optimized Dynamic Mode Decomposition for Reconstruction and Forecasting of Atmospheric Chemistry Data’, Oct. 01, 2024. doi: 10.5194/gmd-2024-77.

[45] C. Y. Li *et al.*, ‘Best practice for the dynamic mode decomposition in wind engineering applications’. [Online]. Available: https://www.researchgate.net/publication/365872801

[46] B. Zhang, L. Zhou, T. K. T. Tse, L. Wang, J. Niu, and C. M. Mak, ‘Extended spectral proper orthogonal decomposition for analysis of correlated surrounding flow structures and wind load components of a building’, *Journal of Wind Engineering and Industrial Aerodynamics*, vol. 240, Sep. 2023, doi: 10.1016/j.jweia.2023.105512.

[47] Y. Su, J. Di, J. Li, and F. Xia, ‘Wind Pressure Field Reconstruction and Prediction of Large-Span Roof Structure with Folded-Plate Type Based on Proper Orthogonal Decomposition’, *Applied Sciences (Switzerland)*, vol. 12, no. 17, Sep. 2022, doi: 10.3390/app12178430.

[48] S. Xiang *et al.*, ‘Fast simulation of high resolution urban wind fields at city scale’, *Urban Clim*, vol. 39, Sep. 2021, doi: 10.1016/j.uclim.2021.100941.

[49] W. Jiang *et al.*, ‘Applicability analysis of transformer to wind speed forecasting by a novel deep learning framework with multiple atmospheric variables’, *Appl Energy*, vol. 353, Jan. 2024, doi: 10.1016/j.apenergy.2023.122155.

[50] L. Ø. Bentsen, N. D. Warakagoda, R. Stenbro, and P. Engelstad, ‘Spatio-Temporal Wind Speed Forecasting using Graph Networks and Novel Transformer Architectures’, Aug. 2022, doi: 10.1016/j.apenergy.2022.120565.

[51] Z. Luo, L. Wang, J. Xu, M. Chen, J. Yuan, and A. C. C. Tan, ‘Flow reconstruction from sparse sensors based on reduced-order autoencoder state estimation’, *Physics of Fluids*, vol. 35, no. 7, Jul. 2023, doi: 10.1063/5.0155039.

[52] D. Wang, F. Xie, T. Ji, X. Zhang, Y. Lu, and Y. Zheng, ‘Prediction of wind shear layer for dynamic soaring by using proper orthogonal decomposition and long short term memory network’, *Physics of Fluids*, vol. 35, no. 8, Aug. 2023, doi: 10.1063/5.0160035.

[53] D. Gomes, A. E. Santo, and J. C. Páscoa, ‘THEORETICAL FRAMEWORK AND USE OF CNN RECONSTRUCTION WITH OPTIMAL SPARSE SENSOR PLACEMENT IN A FLOW FIELD’, 2024. [Online]. Available: http://asmedigitalcollection.asme.org/IMECE/proceedings-pdf/IMECE2024/88667/V008T10A007/7427676/v008t10a007-imece2024-139077.pdf

[54] A. Chatterjee, ‘An introduction to the proper orthogonal decomposition’, 2000. [Online]. Available: https://www.jstor.org/stable/24103957

[55] L. SIROVICH, ‘TURBULENCE AND THE DYNAMICS OF COHERENT STRUCTURES PART III: DYNAMICS AND SCALING’, *Q Appl Math*, vol. 45, no. 3, pp. 583–590, 1987, [Online]. Available: http://www.jstor.org/stable/43637459

[56] G. Berkooz, P. Holmes, and J. L. Lumley, ‘The proper orthogonal decomposition in the analysis of turbulent flows’, *Annu Rev Fluid Mech*, vol. 25, no. 1, pp. 539–575, 1993, doi: 10.1146/annurev.fl.25.010193.002543.

[57] Z. Krawczyk, R. K. S. S. Vuppala, R. Paul, and K. Kara, ‘Evaluating Reduced-Order Urban Wind Models for Simulating Flight Dynamics of Advanced Aerial Mobility Aircraft’, *Aerospace*, vol. 11, no. 10, Oct. 2024, doi: 10.3390/aerospace11100830.

[58] N. Zhao, Y. Jiang, L. Peng, and X. Chen, ‘Fast simulation of nonstationary wind velocity fields by proper orthogonal decomposition interpolation’, *Journal of Wind Engineering and Industrial Aerodynamics*, vol. 219, Dec. 2021, doi: 10.1016/j.jweia.2021.104798.

[59] B. Zhang, L. Zhou, T. K. T. Tse, L. Wang, J. Niu, and C. M. Mak, ‘Extended spectral proper orthogonal decomposition for analysis of correlated surrounding flow structures and wind load components of a building’, *Journal of Wind Engineering and Industrial Aerodynamics*, vol. 240, Sep. 2023, doi: 10.1016/j.jweia.2023.105512.

[60] Y. Liu, C. H. Liu, G. P. Brasseur, and C. Y. H. Chao, ‘Proper orthogonal decomposition of large-eddy simulation data over real urban morphology’, *Sustain Cities Soc*, vol. 89, Feb. 2023, doi: 10.1016/j.scs.2022.104324.

[61] C. Ebert, J. Weiss, M. U. de Haag, C. Ruwisch, and F. J. Silvestre, ‘Trajectory Planning in Windy Urban Environment Using Gappy Proper Orthogonal Decomposition for Wind Estimates’, *AIAA Journal*, vol. 61, no. 6, pp. 2640–2651, 2023, doi: 10.2514/1.J062049.

[62] P. J. Schmid, ‘Dynamic mode decomposition of numerical and experimental data’, *J Fluid Mech*, vol. 656, pp. 5–28, 2010, doi: 10.1017/S0022112010001217.

[63] A. G. Nair, B. Strom, B. W. Brunton, and S. L. Brunton, ‘Phase-consistent dynamic mode decomposition from multiple overlapping spatial domains’, *Phys Rev Fluids*, vol. 5, no. 7, Jul. 2020, doi: 10.1103/PhysRevFluids.5.074702.

[64] ‘Chapter 3: Koopman Analysis’, in *Dynamic Mode Decomposition*, in Other Titles in Applied Mathematics. , Society for Industrial and Applied Mathematics, 2016, pp. 39–53. doi: doi:10.1137/1.9781611974508.ch3.

[65] P. J. Schmid, L. Li, M. P. Juniper, and O. Pust, ‘Applications of the dynamic mode decomposition’, *Theor Comput Fluid Dyn*, vol. 25, no. 1–4, pp. 249–259, Jun. 2011, doi: 10.1007/s00162-010-0203-9.

[66] M. O. Williams, I. G. Kevrekidis, and C. W. Rowley, ‘A Data–Driven Approximation of the Koopman Operator: Extending Dynamic Mode Decomposition’, *J Nonlinear Sci*, vol. 25, no. 6, pp. 1307–1346, Dec. 2015, doi: 10.1007/s00332-015-9258-5.

[67] S. Le Clainche and J. M. Vega, ‘Higher Order Dynamic Mode Decomposition’, *SIAM J Appl Dyn Syst*, vol. 16, no. 2, pp. 882–925, Jan. 2017, doi: 10.1137/15M1054924.

[68] E. Lazpita, Á. Martínez-Sánchez, A. Corrochano, S. Hoyas, S. Le Clainche, and R. Vinuesa, 'On the generation and destruction mechanisms of arch vortices in urban fluid flows', *Physics of Fluids*, vol. 34, no. 5, May 2022, doi: 10.1063/5.0088305.

[69] E. Ghadimi, A. Teixeira, M. G. Rabbat, and M. Johansson, 'The ADMM algorithm for distributed averaging: Convergence rates and optimal parameter selection'.

[70] J. N. Kutz, X. Fu, and S. L. Brunton, 'Multi-Resolution Dynamic Mode Decomposition', Jun. 2015, [Online]. Available: http://arxiv.org/abs/1506.00564

[71] K. Manohar, E. Kaiser, S. L. Brunton, and J. N. Kutz, 'Optimized Sampling for Multiscale Dynamics', Dec. 2017, doi: 10.1137/17M1162366.

[72] H. Gao *et al.*, 'An optimal sensor placement scheme for wind flow and pressure field monitoring', *Build Environ*, vol. 244, Oct. 2023, doi: 10.1016/j.buildenv.2023.110803.

[73] M. M. Kelp, S. Lin, J. N. Kutz, and L. J. Mickley, 'A new approach for determining optimal placement of PM2.5air quality sensors: Case study for the contiguous United States', *Environmental Research Letters*, vol. 17, no. 3, Mar. 2022, doi: 10.1088/1748-9326/ac548f.

[74] K. K. Chen, J. H. Tu, and C. W. Rowley, 'Variants of dynamic mode decomposition: Boundary condition, Koopman, and fourier analyses', *J Nonlinear Sci*, vol. 22, no. 6, pp. 887–915, Dec. 2012, doi: 10.1007/s00332-012-9130-9.

[75] P. J. Baddoo, B. Herrmann, B. J. McKeon, J. Nathan Kutz, and S. L. Brunton, 'Physics-informed dynamic mode decomposition', *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 479, no. 2271, Mar. 2023, doi: 10.1098/rspa.2022.0576.

[76] D. Duke, J. Soria, and D. Honnery, 'An error analysis of the dynamic mode decomposition', *Exp Fluids*, vol. 52, no. 2, pp. 529–542, Feb. 2012, doi: 10.1007/s00348-011-1235-7.

[77] K. Fukami, K. Fukagata, and K. Taira, 'Super-resolution reconstruction of turbulent flows with machine learning', *J Fluid Mech*, vol. 870, pp. 106–120, Jul. 2019, doi: 10.1017/jfm.2019.238.

[78] S. Masoumi-Verki, F. Haghighat, and U. Eicker, 'Improving the performance of a CAE-based reduced-order model for predicting turbulent airflow field around an isolated high-rise building', *Sustain Cities Soc*, vol. 87, Dec. 2022, doi: 10.1016/j.scs.2022.104252.

[79] K. Fukami, T. Nakamura, and K. Fukagata, 'Convolutional neural network based hierarchical autoencoder for nonlinear mode decomposition of fluid field data', *Physics of Fluids*, vol. 32, no. 9, Sep. 2020, doi: 10.1063/5.0020721.

[80] M. Raissi, P. Perdikaris, and G. E. Karniadakis, 'Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations', *J Comput Phys*, vol. 378, pp. 686–707, Feb. 2019, doi: 10.1016/j.jcp.2018.10.045.

[81] W. Chen, Q. Wang, J. S. Hesthaven, and C. Zhang, 'Physics-informed machine learning for reduced-order modeling of nonlinear problems', *J Comput Phys*, vol. 446, Dec. 2021, doi: 10.1016/j.jcp.2021.110666.

[82] N. Geneva and N. Zabaras, 'Transformers for modeling physical systems', *Neural Networks*, vol. 146, pp. 272–289, Feb. 2022, doi: 10.1016/j.neunet.2021.11.022.

[83] P. Wu, F. Qiu, W. Feng, F. Fang, and C. Pain, 'A non-intrusive reduced order model with transformer neural network and its application', *Physics of Fluids*, vol. 34, no. 11, Nov. 2022, doi: 10.1063/5.0123185.

[84] J. Grigsby, Z. Wang, N. Nguyen, and Y. Qi, 'Long-Range Transformers for Dynamic Spatiotemporal Forecasting', Sep. 2021, [Online]. Available: http://arxiv.org/abs/2109.12218

[85] X. Shao, Z. Liu, S. Zhang, Z. Zhao, and C. Hu, 'PIGNN-CFD: A physics-informed graph neural network for rapid predicting urban wind field defined on unstructured mesh', *Build Environ*, vol. 232, Mar. 2023, doi: 10.1016/j.buildenv.2023.110056.

[86] H. Gao *et al.*, 'Urban wind field prediction based on sparse sensors and physics-informed graph-assisted auto-encoder', *Computer-Aided Civil and Infrastructure Engineering*, 2024, doi: 10.1111/mice.13147.

[87] V. Kag, K. Seshasayanan, and V. Gopinath, 'Physics-informed data based neural networks for two-dimensional turbulence', *Physics of Fluids*, vol. 34, no. 5, p. 055130, May 2022, doi: 10.1063/5.0090050.

[88] S. Hochreiter and J. Schmidhuber, 'Long Short-Term Memory', *Neural Comput*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, doi: 10.1162/neco.1997.9.8.1735.

[89] X. Zhang, T. Ji, F. Xie, H. Zheng, and Y. Zheng, 'Unsteady flow prediction from sparse measurements by compressed sensing reduced order modeling', *Comput Methods Appl Mech Eng*, vol. 393, Apr. 2022, doi: 10.1016/j.cma.2022.114800.

[90] G. Zhang and S. Liu, 'Reconstruction of Unsteady Wind Field Based on CFD and Reduced-Order Model', *Mathematics*, vol. 11, no. 10, May 2023, doi: 10.3390/math11102223.

[91] S. Masoumi-Verki, P. Gholamalipour, F. Haghighat, and U. Eicker, 'Embedded LES of thermal stratification effects on the airflow and concentration fields around an isolated high-rise building: Spectral and POD analyses', *Build Environ*, vol. 206, Dec. 2021, doi: 10.1016/j.buildenv.2021.108388.

[92] 'Advanced Environmental Wind Engineering'.

[93] A. Nikolov and H. Tang, 'General Gaussian Noise Mechanisms and Their Optimality for Unbiased Mean Estimation', Jan. 2023, [Online]. Available: http://arxiv.org/abs/2301.13850

[94] G. Iglesias, E. Talavera, Á. González-Prieto, A. Mozo, and S. Gómez-Canaval, 'Data Augmentation techniques in time series domain: A survey and taxonomy', Feb. 2024, doi: 10.1007/s00521-023-08459-3.

[95] S. Ioffe and C. Szegedy, 'Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift', Feb. 2015, [Online]. Available: http://arxiv.org/abs/1502.03167

[96] D. P. Kingma and J. Ba, 'Adam: A Method for Stochastic Optimization', Dec. 2014, [Online]. Available: http://arxiv.org/abs/1412.6980

[97] K. Gokcesu and H. Gokcesu, 'Generalized Huber Loss for Robust Learning and its Efficient Minimization for a Robust Statistics', Aug. 2021, [Online]. Available: http://arxiv.org/abs/2108.12627

Appendix A: Python Implementation of SP-DMD for GPU Computing

Listing A.1: Data preprocessing for DMD (Removing mean value from each velocity channels)

```python
def data_centering_scaling(data):
    t, rows, cols, c = data.shape
    channel_one = data[:, :, :, 0]
    channel_two = data[:, :, :, 1]

    c_one_mean = np.mean(channel_one)
    c_two_mean = np.mean(channel_two)

    c_one_minus_mean = channel_one - c_one_mean
    c_two_minus_mean = channel_two - c_two_mean

    c_one_min = np.amin(c_one_minus_mean)
    c_one_max = np.amax(c_one_minus_mean)
    c_two_min = np.amin(c_two_minus_mean)
    c_two_max = np.amax(c_two_minus_mean)

    centered_c_one = (c_one_minus_mean - c_one_min) / (c_one_max - c_one_min)
    centered_c_two = (c_two_minus_mean - c_two_min) / (c_two_max - c_two_min)

    dataset = np.zeros((t, rows, cols, c))
    dataset_minus_mean = np.zeros((t, rows, cols, c))
    dataset[:, :, :, 0] = centered_c_one
    dataset[:, :, :, 1] = centered_c_two
    dataset_minus_mean[:, :, :, 0] = c_one_minus_mean
    dataset_minus_mean[:, :, :, 1] = c_two_minus_mean

    return dataset, dataset_minus_mean, (c_one_mean, c_one_min, c_one_max, c_two_mean, c_two_min, c_two_max)
```

```python
def reverse_scaling(centered_data, original_params):
    c_one_mean, c_one_min, c_one_max, c_two_mean, c_two_min, c_two_max = original_params

    centered_c_one = centered_data[:, :, :, 0]
    centered_c_two = centered_data[:, :, :, 1]

    c_one_minus_mean = centered_c_one * (c_one_max - c_one_min) + c_one_min
    c_two_minus_mean = centered_c_two * (c_two_max - c_two_min) + c_two_min

    channel_one = c_one_minus_mean + c_one_mean
    channel_two = c_two_minus_mean + c_two_mean

    original_data = np.zeros(centered_data.shape)
    original_data[:, :, :, 0] = channel_one
    original_data[:, :, :, 1] = channel_two

    return original_data
```

```python
centered_data, dataset_minus_mean, original_params = data_centering_scaling(example_data_train)
reversed_data = reverse_scaling(centered_data, original_params)

# Check if the reversed data is close to the original data
print(np.allclose(example_data_train, reversed_data))
```

```
True
```

Listing A.2: DMD analysis

```
import numpy as np
import cupy as cp
```

```
X = data_flatten[:, :-1]
X_prime = data_flatten[:, 1:]

# Convert the NumPy array to a CuPy array
X_gpu = cp.array(X)
X_prime_gpu = cp.array(X_prime)

# Perform SVD using CuPy for GPU acceleration
U, S, VT = cp.linalg.svd(X_gpu, full_matrices=False)

U_r = U[:, :desired_rank]
S_r = cp.diag(S)[:desired_rank,:desired_rank]
Vt_r = VT.conj().T[:,:desired_rank]

UstarX1 = U.conj()

# Converting A to the Reduced Dimension Atilde
A_t = U_r.conj().T @ X_prime_gpu @ Vt_r @ cp.linalg.pinv(S_r)

A_t = A_t.get()

# Performig egiendecomposition on Atilde
Ydmd, Ddmd = np.linalg.eig(A_t)

eigval = cp.array(Ydmd)
eigvec = cp.array(Ddmd)

# DMD spatial modes
Phi = X_prime_gpu @ Vt_r @ cp.linalg.pinv(S_r) @ eigvec

# Vandermonde Matrix: Time Evolution of Modes
Vander = np.vander(eigval.get(), len(Vt_r), increasing=True)
```

Listing A.3: SP-DMD Algorithm Function (translated from MATLAB [7] into Python and optimized for GPU computing).

```
import cupy as cp
import cupyx
from cupyx.scipy.sparse import csc_matrix
from cupyx.scipy.sparse import hstack as sphstack
from cupyx.scipy.sparse import vstack as spvstack

from scipy.sparse import csc_matrix as cpu_csc_matrix
from scipy.sparse import hstack as cpu_hstack, vstack as cpu_vstack
from scipy.sparse.linalg import spsolve
```

```
L = eigvec
R = Vander
G = S_r @ Vt_r.T  # Using the @ operator for matrix multiplication

L = L.get()
G = G.get()

# Element-wise multiplication of (L' * L) with the conjugate transpose of (R * R')
# In NumPy, .conj().T is used for conjugate transpose and * is used for element-wise multiplication
P = np.multiply((L.conj().T @ L), np.conj(np.dot(R, R.conj().T)))

# Diagonal of the product (R * G' * L) and then take the conjugate
q = np.conj(np.diag(R @ G.conj().T @ L))

# Trace of G' * G. np.trace() computes the trace of a matrix
s = np.trace(G.conj().T @ G)

# creating the penelzie parameter  for 300 values in logarithmic range
gamma_vec = np.logspace(np.log10(0.05), np.log10(100000), 300)

# Converting the parameter arrays to CuPy arrays for GPU-accelerated computation
Ps = cp.asarray(P)
qs = cp.asarray(q)
ss = cp.asarray(s)
gammaval_s = cp.asarray(gamma_vec)
```

```
def sp_dmd(P, q, s, gammaval, options=None):
  default_options = {
      'rho': 1,
      'maxiter': 50000,
      'eps_abs': 1e-6,
      'eps_rel': 1e-4
  }

  if options is None:
    options = default_options

  P = cp.squeeze(P)
  q = cp.squeeze(q)[:,cp.newaxis]
  gammaval = cp.squeeze(gammaval)

  if P is None or q is None or s is None or gammaval is None:
    raise ValueError("Four input arguments are required to run the code!")

  if P.ndim != 2:
    raise ValueError('invalid P')
  if q.ndim != 2:
    raise ValueError('invalid q')
  if gammaval.ndim != 1:
    raise ValueError('invalid gammaval')

  # Extracting values for options
  rho = options.get('rho', default_options['rho'])
  max_ADMM_iter = options.get('maxiter', default_options['maxiter'])
  eps_abs = options.get('eps_abs', default_options['eps_abs'])
  eps_rel = options.get('eps_rel', default_options['eps_rel'])

  # Number of optimization variables
  n = len(q)

  # Identity matrix
  I = cp.eye(n)

  # Create a new type (or class) and then instantiate
  # It indicates that our new class inherits from the base "object" class
  answer = type('ADMMAnswer', (object,), {})()
```

```
  # Assign values to the dictionary keys
  answer.gamma = gammaval
  answer.Nz = cp.zeros([len(gammaval),])  #number of non-zero amplitudes
  answer.Jsp = cp.zeros([len(gammaval),]) #square of Frobenius norm (before polishing)
  answer.Jpol = cp.zeros([len(gammaval),])  #square of Frobenius norm (after polishing)
  answer.Ploss = cp.zeros([len(gammaval),]) #optimal performance loss (after polishing)
  answer.xsp = cp.zeros([n, len(gammaval)], dtype='complex')  #vector of amplitudes (before polishing)
  answer.xpol = cp.zeros([n, len(gammaval)], dtype='complex')  #vector of amplitudes (after polishing)

  # Cholesky factorization of matrix P + (rho/2)I

  Prho = P + (rho/2) * I
  Plow = cp.linalg.cholesky(Prho)
  Plow_star = Plow.conj().T

# Convert matrix P into sparse representation (for KKT system)
Psparse = cupyx.scipy.sparse.csc_matrix(P)


for i, gamma in enumerate(gammaval):
  # initialization
  y = cp.zeros([n, 1])  # represents Lagrange multipliers
  z = cp.zeros([n, 1])  # a copy or an estimate of the optimization variable x



  # ADMM to solve gamma-parameterized problem
  for step in range(max_ADMM_iter):
    # x-minimization step

    u = z - (1/rho) * y
    xnew = cp.linalg.solve(Plow_star, cp.linalg.solve(Plow, q + (rho/2) * u))

    # z-minimization step

    a = (gamma/rho) * cp.ones([n, 1])
    v = xnew + (1/rho) * y

    znew = ((1 - a / cp.abs(v)) * v) * (cp.abs(v) > a)

    # Primal and dual residuals

    res_prim = cp.linalg.norm(xnew - znew, 2)
    res_dual = rho * cp.linalg.norm(znew - z, 2)

    print(f"Iteration {step}: Residual Prime : {res_prim}, Residual Dual : {res_dual}", end="\r")


    # Lagrangian multiplier update step
    y = y + rho * (xnew - znew)

    # Stopping criteria
    eps_prim = cp.sqrt(n) * eps_abs + eps_rel * max([cp.linalg.norm(xnew, 2),
                                                     cp.linalg.norm(znew, 2)])
    eps_dual = cp.sqrt(n) * eps_abs + eps_rel * cp.linalg.norm(y)


    if (res_prim < eps_prim) and (res_dual < eps_dual):
      break
    else:
      z = znew

  print("****Gamma value****:", gamma)

  # Recording results
  answer.xsp[:, i] = cp.squeeze(z)  #vector of amplitudes
  print("****number of nonzero amplitudes****: ", cp.count_nonzero(z))
  answer.Nz[i] = cp.count_nonzero(answer.xsp[:, i]) #number of non-zero amplitudes
  answer.Jsp[i] = (cp.real(z.conj().T @ P @ z) - 2 * cp.real(q.conj().T @ z) + s)
```

```
    # Polishing of the non-zero amplitudes
    ind_zero = cp.flatnonzero(abs(z) < 1e-12)  # find indices of zero elements
    print("number of nonzero elements within 10^-10 and 10^-8: ", len(cp.flatnonzero((abs(z) > 1e-10) & (abs(z) < 1e-8))))
    print("number of nonzero elements within 10^-8 and 10^-6: ", len(cp.flatnonzero((abs(z) > 1e-8) & (abs(z) < 1e-6))))
    print("number of nonzero elements within 10^-6 and 10^-4: ", len(cp.flatnonzero((abs(z) > 1e-6) & (abs(z) < 1e-4))))
    print("number of nonzero elements within 10^-4 and 10^-2: ", len(cp.flatnonzero((abs(z) > 1e-4) & (abs(z) < 1e-2))))
    print("number of nonzero elements within 10^-2 and 10^-1: ", len(cp.flatnonzero((abs(z) > 1e-2) & (abs(z) < 1e-1))))
    print("number of nonzero elements within 10^-2 and 10^-1: ", len(cp.flatnonzero((abs(z) > 1e-1) & (abs(z) < 1))))
    m = len(ind_zero) # find number of zero elements
    print("****number of zero amplitudes:**** ", m)

    if m > 0:
      # From the constraint matrix E for E.T @ x = 0
      E = cp.eye(n)[:, ind_zero]
      E = cupyx.scipy.sparse.csc_matrix(E, dtype='complex')

      # From KKT system for the optimality conditions
      KKT = spvstack([
          sphstack([Psparse, E], format='csc'),
          sphstack([E.conj().T, csc_matrix((m, m), dtype='complex')], format='csc')
      ], format='csc')

      rhs = cp.vstack([q, cp.zeros([m, 1], dtype='complex')])

      # Solve KKT system
      sol = spsolve(KKT, rhs)
    else:
      sol = cp.linalg.solve(P, q)

    #print("xpol nonzero: ", cp.count_nonzero(abs(sol)))
    # Vector of polished (optimal) amplitudes
    xpol = sol[:n]

    # Record output data
    answer.xpol[:, i] = cp.squeeze(xpol)

    # Polished (optimal) least-squares residual
    answer.Jpol[i] = (
        cp.real(cp.dot(cp.dot(xpol.conj().T, P), xpol)) - 2 * cp.real(cp.dot(q.conj().T, xpol)) + s)

    # Polished (optimal) performance loss
    answer.Ploss[i] = 100 * cp.sqrt(answer.Jpol[i] / s)
    print("Ploss:", answer.Ploss[i])

    print("-----------------------------------iteration number-----------------------------------:",i+1)
  return answer, m


answer_sh, m = sp_dmd(P, q, s, gammaval_s, options=None)
```

Appendix B: Python Implementation of Dynamic Coefficients Encoder–Decoder Scheme

Listing B.1: Encoder

```
import numpy as np
```

```
def remove_column_zeros(X: np.ndarray, b: int, zero_value: float = 0.0) -> tuple[np.ndarray, np.ndarray]:
    """
    Remove exactly `b` zeros per column from a 2D array.

    Args:
        X (np.ndarray): Input 2D array of shape (n, m).
        b (int): Number of zero entries expected per column.
        zero_value (float, optional): Value to treat as "zero". Default is 0.0.

    Returns:
        non_zero_data (np.ndarray): Array of shape (n - b, m) with zeros removed.
        zero_locations (np.ndarray): Boolean mask of shape (n, m), True where entries == zero_value.

    Raises:
        AssertionError: If any column does not contain exactly `b` zeros.
    """
    if X.ndim != 2:
        raise ValueError("Input X must be a 2D array.")

    # Identify zero locations
    zero_locations = (X == zero_value)

    # Check each column has exactly b zeros
    counts = np.sum(zero_locations, axis=0)
    assert np.all(counts == b), f"Each column must have exactly {b} zero values; got {counts}."

    # Collect nonzero values column by column
    non_zero_cols = [col[~mask] for col, mask in zip(X.T, zero_locations.T)]
    non_zero_data = np.stack(non_zero_cols, axis=1)

    return non_zero_data, zero_locations
```

```
def isolated_coeff_remover(matrix, rows_to_remove):

      # Create a new matrix with the removed rows
      removed_rows = matrix[:, rows_to_remove]

      # Create a matrix with the remaining rows
      remaining_matrix = np.delete(matrix, rows_to_remove, axis=1)

      print("Original Matrix:\n", matrix.shape)
      print("Removed Rows:\n", removed_rows.shape)
      print("Remaining Matrix:\n", remaining_matrix.shape)
      return removed_rows, remaining_matrix
```

```
def Psi_modifier_MTL(Psi_train, Psi_test):

    # isloted real number index
    rows_to_remove = np.where(abs(Psi_train.imag[0]) < 1e-10)[0]
    print("Isolated row index which removed :",rows_to_remove)


    y_removed_train, y_mirrored_train = isolated_coeff_remover(y_train, rows_to_remove)
    y_removed_test, y_mirrored_test = isolated_coeff_remover(y_test, rows_to_remove)
    print("removed first time sample train :",y_removed_train[0])

        # Extract the even column indices (0, 2, 4, ...)
    even_columns_train = y_mirrored_train[:, ::2]
    even_columns_test = y_mirrored_test[:, ::2]


    train_real_part_noready = even_columns_train.real
    train_real_part = np.concatenate((train_real_part_noready, y_removed_train.real), axis=-1)
    train_imaginary_part = even_columns_train.imag

    test_real_part_noready = even_columns_test.real
    test_real_part = np.concatenate((test_real_part_noready, y_removed_test.real), axis=-1)
    test_imaginary_part = even_columns_test.imag

    print(even_columns_train.shape)

    print("task 1 (real part) train shape :",train_real_part.shape)
    print("task 2 (imaginary part) train shape :",train_imaginary_part.shape)
    print("task 1 (real part) test shape :",test_real_part.shape)
    print("task 2 (imaginary part) test shape :",test_imaginary_part.shape)
    return train_real_part, train_imaginary_part, test_real_part, test_imaginary_part, rows_to_remove
```

```
## y_train are the y_test are the outputs of the MTL model (complex_valued dynamic coefficients) in train and test set
train_real_part, train_imaginary_part, test_real_part, test_imaginary_part, rows_to_remove = Psi_modifier_MTL(y_train, y_test)
```

Listing B.2: Decoder

```python
def matrix_modifier_deisloted(matrix,n):


      # Remove the last n rows
      removed_columns = matrix[:, -n:]

      # Remaining matrix after removing the last n rows
      remaining_matrix = matrix[:, :int(int(matrix.shape[1]-n))]

      print(removed_columns.shape)
      print(remaining_matrix.shape)
      return removed_columns, remaining_matrix
```

```python
def Psi_recunstruct_MTL (Psi_real, Psi_imag, rows_to_remove):
  ##### as only real part has more features :
  isloted_coeff, real_coeff = matrix_modifier_deisloted(Psi_real, rows_to_remove.shape[0])

  complex_removed = isloted_coeff + 1j * 0
  # Repeat each element in the columns for real part [a,b,c],  to : [a,a,b,b,c,c]
  predicted_real_scenario = np.repeat(real_coeff, 2, axis=1)



  # the imaginary part process [a,b,c],  to : [a,-a,b,-b,c,-c]
  # Create the negated matrix
  negated_matrix = -Psi_imag

  # Interleave the original matrix with the negated matrix along the columns
  predicted_imag_scenario = np.empty((Psi_imag.shape[0], Psi_imag.shape[1] * 2), dtype=Psi_imag.dtype)
  predicted_imag_scenario[:, 0::2] = Psi_imag
  predicted_imag_scenario[:, 1::2] = negated_matrix



  ###### complex matrix
  y_pred_complex = predicted_real_scenario + 1j * predicted_imag_scenario

  where_to_add_isoltaed = rows_to_remove[0]-1

  first_part = y_pred_complex[:, :where_to_add_isoltaed+1]
  second_part = y_pred_complex[:, where_to_add_isoltaed+1:]

  # Concatenate the parts with the removed rows in between
  y_pred_complex_final = np.concatenate((first_part, complex_removed, second_part), axis=1)
  print('Psi with zero valus removed shape : ', y_pred_complex_final.shape)
  return y_pred_complex_final
```

```python
def zero_add_psi (Psi, zero_locations, n):
      y_pred_complexT = Psi.T
      y_pred_complexT.shape
      m = Psi.shape[0]
      zero_locationsnew = zero_locations[:,:m]
      # Initialize an array with zeros ################################################

      restored_data = np.zeros((n, m),dtype=complex)
      # Place the modified non-zero values back into their original locations
      for col in range(m):
          non_zero_indices = ~zero_locationsnew[:, col]
          restored_data[non_zero_indices, col] = y_pred_complexT[:, col]

      print("Restored Data:\n", restored_data[:,0])
      return restored_data
```

```python
def create_final_data(Phi, restored_data, original_params):

  recunstructed_data = dot(Phi, restored_data)
  recunstructed_data = recunstructed_data.real
  recon_data_reshape = np.reshape(recunstructed_data.T, (restored_data.shape[1],60,160,2))
  reversed_data_resc = reverse_scaling(recon_data_reshape, original_params)

  print(reversed_data_resc.shape)


  return reversed_data_resc
```

```python
def recunstroter_data(data):

  data_real = data[0]
  data_imag = data[1]

  y_pred_complex_final = Psi_recunstruct_MTL(data_real, data_imag, rows_to_remove)
  restored_data = zero_add_psi(y_pred_complex_final, zero_locations, 219)

  reversed_data_resc = create_final_data(Phi, restored_data, original_params)

  reversed_data_resc_uH = reversed_data_resc/1.37

  return reversed_data_resc_uH, reversed_data_resc
```

Appendix C: Python Codes of Feedforward MTL Model

```
import numpy as np
import tensorflow as tf

from tensorflow.keras import layers, Model, losses, metrics
from tensorflow.keras.losses import Hube

def multi_task_model(num_features,
                     outputfeatures1,
                     outputfeatures2,
                     shared_dense_units_list,
                     regression1_dense_units_list,
                     regression2_dense_units_list,
                     dropout_shared,
                     dropout_realpart,
                     dropout_imagpart,
                     batch_norm=False,
                     leaky_relu_alpha=0.2):

    # Input
    input_layer = layers.Input(shape=(num_features,), name='input')

    # Initial Batch Normalization
    shared_x = layers.BatchNormalization(name='bn_input')(input_layer)

    # Shared Layers
    for i, dense_units in enumerate(shared_dense_units_list):
        shared_x = layers.Dense(units=dense_units, name=f'shared_dense_{i+1}')(shared_x)
        shared_x = layers.LeakyReLU(alpha=leaky_relu_alpha, name=f'shared_leaky_relu_{i+1}')(shared_x)
        if batch_norm:
            shared_x = layers.BatchNormalization(name=f'shared_bn_{i+1}')(shared_x)
        shared_x = layers.Dropout(dropout_shared, name=f'shared_dropout_{i+1}')(shared_x)

    # Regression Head 1
    if len(regression1_dense_units_list) > 0:
        regr1_x = layers.Dense(units=regression1_dense_units_list[0], name='regr1_dense_1')(shared_x)
        regr1_x = layers.LeakyReLU(alpha=leaky_relu_alpha, name='regr1_leaky_relu_1')(regr1_x)
        for i, dense_units in enumerate(regression1_dense_units_list[1:]):
            regr1_x = layers.Dense(units=dense_units, name=f'regr1_dense_{i+2}')(regr1_x)
            regr1_x = layers.LeakyReLU(alpha=leaky_relu_alpha, name=f'regr1_leaky_relu_{i+2}')(regr1_x)
            if batch_norm:
                regr1_x = layers.BatchNormalization(name=f'regr1_bn_{i+2}')(regr1_x)
            regr1_x = layers.Dropout(dropout_realpart, name=f'regr1_dropout_{i+2}')(regr1_x)
        regr1_output = layers.Dense(outputfeatures1, activation='linear', name='regr1_output')(regr1_x)
    else:
        regr1_output = layers.Dense(outputfeatures1, activation='linear', name='regr1_output')(shared_x)

    # Regression Head 2
    if len(regression2_dense_units_list) > 0:
        regr2_x = layers.Dense(units=regression2_dense_units_list[0], name='regr2_dense_1')(shared_x)
        regr2_x = layers.LeakyReLU(alpha=leaky_relu_alpha, name='regr2_leaky_relu_1')(regr2_x)
        for i, dense_units in enumerate(regression2_dense_units_list[1:]):
            regr2_x = layers.Dense(units=dense_units, name=f'regr2_dense_{i+2}')(regr2_x)
            regr2_x = layers.LeakyReLU(alpha=leaky_relu_alpha, name=f'regr2_leaky_relu_{i+2}')(regr2_x)
            if batch_norm:
                regr2_x = layers.BatchNormalization(name=f'regr2_bn_{i+2}')(regr2_x)
            regr2_x = layers.Dropout(dropout_imagpart, name=f'regr2_dropout_{i+2}')(regr2_x)
        regr2_output = layers.Dense(outputfeatures2, activation='linear', name='regr2_output')(regr2_x)
    else:
        regr2_output = layers.Dense(outputfeatures2, activation='linear', name='regr2_output')(shared_x)

    # Create the model
    model = Model(inputs=input_layer, outputs=[regr1_output, regr2_output])

    return model
```

```
num_features = 50
output_features_real = 61
output_features_imag =59
shared_dense_units_list = [336,697,870,927]
regression_dense_units_list_1 = [217,382,814,936,636]
regression_dense_units_list_2 = [975,803,324,897,463]
dropout_shared=0.002
dropout_realpart=0.002
dropout_imagpart=0.002
batch_norm=True

model = multi_task_model(num_features,
                         output_features_real,
                         output_features_imag,
                         shared_dense_units_list,
                         regression_dense_units_list_1,
                         regression_dense_units_list_2,
                         leaky_relu_alpha=0.2,
                         batch_norm=True)


initial_lr = 0.01
# Apply the scheduler to an optimizer
optimizer = tf.keras.optimizers.Adam(learning_rate=initial_lr)
```

```
# Huber Loss with a delta of 1.0
huber_loss = Huber(delta=1.0)
```

```
def scheduler(epoch, lr):
    return lr * 0.99

lr_scheduler = tf.keras.callbacks.LearningRateScheduler(scheduler)

early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=100, restore_best_weights=True)
```

```
model.compile(
    optimizer=optimizer,
    loss={
        'regr1_output': losses.Huber(delta=1.0),
        'regr2_output': losses.Huber(delta=1.0)
    },
    loss_weights={
        'regr1_output': 1.0,  # the weights can get adjusted depending on the importance of each task
        'regr2_output': 1.0
    },
    metrics={
        'regr1_output': [metrics.MeanAbsoluteError()],
        'regr2_output': [metrics.MeanAbsoluteError()]
    }
)
```

```
# Train the model
history = model.fit(X_train_noisy_random,
                    {'regr1_output': train_real_part, 'regr2_output': train_imaginary_part},
                    validation_data=(X_test_random, {'regr1_output': test_real_part, 'regr2_output': test_imaginary_part}),
                    epochs=500,
                    batch_size=512,
                    callbacks=[lr_scheduler, , early_stopping])
```

Appendix D: Python Codes Noise Augmentation

```
def add_white_noise(x, level=0.05, by='relative', seed=None):
    """
    Add white Gaussian noise to x.

    Parameters
    ----------
    x : np.ndarray
        Input array (any shape).
    level : float
        • by='relative': fraction of x.std()  (e.g. 0.05 = 5 %)
        • by='fixed'   : σ itself
        • by='snr'     : desired SNR in dB
    by : {'relative', 'fixed', 'snr'}
    seed : int or None
    """
    rng = np.random.default_rng(seed)
    if by == 'relative':
        sigma = level * x.std()
    elif by == 'fixed':
        sigma = level
    elif by == 'snr':
        sigma = x.std() / (10**(level / 20))
    else:
        raise ValueError("by must be 'relative', 'fixed', or 'snr'.")
    return x + rng.normal(0.0, sigma, size=x.shape)


sensor_data_modif_noisy_random = add_white_noise(sensor_data_modif_random, 0.05, 'relative',seed=42)
```

Appendix E: Python Scripts for BILSTM Network

```
import tensorflow as tf
from tensorflow.keras import layers, Model

# ----------------------------
# Build the BiLSTM model
# ----------------------------
n_features = 50

inputs = layers.Input(shape=(None, n_features), name="input_sequence")

x = layers.Bidirectional(
    layers.LSTM(64, return_sequences=False),
    name="bilstm"
)(inputs)

x = layers.BatchNormalization(name="bn_after_lstm")(x)
x = layers.Dropout(0.02, name="dropout_after_lstm")(x)

# Dense layer without activation
x = layers.Dense(50, name="dense_projection")(x)

# LeakyReLU activation
outputs = layers.LeakyReLU(alpha=0.2, name="leakyrelu_output")(x)

BILSTM = Model(inputs=inputs, outputs=outputs, name="BiLSTM_Model")

# ----------------------------
# Compile
# ----------------------------
initial_lr = 0.01
optimizer = tf.keras.optimizers.Adam(learning_rate=initial_lr)

BILSTM.compile(
    optimizer=optimizer,
    loss="mse",
    metrics=[tf.keras.metrics.MeanAbsoluteError(name="mae")]
)

# ----------------------------
# Learning rate scheduler
# ----------------------------
def scheduler(epoch, lr):
    return lr * 0.99

lr_scheduler = tf.keras.callbacks.LearningRateScheduler(scheduler, verbose=1)

early_stopping = tf.keras.callbacks.EarlyStopping(
    monitor="val_loss",
    patience=20,
    restore_best_weights=True
)

# ----------------------------
# Train
# ----------------------------
with tf.device("/GPU:0"):   # ensures GPU usage if available
    history = BILSTM.fit(
        X_train_lstm_square_noisy_5, y_train_lstm_square,
        validation_data=(X_test_lstm_square, y_test_lstm_square),
        epochs=300,
        batch_size=256,
        verbose=1,
        shuffle=True,
        callbacks=[lr_scheduler, early_stopping]
    )
```