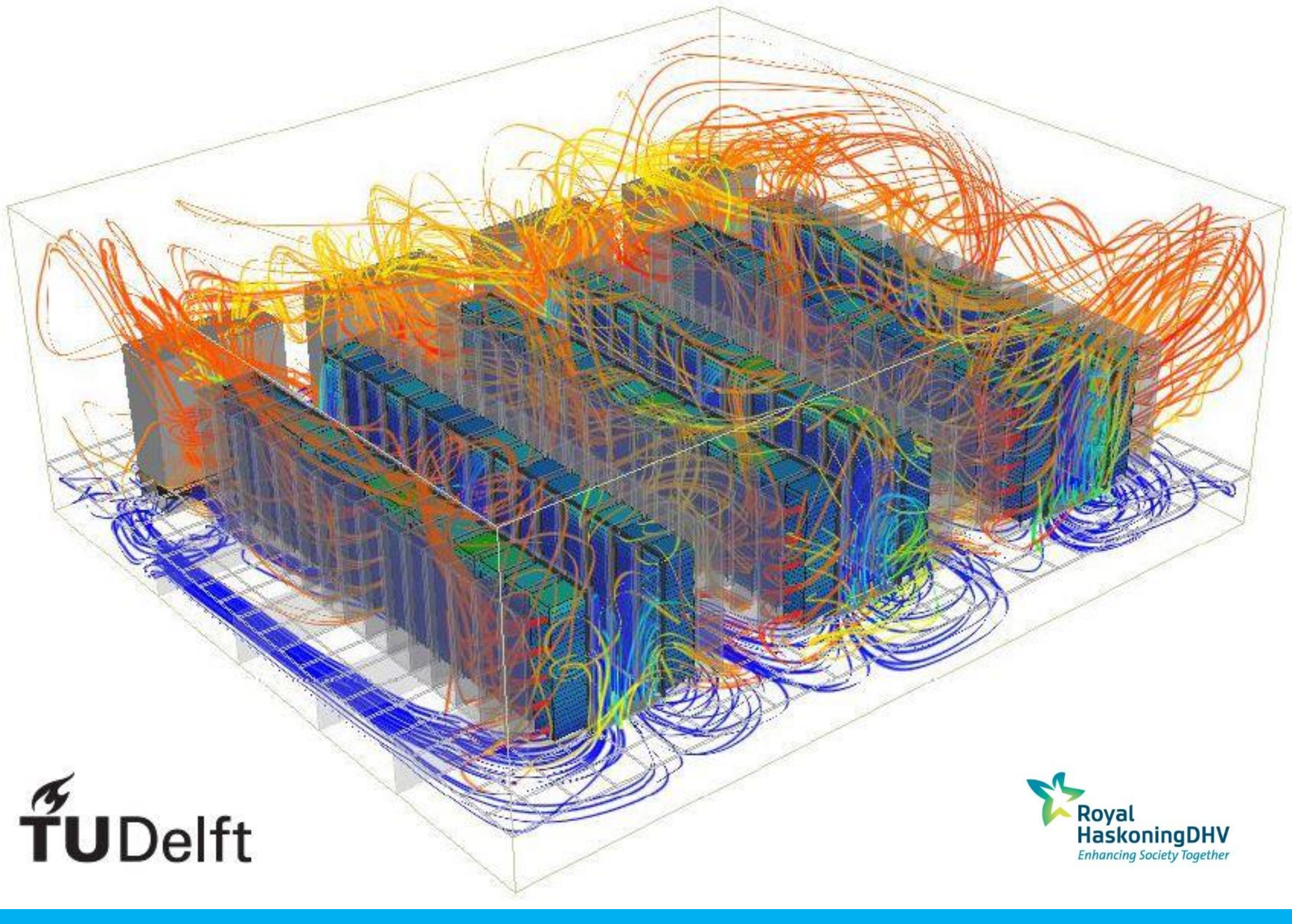
Thermal Management of a Data Center White Space

A numerical study using computational fluid dynamics, flow & temperature predictions using neural networks, and white space design optimization using a genetic algorithm.



Thermal Management of a Data Center White Space

by

Somnath Pal

to obtain the degree of Master of Science

at the Delft University of Technology,

to be defended publicly on Wednesday April 29, 2020 at 14:30.

Student number:	4724321	
Project duration:	June 12, 2019 – March 31, 2020	
Thesis committee:	Prof. Dr. Ir. J. Westerweel,	TU Delft, chairman
	Dr. Ir. M.J.B.M Pourquie,	TU Delft, supervisor
	Ir. S.H.J.J Smit,	TU Delft, daily supervisor
	PDEng. Ir. Argyrios Papadopoulos,	Royal HaskoningDHV, daily supervisor
	Dr. Ir. M. Langelaar,	TU Delft

”We have the responsibility as Engineers to do the best we can with what we have.”

– Prof. William K. George

Abstract

The total electrical energy consumption by all the operational data centers (located all over the world) is enormous (approx. 1% of the global electricity demand $[20000 TWh]$[55]). This electrical energy is required $24x7$ to operate and cool all the IT equipment present in the data center. The electrical energy required to cool all the servers present in the white space can range from as low as 10% to as high as 40%[19] of the total data center electrical consumption. The server's inlet temperature has to be within the ASHRAE recommended range ($18°C$ - $27°C$), so that they can function correctly.

A simplified design of a raised floor white space with hot aisle / cold aisle configuration is considered. The tile flow rate through the floor tiles influences the server's inlet temperature. To control the tile flow rate, 11 design variables of the data center white space are identified. These are the position of the 4 perforated plates, the amount of perforations of each perforated plates, the floor tiles perforation, the raised floor height, and the CRAH distance to the cabinet. 600 design samples of the white space are generated by applying the Latin Hypercube Sampling (LHS) technique on these 11 design variables. A well-validated CFD software 6SigmaRoom by Future Facilities is used to generate the CFD results. The standard $k-\epsilon$ model is used to model the turbulence in the CFD simulations, in a steady-state condition. A database of 600 samples is generated by recording the cabinet's inlet temperature, flow rate of floor tiles from the CFD simulations, and the corresponding changeable design parameters generated by the LHS technique. The error due to CFD simulation is estimated at less than 4% for the tile flow rate and $1.7°C$ for the server inlet temperature[42].

Four Artificial Neural Networks (ANN) are trained on the data from the database to predict the floor tile's tile flow rate and the cabinet's inlet temperature, respectively. The average R^2 prediction (testing) accuracy is 0.97 for the tile flow rate predictions and 0.92 for the cabinet's inlet temperature predictions. Their average prediction error is less than 5% for the tile flow rate and less than $2°C$ for the cabinet's inlet temperature. The Non-dominated Sorting Genetic Algorithm-II (NSGA-II), a variant of the genetic algorithm, is used to find the optimum values of the 11 design parameters of the white space. These optimum values are going to ensure the server's inlet temperature to be within the ASHRAE recommended range. The genetic algorithm optimizes the design variables based on the predictions made by the neural network predicting the tile flow rate. The values of the optimized design parameters are verified using the 6SigmaRoom software by comparing the server's (mean) inlet temperature of the optimized case with the non-optimized case. More number of servers have their (mean) inlet temperature below $27°C$ in the optimized case as compared to the non-optimized case. The electrical power required by the CRAHs to cool the white space is reduced by 10% in the optimized case as compared to the power which is required by the CRAHs in the non-optimized case. In this study, a CFD simulation of the white space took 40 minutes. The neural networks took less than a minute to make the predictions, and the NSGA-II algorithm took less than 10 minutes to find the optimized design parameters of the white space.

Thus, in this thesis, it is shown that using an artificial neural network and a genetic algorithm, in combination with computational fluid dynamics gives satisfying results in optimizing the white space design, required to keep the server's inlet temperature within the ASHRAE recommended range. The computational time required to find the optimum white space design is also reduced by using a neural network and a genetic algorithm. The prediction by the neural network and the optimization performed by the genetic algorithm can be improved further with the availability of more training data and in-depth knowledge of applying these techniques (the neural network and the genetic algorithm) in predicting and optimizing the solutions respectively.

Acknowledgement

Finally the day has come to graduate as an Ingenieur from Technische Universitiet Delft. This would not have been possible without the constant love and support from my parents. Thank you Mummy and Babi!. I dedicate this thesis to both of you.

I would like to thank Royal HaskoningDHV for providing me the opportunity to work on this topic for my masters' thesis. This project has taught me quite a lot about data centers and has also offered me a chance to apply my academic knowledge. I want to thank my company supervisor PDEng. Ir. Argyrios Papadopoulos for guiding me throughout this thesis and the Mission Critical Team at RHDHV for providing me a pleasant atmosphere to conduct my research study. I want to thank my ex-colleague at RHDHV, Mr. Kishan Basropansingh, for the support he provided during my thesis. I would also like to thank the support team from Future Facilities for delivering me the technical support to obtain the CFD simulations.

I would like to thank my university supervisors, Dr. Ir. Mathieu Pourquie and Ir. Stephan Smit, for providing me the academic guidance to research on this topic, which was new to all of us, and for motivating me at times when I lost momentum.

Mr. Evert Vixseboxse, thank you for listening to my troubles faced during my masters and supporting me to continue forward with my masters smoothly.

I would like to thank my friend Darshan and the mentors from Rotterdam.AI community, for also guiding me in implementing the neural network concepts for my thesis.

The journey through my masters would not have been possible without support from my undergrad professors Dr. Sujit Nath and Dr. Dipankar Bhanja. My journey as a researcher started under your supervision. Thank you for providing me all the guidance, support, and motivating me during my difficult times.

Also, this journey would not have been possible without the constant support and trust from my friends. Thank you, Grevil, Darshan, Anindya, Manvi, Sagar, and Manvika, for being there for me and being always available when I needed your help.

To all the other people who I could not mention here, I would also like to thank them for their love and support. Last but not least, I would like to thank the Delft University of Technology for providing me an opportunity to attain my Masters.

Somnath Pal
Delft, April 2020

Contents

Nomenclature

Acronyms

ANN	Artificial Neural Network
ASHRAE	American Society of Heating, Refrigeration and Air-Conditioning Engineers
CFD	Computational Fluid Dynamics
DNN	Dense Neural Network
etc	Et cetera
FDM	Finite Difference Method
FEM	Finite Element Method
FVM	Finite Volume Method
GA	Genetic Algorithm
HVAC	Heating Ventilation and Air-Conditioning
ICT	Information & Communication Technology
IT	Information Technology
MSE	Mean Squared Error
POD	Proper Orthogonal Decomposition
RSM	Response Surface Methodology

Greek Symbols

α	Thermal Conductivity
β	Thermal Expansion Coefficient
ϵ	Dissipation of Turbulent Kinetic Energy
κ	Von-Karman's Constant
μ	Laminar Viscosity
μ_{eff}	Effective Viscosity
μ_t	Turbulent Viscosity
ν	Kinematic Viscosity
ρ	Density
$\sigma_k, \sigma_\epsilon$	Effective Prandtl Number of k_e and ϵ_p
τ_w	Wall Shear Stress
τ_{ij}	Stress Tensor

Roman Symbols

$\overline{u}$	Average Component of Velocity
B	Constants
b	Bias Term
C	Celsius
c	Speed of Light in Vacuum
C_H	Height of the Cabinet
C_P	Specific Heat at Constant Pressure
C_W	Width of the Cabinet
CR_D	Depth of the CRAC
CR_H	Height of the CRAC
CR_W	Width of the Cabinet
CR_D	Depth of the CRAC
D	Perpendicular Distance of the CRAC form the first Cabinet of the Row
F_i	Body Force Term
Gr	Grasshof Number
h	Height of the Under-Plenum
K	Effective Thermal Conductivity
k	Turbulent Kinetic Energy
kW	Kilo-Watts
L	Characteristic Length Scale
m	Metre
P	Static Pressure
$P1$	Perforated Rectangular Plate 1
$P2$	Perforated Rectangular Plate 2
$P3$	Perforated Rectangular Plate 3
$P4$	Perforated Rectangular Plate 4
P_H	Height of the Perforated Plate
P_L	Length of the Perforated Plate
Pr	Prandtl Number
Ra	Rayleigh Number

Re	Reynolds Number
s	Second
S_D	Depth of the Server
S_H	Height of the Server
S_h	Energy Source per unit Volume
S_W	Width of the Server
T	Temperature
T_H	Height of the Square Tile
T_W	Width of the Square Tile
U	Characteristics Velocity Scale
u'	Fluctuating Component of Velocity
u_τ	Wall friction Velocity
u_i	Instantaneous Velocity Component in Cartesian direction
w	Weight of a Neuron
x	Input to a Neuron
y	Output of a Neuron
TWh	Tera Watt Hours

Subscripts

D	Depth
e	Energy
eff	Effective
H	Height
i	Cartesian direction (x, y, z)
j	Cartesian direction (x, y, z)
L	Length
l, k	Neuron number in a given Layer
P	Pressure
ref	Reference
t	Turbulent
W	Width
w	Wall

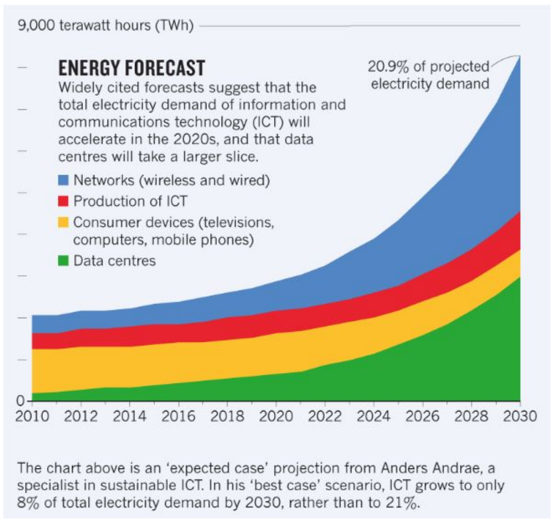
1 Introduction

1.1. Background

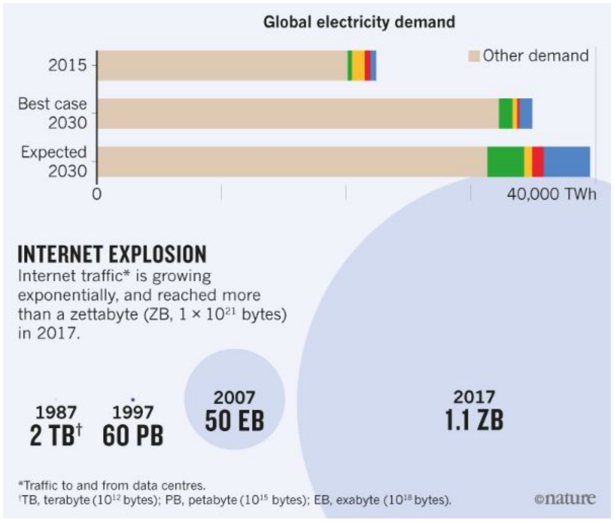
Data Centers form a vital part of our society and plays an essential role in the digital economy. With the increased use of the internet and the advancement of electronics hardware technologies, a large amount of data gets stored in data centers, accessed and exchanged from them daily. Data centers form the backbone of the internet. The size of a data center can be as small as a room containing 10 to 100 racks of servers[35] to as large as a massive building comparable to the size of aircraft carriers[55]. Generally, a data center consists of numerous servers, stacked upon each other, spanning for rows down the windowless halls[55].

As more people start using the internet for accessing information, sending emails, social networking, video and voice calls, cloud computing services, scientific computing, the demand for data centers increases. New data centers have to be constructed, or existing data centers need to be scaled up to meet this demand. This is going to lead to more consumption of electrical energy by the data centers. Data centers might receive its energy from renewable or non-renewable sources. In figure 1.1a, it is seen that the ICT and data center electricity demand is going to increase in the future. Figure 1.1b shows the increased usage of the internet over the years which has lead to the increased access of data from data centers. The increased use of IT and ICT equipment in data centers has led to more usage of electricity by data centers.

It is estimated that data centers consume around 200 terawatt-hours (TWh) of energy each year. It is more than the national energy consumption of certain countries like Iran, approximately half of the electricity consumed in transportation all over the world, and 1% of the global electricity demand which is around $20000 TWh$[55]. Data Centers[55] contributes 0.3% of the overall carbon emissions. It is estimated that the energy usage for cooling IT equipment in the data center can go from as low as 10% to as high as 40% of the total energy consumed by the data center[19]. The cooling energy required depends on the climatic conditions of the region in which the data center is built, the cooling systems implemented, the cooling control strategies used, and the utilization rate of the servers in the data center.



(a) Forecast of the energy consumption(TWh)



(b) Global electricity demand and the internet explosion

Figure 1.1: (a) The forecast of energy consumption (TWh) by networks, ICT, data center and consumer devices against time(year).(b)The global electricity demand by the networks, ICT, data center, and consumer devices and the growth of the internet traffic due to its increased usage[41]. The graphs forecast more energy usage in data centers as the digital economy booms.

Thus, the data centers must operate optimally by consuming energy efficiently. One of the main components of a data center are the cooling units and the cabinets containing servers. The server dissipates heat, which needs to be removed and the inlet temperature of the servers have to be maintained within the recommended temperature (dry bulb temperature) range ($18°C - 27°C$) according to American Society of Heating, Refrigeration and Air-Conditioning Engineers (ASHRAE)[2]. In figure 1.2, is a plot between the Information Technology (IT) inlet relative humidity on the y-axis and IT inlet temperature (in $°C$) on the x-axis. The green zone highlights the recommended temperature range, and the blue zone highlights the acceptable temperature (dry bulb temperature) range ($15°C - 32°C$), which is the extreme limit for the inlet temperature of the servers. According to industry-standard, the server's inlet temperature cannot exceed the allowable range (in the worst-case scenario). But it is a choice of the client, who is renting the data center spaces to decide in which range the server's inlet temperature should lie in. Mainly, the extreme inlet temperature of servers in data centers should fall within the A1 class.

If the inlet temperature of the server exceeds the allowable range, the servers are going to be overheated, and causing it to malfunction. This will lead to a shutdown of the IT services and loss of revenue for the company providing these services. A data center needs to be sufficiently cooled but not over-cooled, as it can lead to operational financial loss due to the cost associated with over-cooling. The data center white space[1] can be cooled either by air-based cooling or by liquid-based cooling[4]. In air-based cooling, the servers in the white space are cooled using cold air delivered by the air cooling units. In liquid-based cooling, the servers are cooled by using cold water passing through the hot sides of the servers or by immersing the servers in a dielectric material. The flow field determines the cooling in an air-cooled data center, and Computational Fluid Dynamics (CFD) provides a suitable tool to determine these flow fields,

[1]The data center has an area designated as white space and another area as grey space. In a white space, the IT equipments are placed. It consists of servers, storage, network gear, racks, air conditioning units, and power distribution systems. Grey space is the area where the back-end equipment like the switch gear, UPS, transformers, chillers and generators are located.

temperature distributions[64] and is used to study the cooling problem in a data center.

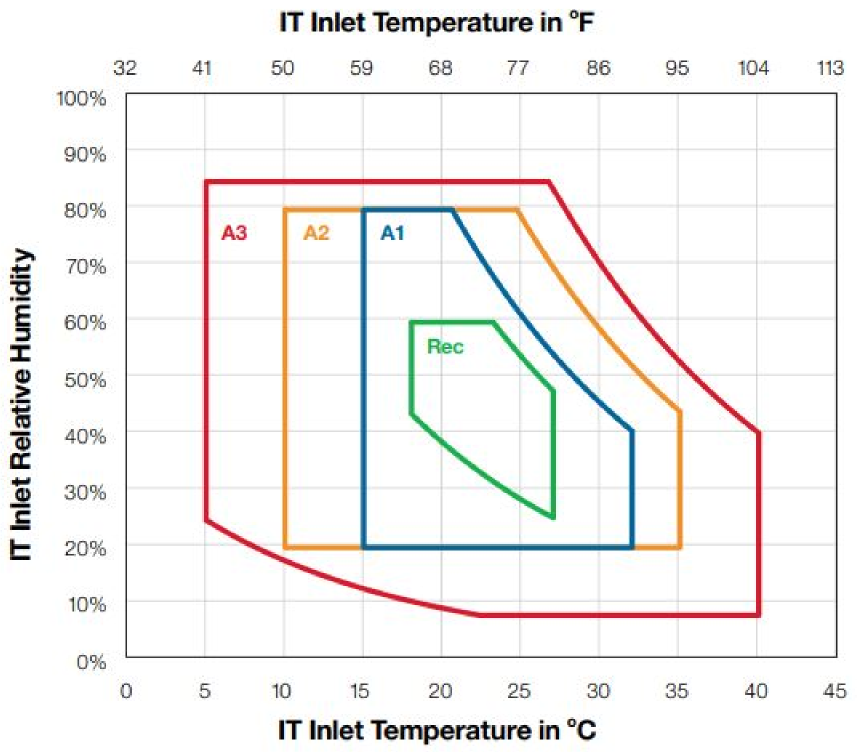


Figure 1.2: Recommended and allowable operating temperature range according to ASHRAE classes A1/A2/A3 for IT product operation. Inlet temperature of the servers are mainly kept within the recommended (Green Zone) range. The temperature range of the other Information Technology (IT) products are chosen among the three (A1/A2/A3) classes based on client operational requirement[3].

1.2. Purpose/Goal of the Thesis

The purpose/goal of the present study is to find out the set of optimized design parameters of the data center white space that will influence the temperature distribution and flow fields in the white space. A new cooling concept is going to be tested. This goal is going to be achieved by creating a database of CFD simulations, predictions of the flow/temperature field using a neural network, and optimization using a genetic algorithm. The servers are assigned to be running at full heat load capacity to simplify the constraints of the CFD model. In reality, each server in a data center has a different heat dissipation rate based on its usage. Another goal would be to reduce the power consumption of the cooling units, which are used to cool the servers, and to reduce the computational time required to simulated the temperature and flow fields by using a neural network and a genetic algorithm.

1.3. Thesis Outline

General structure of the thesis is presented as follows.

- Chapter 1- Consists of the background, and the purpose/goal of the study done in this thesis.
- Chapter 2- Literature review of the data center white space design. It also contains a review of the research done using CFD to analyze the thermal and flow field in a data center. The use of a neural network for thermal and flow prediction and the genetic

algorithm as an optimization tool in heat transfer application is also presented.

- Chapter 3- It contains the theory of the governing equation used in the CFD simulations, and parameters required to design a neural network and a genetic algorithm.
- Chapter 4- Describes the methodology followed to generate the results from the CFD simulation, Artificial Neural Network (ANN), and Genetic Algorithm (GA).
- Chapter 5- It contains the results of the CFD simulations, the ANN, and the GA, respectively.
- Chapter 6- It provides a conclusion for the thesis and the recommendations for future work.

2
Literature Survey

2.1. Data Center Cooling Infrastructure

The main cooling infrastructures of a data center are the cooling tower, chiller(s), pumps, cooling units (CRAH/CRAC)[1], and economizer(s). This is shown in figure 2.1.

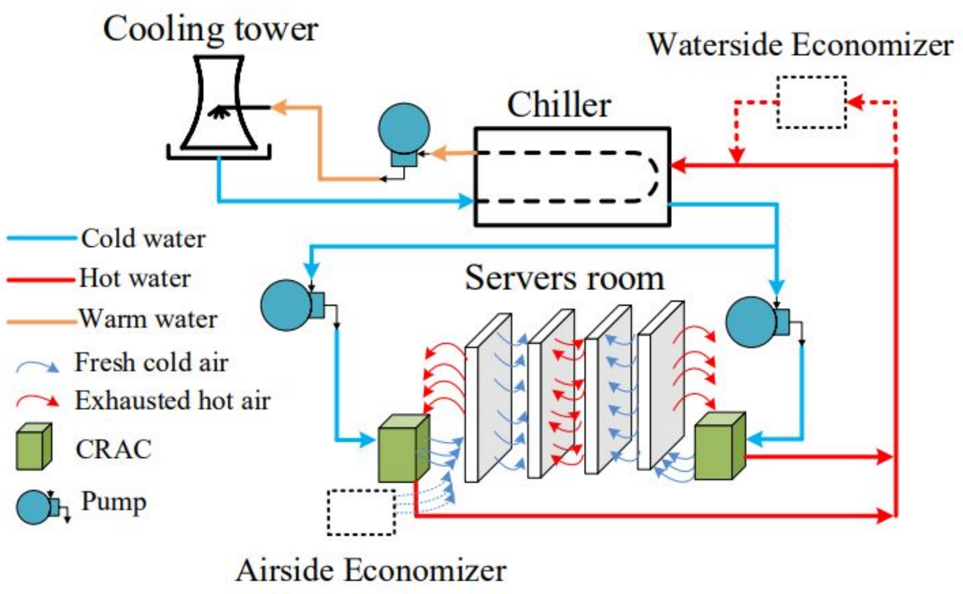


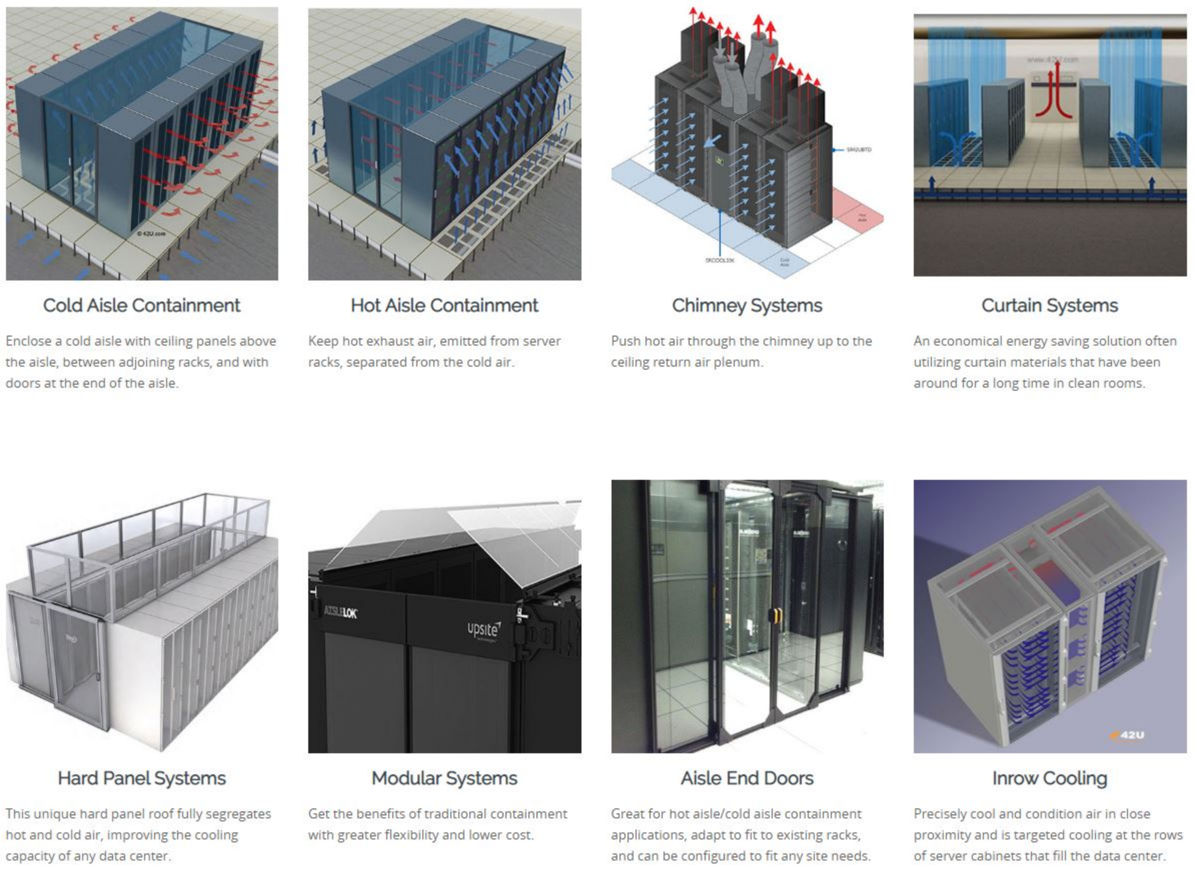
Figure 2.1: The cooling infrastructure of a data center. It generally consists of a cooling tower, chiller(s), pumps, cooling units (CRAH/CRAC), and economizer(s). The CRAC absorbs the hot air from the servers. This hot air heats the cold water passing through it. The hot water from the CRAC is supplied to the chiller to be cooled again. The chiller exchanges the heat internally with the cold water from the cooling tower. The cooling tower reduces the temperature of the incoming warm water through an evaporative process. The heat is expelled to the atmosphere. The pumps regulate the flow. The economizer reduces the energy consumption for cooling.[76].

[1]A cooling unit can either be a CRAH or a CRAC. The cooling unit consists of a cooling coil through which the coolant/refrigerant is passed. The hot air is passed around the cooling coil to cool it down to the supply temperature of the cooling unit. Computer Room Air Handler (CRAH) uses chilled water as a coolant in its cooling coil. A Computer Room Air Conditioner (CRAC) uses a refrigerant as a coolant in its cooling coil. The operational cost of a CRAC is more than a CRAH. The type of cooling unit to be used is decided by the climatic condition of the region in which the data center is built. The CRAHs are usually preferred as a cooling unit in the data centers which are built in places having a cold/temperate climatic condition, and the CRACs in the data centers which are built in hot climatic regions.

In figure 2.1, the hot air from the servers is pulled into the cooling unit (CRAH/CRAC). This air is cooled in the cooling unit and passed back to the server room. The cold water passing through the cooling units (CRAH/CRAC) is heated up from the exhausted hot air and transported to the chillers. The water cools down in the chiller and is again used to cool the server room through the cooling units. Heat is exchanged in the chiller between the hot water coming from the cooling units and the cold water coming from the cooling tower. The warm water from the chiller is cooled in the cooling tower, by letting it come in direct contact with the air. As this process happens, a small volume of warm water is evaporated. This reduces the temperature of the warm water and it is passed back to the chiller. The heat from the cooling tower is generally expelled to the atmosphere. Nowadays, due to sustainability goals, the heat from the data center can be used for district heating. One such research study has been done in the masters' thesis on "Data centers as residual heat source for district heating in residential neighborhoods of Amsterdam"[60]. Amsterdam plans to replace the gas-fired central heating system with district heating by 2050. Amsterdam's district heating goal is going to be boosted by utilizing the waste heat of the data center[17]. The pumps are used to regulate the flow and the cooling rate in a data center. An economizer is used to reduce the energy consumption for cooling. Waterside economizer can be used when the atmosphere is cold, which can reduce the burden on the chiller. The waterside economizer can be used to reduce the usage of a chiller in regions having a cold climatic condition and thus saving cooling energy. In a waterside economizer, part of the heat is collected from the hot water to heat-up the cold atmospheric air to a temperature required by the white space. Air-side economizer facilitates the fresh air to flow directly into the white space if the temperature of the atmospheric air is less than $32°C$ directly.

2.2. Data Center Containment Strategy

Figure 2.2 shows the different kinds of containment strategies commonly used in data center white space. A containment is required to separate the cold air from the hot air and prevent them from mixing up. The cooling energy consumption for a white space having a containment is less than a white space having no containment. This, in turn, saves the revenue required for cooling the white spaces. Each containment strategy has its advantages and disadvantages. The costs associated with implementing the different types of containment systems are different. The type of containment system to be implemented depends on the construction budget allocated to build it, the amount of maintenance required, safety factors like fire safety and plug leakage measures implemented in the white space, temperature environment required for personals working in the white spaces, and on the heat load of the servers in the white spaces. Typically, a hot aisle or cold aisle containment strategy is used in a data center. Thus, containment strategy helps in the reduction of energy consumption required for cooling, as 60% of the cooling energy can be wasted without containment strategies. A containment strategy also helps in the construction of a high-density data center white space in a short duration of time[16]. The cold air can flow from the cooling units into the data center white space through a raised floor configuration or a hard floor configuration. In a raised floor configuration, the cold air flows from the cooling unit into the raised floor, and it is then supplied to the white space through the perforated tiles. In a hard floor configuration, the cold air flows directly into the white space from the cooling unit. These two configurations are shown in figure 2.3. Usually, the raised floor configuration is preferred over the hard-floor configuration as it provides flexibility in rearranging the racks in the white spaces[9].



Cold Aisle Containment

Enclose a cold aisle with ceiling panels above the aisle, between adjoining racks, and with doors at the end of the aisle.

Hot Aisle Containment

Keep hot exhaust air, emitted from server racks, separated from the cold air.

Chimney Systems

Push hot air through the chimney up to the ceiling return air plenum.

Curtain Systems

An economical energy saving solution often utilizing curtain materials that have been around for a long time in clean rooms.

Hard Panel Systems

This unique hard panel roof fully segregates hot and cold air, improving the cooling capacity of any data center.

Modular Systems

Get the benefits of traditional containment with greater flexibility and lower cost.

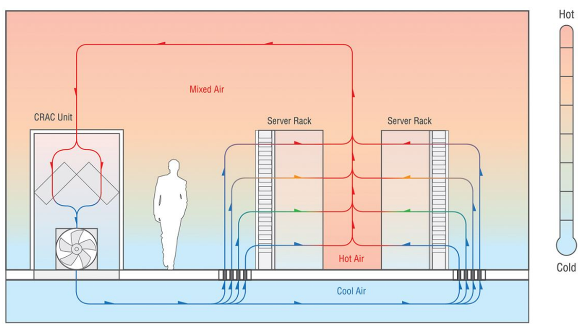
Aisle End Doors

Great for hot aisle/cold aisle containment applications, adapt to fit to existing racks, and can be configured to fit any site needs.

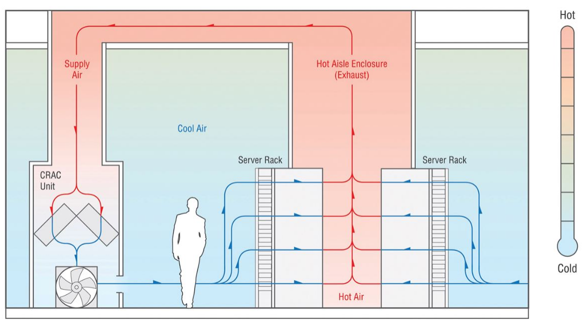
Inrow Cooling

Precisely cool and condition air in close proximity and is targeted cooling at the rows of server cabinets that fill the data center.

Figure 2.2: The different types of data center containment strategies used in white spaces. Containment is done to separate the hot air from the cold air. This, in turn, saves revenue by reducing the consumption of cooling energy. Each containment strategy has its advantages and disadvantages[15].



(a) Raised floor configuration



(b) Hard floor configuration

Figure 2.3: The raised floor configuration shown in sub-figure (a) and hard floor configuration shown in sub-figure (b) of the data center white space. In sub-figure (a), the cold air flows from the cooling unit (CRAC) into the raised floor and then into the white space through the perforated floor tiles. In sub-figure (b), the cold air flows directly into the white space from the cooling unit. A hot aisle containment system will be required in a hard floor configuration, to prevent the mixing of hot and cold air[16].

2.3. Data Center CFD

Numerous Computational Fluid Dynamics studies have been done to analyze the thermal and flow field in an air-cooled data center. Some of the studies are presented as follows. Patankar[64] presents a holistic overview of the airflow distribution and cooling in a data center. Wibron et al.[80] have used Ansys CFX 16.0 to validate different turbulence models against experiments performed in measuring the velocity fields and temperature in a data center. CFD simulation of raised floor data center configuration has been performed in the paper "Experimentally Validated Computational Fluid Dynamics Model for Data Center With Active Tiles" by Athavale et al.[42]. In this paper, a CFD simulation of a raised floor data center configurations is performed, employing active tiles and validated it against experiments done in the data center lab at Georgia Institute of Technology.

2.4. Raised-Floor Data Center Design

2.4.1. General Arrangement

A typical cooling distribution working principle for data center white space with a raised floor is shown in figure 2.4. The CRAC is placed on the raised floor. The cold air moves downward from the CRAC[2] into the raised floor and then through perforated tiles into the inlet of the server racks. Heat exhausted from the rear of the server racks is drawn into the top face of the cooling unit (CRAC)[64].

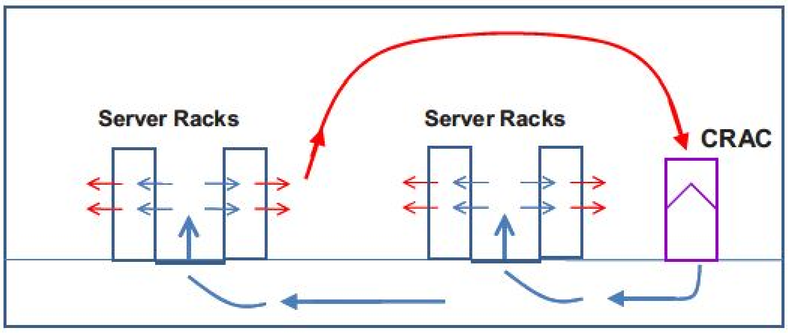


Figure 2.4: Data center with a raised floor configuration. Cold air from the CRAC is supplied through the raised floor into the server racks. Hot air from the server racks moves toward the ceiling (due to lighter density) and is then sucked into the CRAC[64].

2.4.2. Cold Aisle / Hot Aisle Arrangement

To reduce the possible mixing of exhausted hot air from the servers with the supplied cold air, the racks of servers are arranged in the "Hot Aisle / Cold Aisle" configuration, as shown in figure 2.5. This was suggested by Sullivan[11] and has now become a standard practice in data centers. Perforated tiles are placed in the cold aisle. On both sides of the cold aisle, it is surrounded by the inlet face of the server racks. There are no perforated tiles placed in the hot aisle in order to prevent the mixing of hot and cold air. The hot air from the server racks emerges in the hot aisle, which then rises towards the ceiling and returns to the inlet of the cooling unit(s). A certain amount of leakage of hot air happens through the sides of the cabinet (containing the servers), which is 5% of the blanked frontal surface of the cabinet[3].

[2]This is a down-flow CRAC unit. There can also be an up-flow CRAC unit, ceiling unit, and other cooling systems that regulate the airflow in the data center.
[3]Future Facilities have provided this information, the company developing the 6SigmaRoom CFD software.

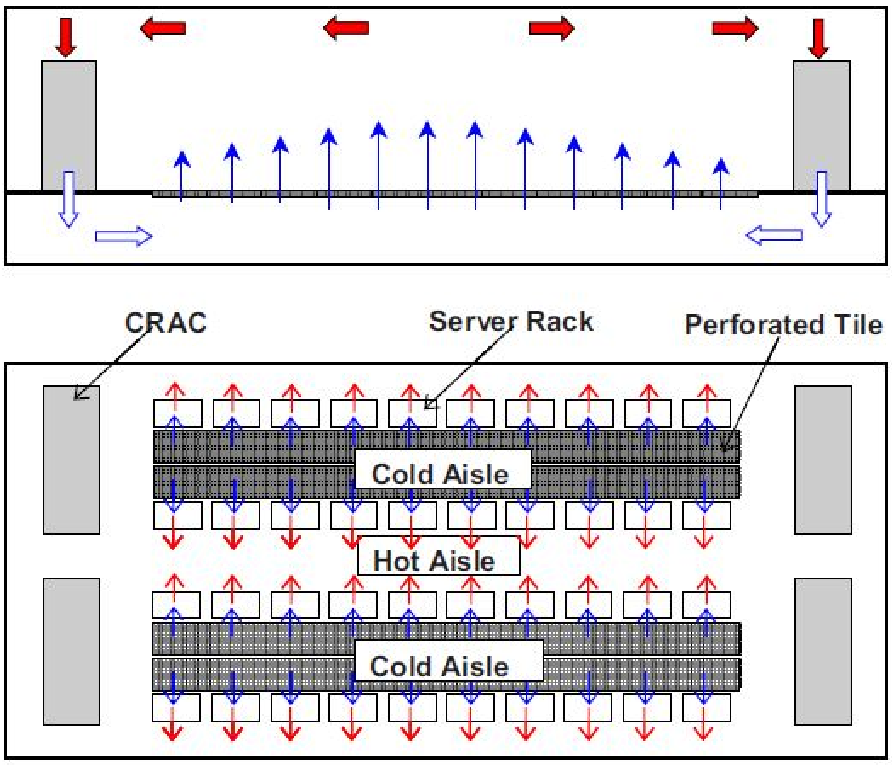


Figure 2.5: The hot aisle / cold aisle arrangement of server racks. The cold aisle supplies the cold air to the server's inlet. The hot aisle collects the hot air from the servers and returns it to the cooling units (CRACs). This kind of aisle arrangement is done to prevent the mixing of hot and cold air[64].

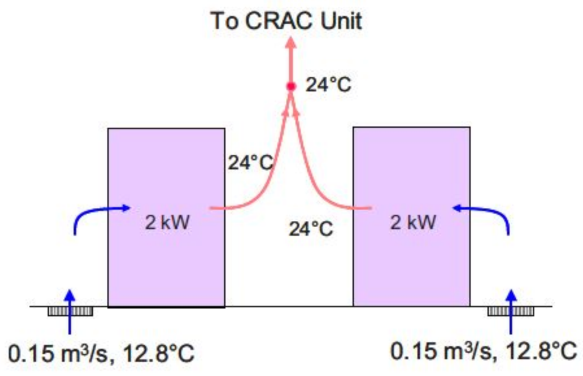
2.4.3. The Airflow Requirement by the Servers in a Cabinet

A cabinet contains multiple servers stacked up upon each other. Each server has an internal fan that draws in the cold air from the cold aisle. To maintain the servers within the recommended temperature range and to prevent overheating, the server fans change the airflow rate required by it.

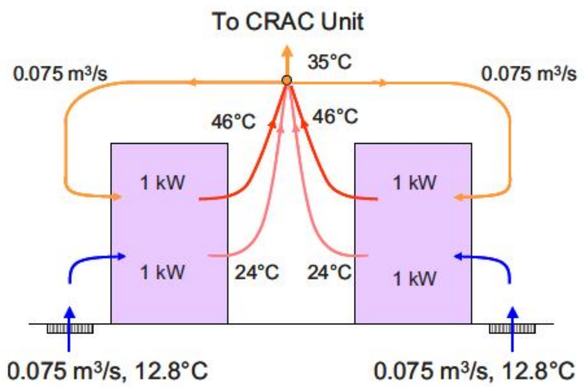
To guarantee that the supplied cold air enters the cabinet (containing the servers), the airflow rate of the tile present at the base of the cabinet has to be greater than or equal to the airflow rate required by all the servers in the cabinet. This is illustrated in figure 2.6a and figure 2.6b. In this example, in each case the cabinets dissipate heat of $2kW$, and it requires an airflow rate of $0.15m^3/s$, i.e., each floor tile should be able to provide this airflow rate to the cabinet.

In figure 2.6a, it is seen that the corresponding tile is meeting the airflow required by the cabinet. Thus there is no recirculation of hot air (from the hot aisle) as the servers are adequately cooled. In figure 2.6b, it is observed that the tile is incapable of providing the required airflow. The lower half of the cabinet is adequately cooled but in the upper half, mixing of hot air $(46°C)$ and cold air $(24°C)$ occurs before entering the inlet of the cabinet. Thus the inlet temperature of the supplied air in the upper half of the cabinet increases, and the servers start to overheat.

The effect of insufficient airflow through the floor tiles is shown in the CFD study of a data center white space in the paper "Numerical study of thermal management of data center using porous medium approach" by Saha et al.[69] (see figure 2.7). In this figure, the temperature plot has been taken at four vertical locations from the floor. It starts from the bottom of the cabinet (as shown in figure 2.7(d)) to the top of the cabinet (shown in figure 2.7(a)). There are recirculation effects near the corner of the last cabinet on the right, and it is more pronounced near the top of the cabinet, causing hot-spot to appear in the servers present in this region. In figure 2.8 the temperature plot has been taken in four horizontal locations, starting from the cabinet present near the CRAC, as shown in figure 2.8(a) to the cabinet farthest away from the CRAC (figure 2.8(d)). Another undesirable phenomenon is observed called the bypass or short-circuiting of cold air. In this phenomenon, excess cold air is produced (more than what is required by the servers), which flows directly into the exit vents present in the white space. This leads to a wastage of energy required to produce the excess cold air and a decrease in the performance of the cooling units (CRAH/CRAC).



(a) Sufficient airflow being supplied



(b) Insufficient airflow being supplied

Figure 2.6: Comparison of cases with (a) sufficient cold airflow supplied from the raised floor to the cabinets, and (b) insufficient airflow supplied from the raised floor to the cabinets. Scenario (b) leads to recirculation of hot air into the top of the cabinets and rise in the inlet temperature at the top of the cabinet[64].

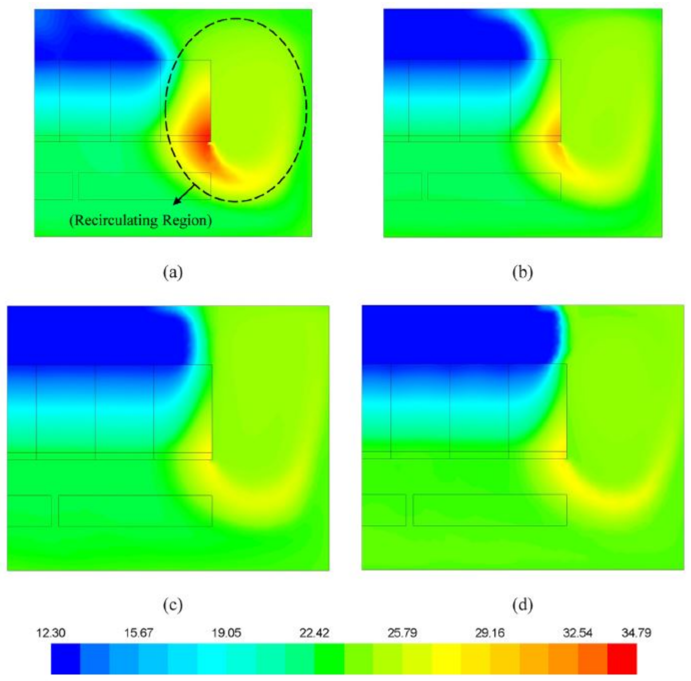


Figure 2.7: The contours of temperature ($^{\circ}C$) at different horizontal cross-sections showing the extend of recirculation of hot air into the inlet of the cabinet containing the servers. This leads to a rise in the inlet temperature in the recirculation region[69]. The floor tiles are providing the cold air to the inlet of the cabinet. Cabinets are modeled as a black box or lumped model approach.

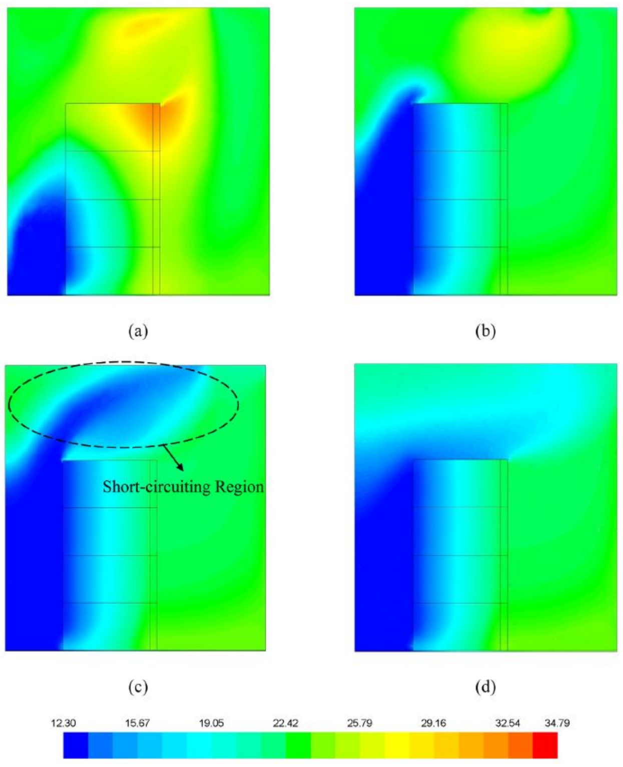


Figure 2.8: Contours of temperature ($^\circ C$) at different vertical cross-sections showing the extend of bypass of the excess cold air to the outlet vents, thus leading to inefficient cooling and wastage of cooling energy[69]. The floor tiles are providing the cold air from the raised floor to the inlet of the cabinet. The cabinets are modeled as a black box or lumped model approach.

2.4.4. The Distribution of Airflow through the Perforated Tiles

In figure 2.9, the tiles are positioned at different distances from the CRAC. The tile farthest away from the CRAC gets the maximum airflow through it, and the one near it gets the minimum airflow[64].

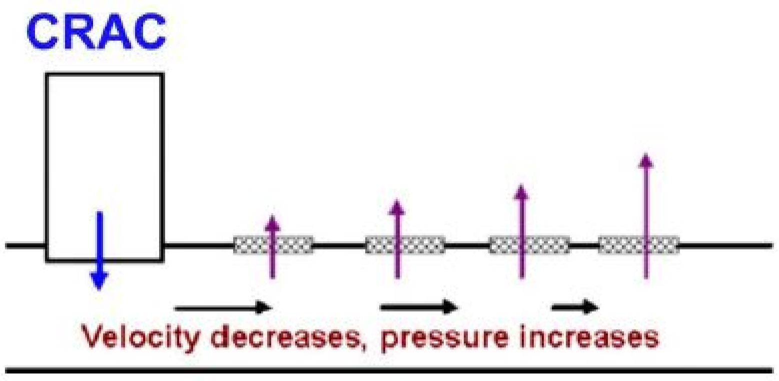


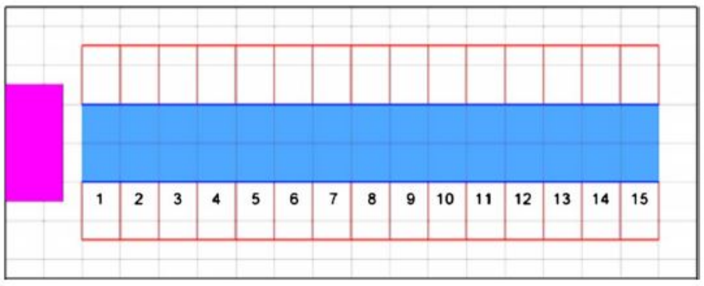
Figure 2.9: The uneven distribution of airflow through the floor tiles, due to the unequal velocity distribution in the raised floor. The velocity decreases away from the CRAC, as the cold air escapes through the floor tiles. According to Bernoulli's equation, the pressure increases away from the CRAC. The pressure above the raised floor is nearly uniform. Thus, the tile away from the CRAC has maximum airflow[64].

The reason for this maldistribution is explained as follows. In figure 2.9, if we consider the velocity in the raised floor in the horizontal direction. It decreases since the air escapes from the perforated tiles[64]. Accordingly to Bernoulli's equation, it implies as we move away from the CRAC, the pressure should increase in the horizontal direction. The pressure above the floor is nearly uniform. The pressure drop across the floor tile determines the airflow rate through it. The tiles present towards the right (away from the CRAC) experience a more significant pressure drop and thus delivers the highest airflow rate[64]. This is shown in figure 2.10. The region in the raised floor near the CRAC has a negative pressure region. This can be explained with the concept of venturi effect[38]. The presence of the perforated tiles on the raised floor, near the CRAC, acts like a constriction. This speeds up the airflow through the tiles causing a negative pressure region to be formed under the floor. This effect can only be seen near the CRAC as the velocity exiting the CRAC is very high as compared to the velocity of the airflow in the other regions of the raised floor.

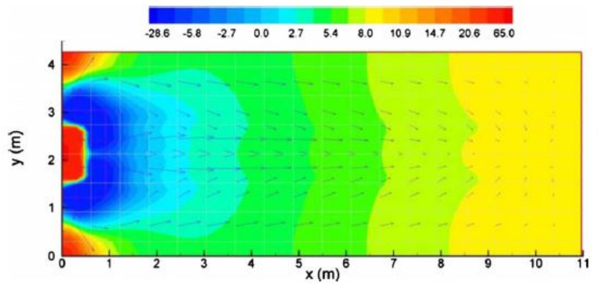
2.4.5. The Effect of Raised Floor Height and Floor Tile Perforations

As seen in figure 2.11a, the non-uniformity in the airflow is reduced on increasing the raised floor height. Above a certain raised floor height, the reduction is not significant due to the complexity of the flow and lateral spreading near the CRAC[64].

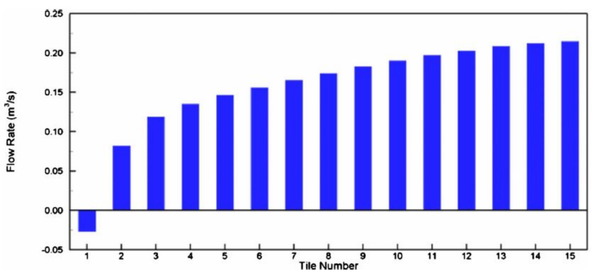
For a fixed layout and raised floor height, the non-uniformity in flow diminishes as the open area of the tile is reduced (see the paper on "Airflow and Cooling in a Data Center"[64]) as shown in figure 2.11b. It looks like the restrictive tiles are a preferred way to achieve a uniform airflow distribution. This is not the case as the flow resistance, of the flow, through the tile, becomes comparable to that of the openings around cables, pipes, and leakages (0.3% of floor area in this case). Thus the cold air would be wasted, as it would leak into the hot aisle (containing the exhausted hot air from the servers) from the raised floor, and the cold air is going to mix with it. This would then reduce the temperature of the hot air and cause the CRAC to perform at a lower efficiency.



(a) The base case consisting of the CRAC (in pink) and 15 cabinets (containing the servers) in both side of the Cold Aisle.



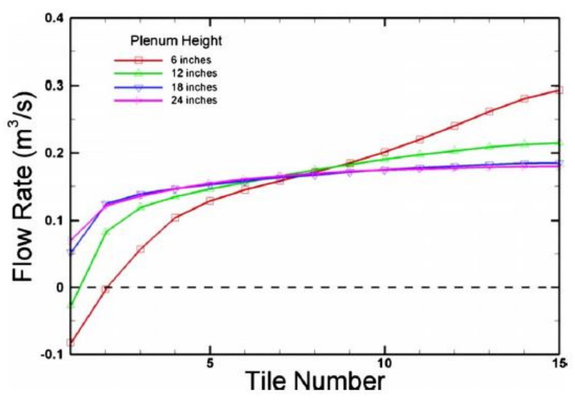
(b) The raised floor horizontal pressure distribution and velocity vectors; pressure in Pa.



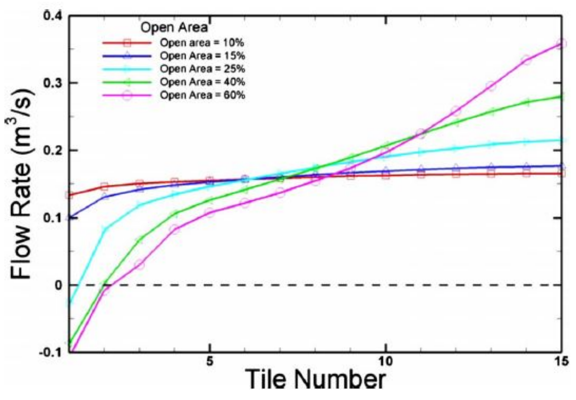
(c) The air flow rate through the perforated tiles.

Figure 2.10: The pressure distribution in the raised floor (sub-figure (b)), and tile flow rate (sub-figure (c)) for the base case (sub-figure (a)). The negative tile flow rate for the first tile is due to the negative pressure in the raised floor near the CRAC. The pressure in the raised floor increases in the direction away from the CRAC[64].

The flow resistance experienced by the CRAC at the tiles is smaller as compared to its internal flow resistances. Thus the CRAC supplies almost the same flow rate for different configuration of layout and tile perforation[64]. Thus the tiles need to have a optimum amount of perforations, and not so less that its flow resistance becomes equivalent to the resistance provided by the leakage in the raised floor.



(a) The variation of the tile flow rate with the raised floor height.

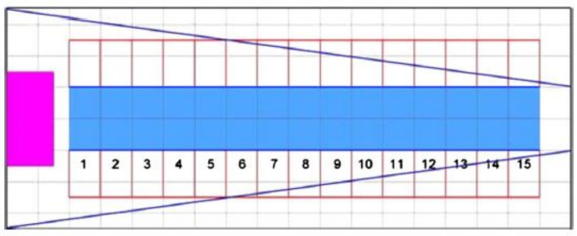


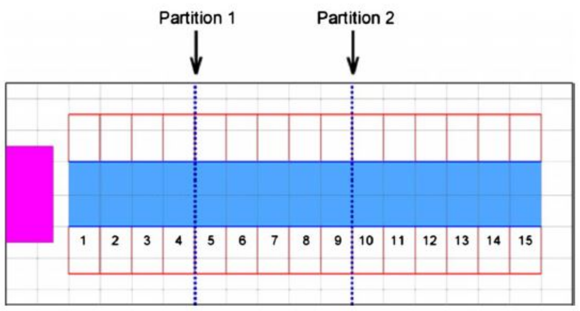
(b) The variation of the tile flow rate with the tile Perforations

Figure 2.11: The effect of the variation of raised floor height (sub-figure (a)) and the change of the floor tile perforations (sub-figure (b)) on the tile flow rate through the floor tiles[64]. The figure shows that above a certain raised floor height, the tile flow rate becomes nearly uniform. Floor tiles having less amount of perforations has a consistent tile flow rate through it.

2.4.6. Partitions in the Raised Floor

The presence of an obstruction in the raised floor can be used to regulate the airflow distribution through the floor tiles. In figure 2.12a, vertically inclined partitions are used to regulate the velocity distribution in the raised flow. As the vertical partitions act like a converging nozzle, to maintain the continuity of flow, the velocity should increase. But as some of the airflow escapes through the tiles, the horizontal velocity remains almost uniformly. Thus the pressure in the raised floor remains almost uniform.


(a) Inclined partition in the raised floor.



(b) Perforated partition in the raised floor.

Figure 2.12: The presence of inclined solid partitions (sub-figure (a)) and perforated partitions (sub-figure(b)) in the raised floor to regulate the airflow rate through the floor tiles[64]. The inclined partition acts as a converging section, which speeds up the airflow inside the raised floor and thus makes the pressure in the raised floor more or less uniform. The perforated partitions can also regulate the tileflow rate through the tiles and do not suffer from the disadvantages of solid partitions.

A comparison of the tile flow rate of a raised floor having inclined partitions with the base case without partitions is shown in figure 2.13. It is seen that the flow is almost uniform, and there is no negative tile flow rate in the raised floor having inclined partitions as compared to the raised floor having no partition. However, the presence of an inclined partition can be detrimental if there is a failure in one of the CRACs. The cold airflow from the neighboring CRAC(s) cannot be supplied to the row of cabinets provided usually by the failed CRAC, as the inclined partition does not contain perforations in it. This can be a catastrophic event as the server(s) can overheat and can shutdown.

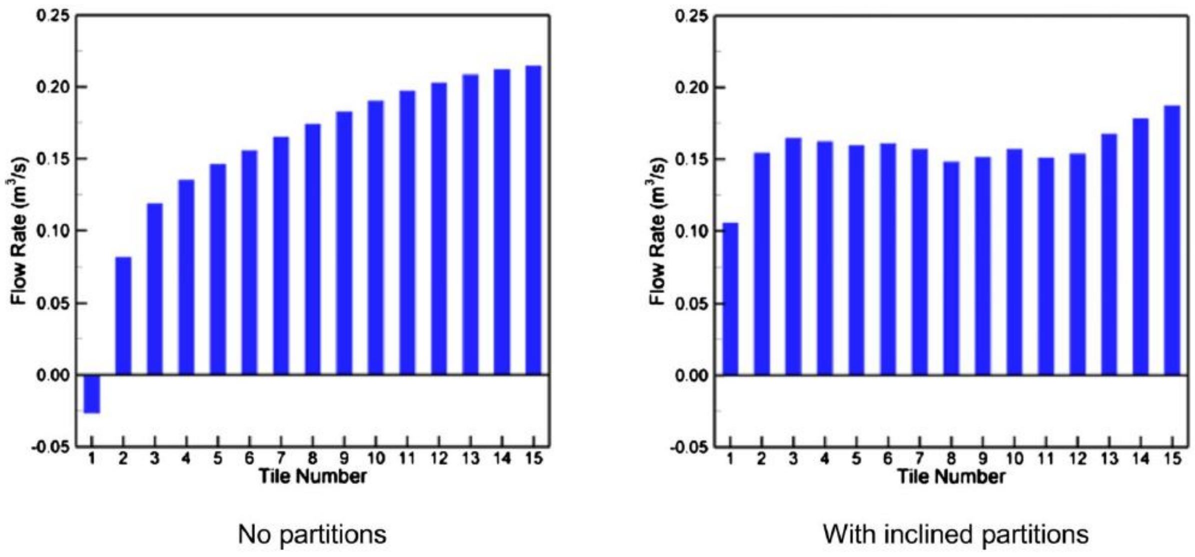


Figure 2.13: The tile flow rate without and with inclined solid partitions[64]. negative tile flow rate is seen in (sub-figure (a)) due to venturi effect near the CRAC.

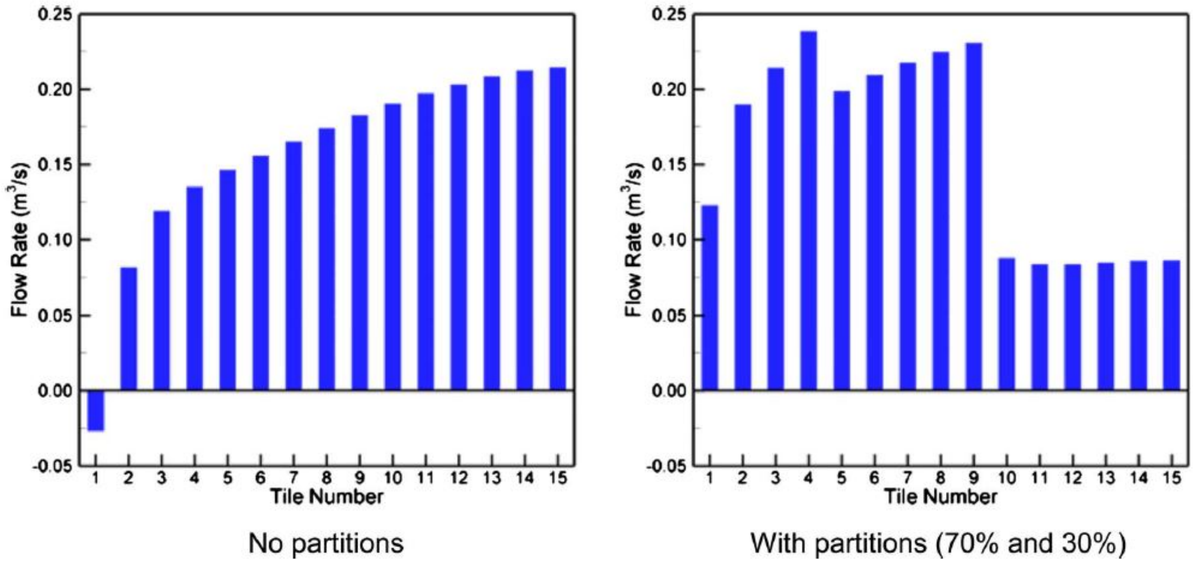


Figure 2.14: The tile flow rate without and with perforated partitions[64]. As usual, there is a negative tile flow rate in the raised floor without a partition(s) (sub-figure(a)), which is rectified by using perforated partitions in the raised floor (sub-figure(b)).

A better alternative would be to use vertical perforated plates, as shown in figure 2.12b. The failure of a CRAC wouldn't hinder the flow from the other CRAC(s) to the row of cabinets, which is normally supplied by the failed CRAC. The airflow distribution can then be controlled by determining the position of the plates and the amount of perforation, as shown in figure 2.14. Taking the inspiration of using a perforated partition(s) in the raised floor to control the air flow rate through the floor tiles, it forms the basis for the new cooling technique. This technique is going to be used to cool the servers in the white space. This is going to be explained further in Chapter 4.

2.5. Sources of Error

A potential source of errors using CFD to model a data center white space can arise due to incorrect modeling of the flow leaving the perforated tiles, modeling of the server racks, and deficiency of the turbulence model. It can also arise due to boundary conditions, unaccounted leakage effects, neglecting buoyancy, and coarseness of the mesh[53]. It has been investigated that a simplified representation of the server rack as a "black-box" model (where the flow inside it is not resolved) as compared to a detailed model, does not cause a significant difference in the CFD result (see [81]). Thus, Using a black-box model for the server racks does not contribute significantly to the source of error.

2.6. Turbulence Model

The $k - \epsilon$ turbulence model is a well-established model used in CFD modeling of Data Centers[81]. A potential source of error that has been further studied is the occurrence of cold and hot spots due to incorrect modeling of buoyancy. Buoyancy should be included in the CFD models, and its importance is shown in the paper "Improved CFD modeling of a small data center test cell"[39]. The inclusion of buoyancy in the CFD model is explained in Chapter 3.

2.7. Neural Network and Genetic Algorithm

Designing and optimizing the cooling requirement of both existing and newly designed data centers is essential[72]. Usually, CFD models are used to determine the temperature and velocity fields, but it is impractical for real-time thermal management as it is computationally expensive and time-consuming. Thus compact models are used. They can be divided into reduced-order models and statistically based data-driven models[57]. Data-driven predictive algorithms or meta-models can be sub-categorized into response surface methodology (RSM), artificial neural networks, and kriging models[56].

The Response Surface Method is based on mathematical and statistical procedures applied to smooth, continuous functions to build empirical models. The latter methods depend on stochastic patterns, which are better suited to capture the complex, nonlinear behavior of thermal transport by turbulent convection in large rooms[72]. Artificial Neural Network (ANN) is widely used in this category. The ANN can be trained with data from experiments or computational models. For instance, ANN has been applied to thermally manage the HVAC system of buildings[47, 75]. They are also used for thermal management of cabinet containing servers in data centers[42, 70–72, 79]. For example, Song et al.[72] used a neural network to predict the flow rate of floor grills and the temperature of servers in a data center. A neural network has also been used to predict the server temperature from temperature sensors present in the data center[58].

ANNs can be well integrated with optimization techniques like GA. Optimization of air distribution system operation and design[82], and absorption chiller design optimization[44] are a few examples of application in the HVAC field. GA has also been used in a variety of heat transfer applications[72, 78].

Proper Orthogonal Decomposition (POD) is a data-driven statistical method that has been extensively applied for prediction of temperature in data centers[50, 65, 67]. But studies have shown that POD is accurate for interpolating predictions but has a poor extrapolation accuracy, i.e., it is not able to predict beyond the input parameter space[49]. Moreover, predictions for a new set of data from the input space requires calculating fresh POD coefficients, which increases the complexity if the input parameter space is more than or equal to two dimensions[42]. Given these considerations, ANN is preferred over POD for developing the predictive model.

3 Theory

3.1. The Governing Equations of Fluid Flow

The conservation of mass, momentum, and energy can be used to derive the governing equations of fluid mechanics. By applying these equations over a point or integrating them over a region of the fluid, they can be stated in the differential or integral form, respectively. The fluid point is considered to be infinitesimally small such that it is not smaller than the molecules of the fluid, thus maintaining the continuum principle. All fluid properties are considered as a function of space and time.

The continuity (equation 3.1), navier-stokes (equation 3.2) and energy equation (equation 3.3) respectively for an in-compressible fluid form the conservation of mass, momentum and energy equations[68].

$$\frac{\delta}{\delta x_i}(\rho u_i) = 0, \tag{3.1}$$

$$\frac{\delta}{\delta t}(\rho u_i) + \frac{\delta}{\delta x_j}(\rho u_i u_j) = -\frac{\delta P}{\delta x_i} + \frac{\delta \tau_{ij}}{\delta x_j} + \rho g, \tag{3.2}$$

$$\frac{\delta}{\delta t}(\rho c_P T) + \frac{\delta}{\delta x_i}(\rho u_i c_P T) = -\frac{\delta}{\delta x_i}(K\frac{\delta T}{\delta x_i}) + S_h \tag{3.3}$$

There are five governing equations in total and seven unknown variables, namely the velocities in the three directions, pressure, density, temperature, and enthalpy. The constitutive equation of state is considered for enthalpy and density in order to close the system of equations. They are presented in equation 3.4.

$$\rho = \rho(P,T), \quad h = c_p T = h(P,T) \tag{3.4}$$

For an ideal gas, the ideal gas law is used to calculate the density and the specific heat capacity as a function of temperature which is shown in equation 3.5.

$$\rho = \frac{nP}{R_O T}, \quad dh = c_p dT, \quad c_p = c_p(P,T) \tag{3.5}$$

For flows in which gravity plays a role, the buoyancy source term is included. Buoyancy is due to the variation of density with respect to temperature. The boussinesq approximation is used to model the variation in buoyancy. The boussinesq approximation states that the variation of the density is only important in the buoyancy term i.e. $\Delta\rho g$, and can be neglected in the other terms of the momentum equation. Thus using Boussinesq approximation, the in-compressible continuity equation is $\nabla.u_i = 0$ and the momentum equation reduced to:

$$\rho_o(\frac{\delta}{\delta t}(u_i) + \frac{\delta}{\delta x_j}(u_i u_j)) = -\frac{\delta P}{\delta x_i} + \frac{\delta \tau_{ij}}{\delta x_j} + \Delta\rho g \tag{3.6}$$

In equation 3.6, the buoyancy term $\Delta\rho g$ term can be rewritten as $(\rho - \rho_o)g$, where ρ_o is the reference density. This buoyancy term can be further rewritten as $(\rho - \rho_o)g = -\rho_o\beta(T - T_o)g$, where β is the coefficient of thermal expansion and T_o is the reference temperature. For an ideal gas, $\beta = \frac{1}{T_o}$. Thus the buoyancy term becomes:

$$(\rho - \rho_o)g = -\rho_o(T - T_o)g/T_o \tag{3.7}$$

This is valid when the temperature and density variation is small ($\Delta T < 15°C$). Buoyancy is included as a source term in the vertical direction in the conservation of momentum equation.

For the flow to be laminar, the Reynolds number has to be below some critical value. At values above the critical Reynolds number, the flow transitions into turbulent flow, which is characterized by chaotic motion and random fluctuations. However, we are mostly interested in the ensemble-averaged properties. This is obtained from the averaged Continuity equation (3.8), Reynolds-Averaged Navier-Stokes (RANS) equation (3.9) and averaged energy equation (3.10), by decomposing the variables in the continuity, momentum and energy equation into a fluctuating and an average quantity. After decomposition, the average is taken over the whole equation. We get the following equations for the averaged quantities $\overline{u}, \overline{v}$, $\overline{w}$, and $\overline{T}$:

$$\frac{\delta \overline{u_i}}{\delta x_i} = 0, \tag{3.8}$$

$$\frac{\delta \overline{u_i u_j}}{\delta x_j} = -\frac{1}{\rho}\frac{\delta \overline{p}}{\delta x_i} + \frac{\delta}{\delta x_i}[\nu(\frac{\delta \overline{u_i}}{\delta x_j} + \frac{\delta \overline{u_j}}{\delta x_i}) - \overline{u_i' u_j'}] - g_i\beta(\overline{T} - T_o), \tag{3.9}$$

$$\overline{u_j}\frac{\delta \overline{T}}{\delta x_j} = \frac{\delta}{\delta x_j}(\alpha\frac{\delta \overline{T}}{\delta x_j} - \overline{u_j' T'}) \tag{3.10}$$

In equation 3.9, $-\rho\overline{u_i' u_j'}$ is the Reynold's stress term and in equation 3.10, $-\overline{u_j' T'}$ is the turbulent temperature flux which has to be modelled using closure models, and α is the thermal diffusivity.

The turbulence models based on RANS equations focus on the mean flow. The $k - \epsilon$ is a two-equation turbulence model that solves additional transport partial differential equations by solving for the turbulent kinetic energy and the rate of dissipation of the turbulent kinetic energy. The $k-\epsilon$ [59] is used as a turbulence model in this thesis as it is a widely validated model for industrial and environmental flows. However its performance is bad for flows involving adverse pressure gradient, rotating flows, and curved boundary layers to name a few. The $k - \epsilon$ model assumes the turbulent viscosity to be isotropic and the turbulence is locally isotropic (i.e. $\overline{u'}^2 = \overline{v'}^2 = \overline{w'}^2$). Thus the Reynolds stress is assumed to be proportional to the mean velocity gradients as shown in equation 3.11:

$$-\rho\overline{u_i'u_j'} = \mu_t(\frac{\delta\overline{u_i}}{\delta x_j} + \frac{\delta\overline{u_j}}{\delta x_i}) - \frac{2}{3}\rho k\delta_{ij} \tag{3.11}$$

The turbulent viscosity μ_t is modelled using the turbulent kinetic energy (k_e) and the dissipation (ϵ_p) as shown in equation 3.14:

$$\frac{\delta}{\delta t}(\rho k_e) + \frac{\delta}{\delta x_i}(\rho u_i k_e) = \frac{\delta}{\delta x_i}((\mu + \frac{\mu_t}{\sigma_k})\frac{\delta k_e}{\delta x_i}) + G_k + G_b - \rho\epsilon_p, \tag{3.12}$$

$$\frac{\delta}{\delta t}(\rho\epsilon_p) + \frac{\delta}{\delta x_i}(\rho u_i\epsilon_p) = \frac{\delta}{\delta x_i}((\mu + \frac{\mu_t}{\sigma_\epsilon})\frac{\delta\epsilon_p}{\delta x_i}) + C_{1\epsilon}\frac{\epsilon_p}{k_e}(G_k + C_{3\epsilon}G_b) - C_{2\epsilon}\rho\frac{\epsilon_p^2}{k_e}, \tag{3.13}$$

where G_k is the production of turbulent kinetic energy by the mean velocity gradient in the formula $G_k = \rho\overline{u_i'u_j'}\frac{\delta\overline{u_j}}{\delta u_i}$,$G_b$ is the turbulent kinetic energy generated by buoyancy, where $G_b = \beta g_i\frac{\mu_t}{Pr_t}\frac{\delta T}{\delta x_i}$, $Pr_t = 0.85$, $\beta = -\frac{1}{\rho}(\frac{\delta\rho}{\delta T})_p$ is the expansion coefficient. The turbulent viscosity can be modelled as shown in the equation 3.14:

$$\mu_t = \rho C_\mu\frac{k_e^2}{\epsilon_p} \tag{3.14}$$

The constant parameters in the standard $k - \epsilon$ model can be determined by data fitting from a wide number of experiments[77]. They are presented as follows:

$$C_\mu = 0.09, \quad \sigma_k = 1.00, \quad \sigma_\epsilon = 1.30, C_{1\epsilon} = 1.44, \quad C_{2\epsilon} = 1.92, \quad C_{3\epsilon} = 0 \tag{3.15}$$

Similarly a closure model can be assumed for the turbulent temperature flux $-\overline{u_j'T'}$, shown in the equation 3.16:

$$-\overline{u_j'T'} = \alpha_t(\frac{\delta\overline{T}}{\delta x_i}), \tag{3.16}$$

where $\alpha_t = \frac{\mu_t}{Pr_t}$

3.2. Near Wall Modelling

Close to a solid wall, the velocity reduces to zero due to the no-slip boundary condition. Near the wall, viscous force dominates over the inertial force. This means the flow can no longer be turbulent as the viscosity effect dominates. The region where the viscous force is going to be equal to or greater than the inertial force is called the viscous sub-layer. In this region the dimensionless velocity profile is related to the dimensionless wall distance as follows:

$$u^+ = \frac{\overline{u}}{u_\tau} = \frac{u_\tau y}{\nu} = y^+, \tag{3.17}$$

where $u_\tau = (\tau_w/\rho)^{1/2}$ is the wall friction velocity. The viscous sub-layer is only valid for $y^+ < 5$ and is very thin. Outside the viscous sub-layer, there is a region where both the inertia force and viscous force are equally important. This layer is called the log-law layer and is valid for $30 < y^+ < 300$. The dimensionless velocity profile of this layer is related to the dimensionless wall distance as follows:

$$u^+ = \frac{1}{\kappa} ln(y^+) + B, \tag{3.18}$$

where $\kappa = 0.4$ and $B = 2.0$ for turbulent flow having smooth wall at high Reynolds number[77]. In the $k - \epsilon$ model, the y^+ should be above 30 to apply the log-law profile to model the boundary layer.

3.3. Discretization of Governing Equations

CFD models are used to solve the governing equations of fluid mechanics as they rarely have analytical solutions. Differential equations are approximated by converting them into algebraic equations. This is done by using the discretization method like Finite Element Method (FEM), Finite Difference Method (FDM), and Finite Volume Method (FVM). 6SigmaRoom uses the Finite Volume Method. The computational domain is discretized into small discrete control volumes, and using the conservation of mass, momentum, and energy over these volumes. The control volume is meshed with hexahedral elements. In general, over each control volume, the governing equations are integrated to obtain discretized equations at the node, which is present at the center of the control volumes. Then the resulting system of algebraic equations is solved. Generally, the values of all the flow variables and properties are stored at the node[77].

In the 6SigmaRoom Software (to be used for CFD simulation of data center white space), the pressure-based finite volume solution uses a proprietary SIMPLE-like solution methodology to solve the governing equations which are discretized onto a structured or unstructured cartesian grid. To avoid typical solution difficulties associated with a stair-step representation from the Cartesian meshing of arbitrarily shaped objects, the Cartesian grid is supplemented with a treatment of surfaces that do not align with the grid using a proprietary cut-cell-like approach with proprietary enhancements, such as cell merging, to avoid small polyhedral cells creating numerical difficulties. The variable storage is staggered to take advantage of solution simplicity. The pressure and the scalar quantities are calculated and stored at the cell centers while the velocities are stored at the cell faces making the derivation of pressure much simpler, avoiding the need for the special interpolation scheme required when all the variables are stored at the cell centers, as in a collocated approach. For most data center applications, it is reasonable to assume that the flow is incompressible. The iterative solution procedure continues until the sum of the numerical errors for each variable is less than a specified fraction of the estimated incoming mass, momentum, energy, etc as appropriate. The default termination criteria is 10^{-3} times the incoming mass, momentum, and energy as appropriate for the particular equation, but the user can specify any chosen value as appropriate for their problem. If a time-dependent simulation is being undertaken, the solution uses an implicit time marching scheme, known as the Backward Euler Method, with the responsibility of the user to choose a sufficiently small time step for the solution to be time-step independent. The default settings for the solution approach for a structured Cartesian grid used in this research include spatial discretization - 1^{st} order upwind, the Algebraic Multi-Grid method is used to solve the energy equation, the conjugate residual linear solver with ILU_o (incomplete LU) preconditioner is used to solve the pressure.

3.4. Uncertainities and Errors

There is always some errors and uncertainty involved with CFD modeling. Numerical errors are due to identifiable insufficiency in the CFD models. It consists of Round-off Errors, Iterative Convergence Errors and Discretisation Errors. Round-off error is caused due to the representation of the real number with a finite number of significant digits. It can be controlled. Iterative Convergence error is caused due to stopping of the iterative method after certain number of iterations. This error can be controlled by stating how close the numerical value should be to the exact value. In an iterative procedures,in steady-state simulations, the scaled/normalized value of the residual of an equation at an iteration is compared with a user-specified value. If the residual is below the user-specified value, then the equation is said to be converged. Discretization Error occurs due to converting the continuous differential equation into discrete algebraic equations in the discrete domain. Discretization Error can further be classified into Spatial Discretization Error (SPD) and Temporal Discretization Error (TDE). SPD can be controlled by doing a Grid Independence Test and TDE can be controlled by examining the temporal convergence.

Uncertainties are insufficiency in the CFD models due to a lack of knowledge. Uncertainty can be due to input uncertainty, which is due to the difficulty in representing the physical domain, boundary conditions, and properties of the fluid. The other type of uncertainty is due to physical models which are representing the fluid flow phenomena using empirical models like turbulence models.

3.5. Latin Hypercube Sampling Technique (LHS)

The Latin Hypercube Sampling (LHS) technique generates a random sample of values that are present in the multidimensional space[74]. Each variable in the LHS constitutes a dimension in the multi-dimensional space. LHS distributes the samples evenly over the sample space. It can be used to generate better sample results, with a lower standard error, and in less number of trials[10].

A latin square is defined as a square grid containing sample points, such that there is only one sample present in each row and each column of the square grid. A hypercube is defined as a cube with more than three dimensions. A latin hypercube is a generalized version of the latin square in multidimensional space.

When sampling N variables using LHS, the range of each variable is divided into M equally probable intervals. The value of M is determined by the number of sample points to be generated. The size of each interval in a dimension is equal to each other. The advantage of using LHS is that with the increase in more number of dimensions, it does not require more number of sample points. The other advantage is that while choosing a sample point from an interval, it remembers which samples were picked before.

LHS has its advantage over random sampling technique. In a random sampling technique, new sample points are generated without considering the previously generated sample points. This can lead to duplication of the samples, which can be avoided in LHS. Also, in a random sampling technique, it is not required to know the total number of sample points needed to be generated. Whereas, the number of sample points needed to be generated in LHS is required to be known beforehand. This also prevents duplication of the samples generated.

Thus, the LHS is preferred as a sampling method as it has an unbiased representation of the samples being generated[10].

3.6. The Neural Network

The idea of an Artificial Neural Network (ANN) is inspired from the biological neural network that is constituting the human brain. Artificial neural network is suitable to predict the thermal model of data centers as they can establish a non-linear relationship between the input design variables and the thermal variables (temperature and flow profile)[48, 54]. An ANN consists of a time-consuming training process (which depends on the computational resources available, the training data, and the complexity of the problem). Once it is trained well and it can predict new unseen data accurately, the neural network can be deployed for real-time prediction of the thermal quantities in a data center. It can later assist evolutionary algorithms like genetic algorithm for optimization of design variables. Existing ANN can be updated using more training data and can also be used to train new models using the concept of transfer learning[36].

The structure of an artificial neural network is shown in figure 3.1. It consists of an input layer, a hidden layer(s), and an output layer. Each layer may consist of multiple neurons. These neurons behave similarly as the neurons found in the human nervous system.

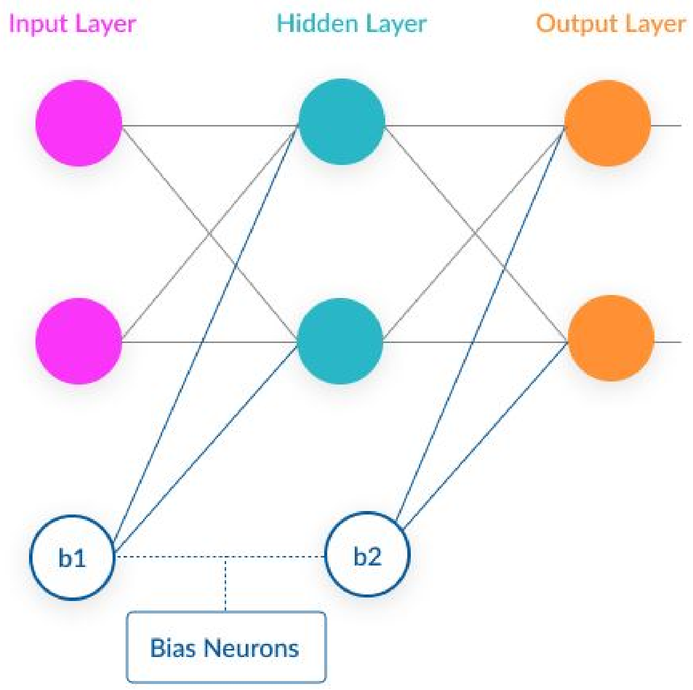


Figure 3.1: The topology of a simple Artificial Neural Network[5]. It consists of an input layer, one or more hidden layers, and an output layer. Each layer has more than one neurons and a bias neuron. There is no bias neuron associated with the output layer. The value of the bias is a constant real number and maybe different for each layer.

A simple neuron is also called a Perceptron. Each neuron in a layer is associated with the neurons and bias unit present in the preceding layer, through weights connecting them. This is shown in figure 3.1. The bias unit is represented as $b1$ and $b2$. The bias unit is like an extra neuron added to all the layers except the output layer, and generally has a value of $+1$. They are not connected to any previous layers. A bias unit is like an intercept of a linear equation. It is used to adjust the output of the neuron. This improves the performance of the neural network.

The value of a neuron is dependent on a transfer function f. The input for the transfer function is dependent on the weighted sum of the value of the neurons in the preceding layer and the bias. This is shown in equation 3.19, where y_k^m is the value of the k^{th} neuron present in the m^{th} layer, x_l^{m-1} is the value of the l^{th} neuron in the $(m-1)^{th}$ layer, assuming there are n neurons present in the $(m-1)^{th}$ layer. w_{lk} is the weight connecting the k^{th} neuron in the m^{th} layer to all the l^{th} neuron in the $(m-1)^{th}$ layer. b_k^{m-1} is the weight of the bias connecting the k^{th} neuron in the m^{th} layer to the value of the bias unit in the $(m-1)^{th}$ layer

$$y_k^m = f(\Sigma_{l=1}^{n} w_{lk} x_l^{m-1} + b_k^{m-1} * (Value\ of\ Bias\ Unit)) \tag{3.19}$$

There can be various transfer functions[12] that can be used based on the type of the problem being predicted i.e. regression or classification problem[1]. An example of a transfer function is the hyperbolic tangent (sigmoid) function given in equation 3.20 and shown in figure 3.2. The output of this function is between -1 and $+1$.

$$tanh(n) = \frac{2}{1+e^{-2n}} - 1 \tag{3.20}$$

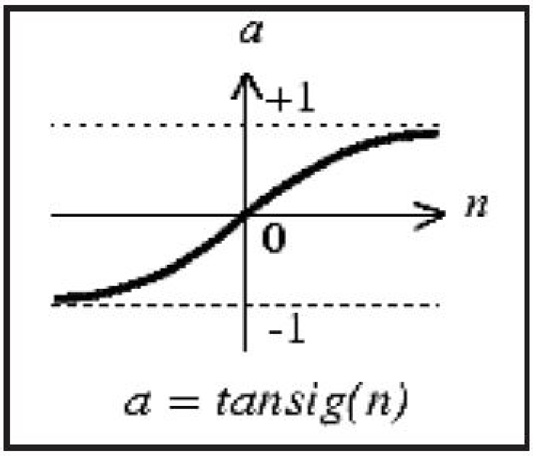


Figure 3.2: The hyperbolic tangent activation function[12]. It is one of the transfer function used to bring non-linearity in a neural network. The value of the tanh function (vertical axis) is between $+1$ and -1.

The way these weights are calculated is though backpropogation[66] technique which employs the gradient descent method to minimize a cost function in order to find the appropriate values of these weights. One example of cost function is the Mean Square Error (MSE) given by equation 3.21

In a neural network, there are certain parameters which the neural network cannot learn by itself, i.e., by gradient descent. These parameters are called the hyperparameters. The values of the hyper-parameters have to be specified before the network starts the training process. The hyperparameters determine the structure of the neural network. The training of the neural network model is affected by the hyperparameters, like the number of neurons, number of hidden layers, the activation function(s), type of training algorithm, etc. One way to find these hyperparameters is through trial and error, or through an automated algorithms

[1]A regression problem is when the output consists of continuous values, like for example predicting the price of a house. A Classification problem is when the output consists of discrete values, like for example to predict if a image contain a cat or a dog.

like HyperOpt[24] or genetic algorithm. A Dense Neural Network (DNN) is a kind of neural network where each neuron in a layer (except the input layer) is connected to all the neurons in the previous layer (densely connected). A simplified representation of a DNN is shown in figure 3.1.

A neural network learns to identify the underlying pattern from a given distribution of the output data by updating its weights after seeing a part of the training data. In order to make the neural network models learn efficiently and quickly, the values of hyper-parameters have to be specified. The hyper-parameters influences the behavior of a neural network heavily, and randomly setting the hyper-parameters may lead to less accurate model. Before diving into the various hyper-parameters, a few other concepts have to be introduced.

Error Function - The error between the predicted result and the actual result is calculated using an error function. In regression problems, Mean Squared Error (MSE) and Mean Absolute Error (MAE) are generally used. In equation 3.21, it is seen in MSE that the square of the error is taken, which makes the more significant error more pronounced, penalizing them more. y is used to represent the output variable(s).

$$MSE = \frac{1}{Numberofsamples(N)} \Sigma_{j=1}^{N} (y_{actual} - y_{predicted})^2, \tag{3.21}$$

$$MAE = \frac{1}{Numberofsamples(N)} \Sigma_{j=1}^{N} |y_{actual} - y_{predicted}| \tag{3.22}$$

Metrics - It is a function that is used to judge the performance of the model. Usually, MSE or MAE is not enough to judge the performance. Thus the so-called R-Squared (R^2) value is used to judge the performance of the neural network in regression problems. The R^2 is calculated as shown in equation 3.23. The value of R^2 can range from $-\infty$ to 1. A R^2 of 1 indicates that the prediction model is performing accurately and is able to predict all the ground truth values. $-\infty$ indicates the worst possible performance. It is preferred to have the R^2 value close to 1.

$$R^2 = 1 - \frac{(y_{actual} - y_{predicted})^2}{(y_{actual} - y_{mean})^2}, \tag{3.23}$$

where $y_{actual}$is the actual value of a output in the set of outputs, $y_{predicted}$ is the predicted value corresponding to the actual value of the output, and y_{mean} is the average of all the actual value of the outputs.

A dataset, is split into a training dataset and a testing dataset. The training dataset consist of data on which the neural network trains, i.e., it adjusts the weights of each neuron to match the ground truth values from the training dataset. The weights of each neuron are adjusted based on the gradient descent of the cost function (Error function). The neural network then checks its performance on the testing dataset. A testing dataset consists of data that the neural network has never seen before. Usually, the train-test split is set in the ratio $80\% - 20\%$ (i.e., 80% of the original dataset is used for training, and 20% is used for testing) for a limited size of the dataset, and $90\% - 10\%$ for large dataset.

The goal of the neural network is to generalize well, i.e., the neural network model should be able to predict with high accuracy on the unseen data points. During training a model, most commonly encountered problems are overfitting and underfitting of data, which is also known as the bias-variance problem.

Over-fitting is due to the neural network learning too much from the training dataset and being unable to predict the test dataset sensibly. This occurs due to high variance, i.e., the NN model becomes sensitive to small fluctuations in the training dataset. The model captures all the trends in the training dataset and not the dominant trend.

Under-fitting is due to the neural network unable to learn from the training dataset, resulting in unreliable predictions for the test dataset. Underfitting occurs due to high bias in the training data causing it to miss relevant relation between the input features and the output variables.

ANN model can be checked if it has generalized well if the metrics for the training dataset and testing dataset are almost equal to each other, and there is little difference between them. This is shown in the figure 3.3[14].

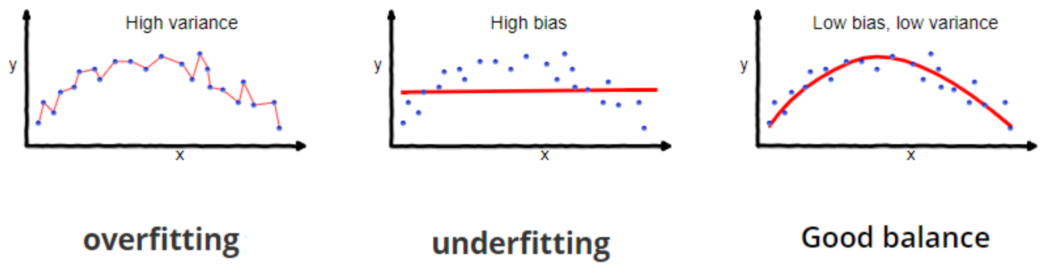


Figure 3.3: The predictions of the neural network plotted on the Y-axis vs. its input features on the X-axis. The red line is used to fit the data points. The graph on the left shows that the model is overfitting (high variance), which means the model is susceptible to small changes in the training dataset. The graph in the middle shows that the model is underfitting (high bias), suggesting that the model is not able to learn from the training dataset. The graph on the right has a good balance (low bias, and low variance), i.e., it can sensibly predict from the training dataset (Generalized)[14].

Qualitatively, a neural network is said to have a high bias if it has low training accuracy and low testing accuracy. A neural network is said to have high variance if it has a high training accuracy and a low testing accuracy.

The hyper-parameters used in the neural network are described as follows.

1. Train-Test Split - This is used to split the dataset into training and testing data set. Usually, the $80\% - 20\%$ ratio of the split is followed.
2. Batch Size - It is the number of samples used for training the neural network per gradient update. If the batch size is 1, then the training time increases, and the accuracy of the model is affected, i.e., it might overfit or underfit the data. If the batch size is equal to the size of the training dataset, then the model starts learning the pattern in the training data and is going to lead to overfitting. Thus a mini-batch size of a power of 2 (because some hardware like the Graphics Processor Unit (GPU) achieves better run time when the batch size is of a power of 2) is chosen, which is less than the training dataset size. Generally, a batch size of 32 or 64 is chosen.
3. Learning Rate - It defines how quickly a neural network can learn from the training dataset. The learning rate determines the step size required while moving towards reducing the cost (error) function during gradient descent. If the learning rate is too low

then the training time is increased, and if it is too large, then the NN model misses the global minimum of the cost function. This is shown in figure 3.4.

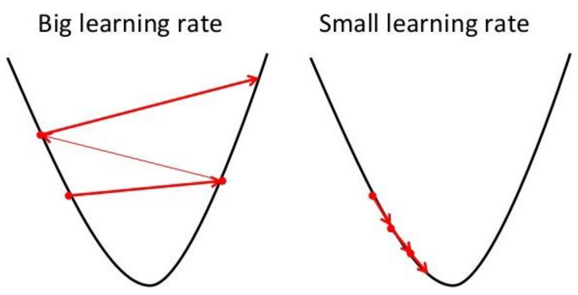


Figure 3.4: The graph shows the influence of the learning rate while reaching the global minima of a convex shaped cost (error) function. If the learning rate is big (indicated by the red arrow), the NN model misses the minima of the cost function (shown on the left graph) and if the learning rate is too low, then longer time is required to reach the minima [27].

4. Epochs - A epoch is the length of the training dataset. The number of epochs is how many numbers of times the whole training dataset is shown to the neural network while training. Usually, the models are trained to a certain epoch, after which the model starts to overfit.

5. Number of Hidden Layers - The number of hidden layers has to be chosen such that the model does not overfit or underfit.

6. Neurons in each Hidden Layer - This determines the number of neurons required in each of the hidden layer(s), such that the neural network model trains without any overfitting or underfitting.

7. Dropout Layer - This is a regularization method for neural networks proposed by Srivastava et al.[73] to prevent overfitting. In this technique, neurons in the layer where dropout is applied are randomly ignored during the forward pass, and their weights are not updated in the backward pass. This results in a network being less reliant on specific weights of neurons, which leads to the network being well generalized. The value of a dropout ranges from 0 to 1. Values close to zero indicates that fewer neurons in a layer have to be randomly dropped out, and 1 indicates that all the neurons of the layer have to be dropped out. Dropout is applied only to the pre-output layers.

8. Activation Function - An activation function is a mathematical equation which determines the output of a neuron. Each neuron is associated with an activation function. An activation function introduces non-linearity in the neural network model. It determines if the neuron's input is important to the prediction or not. Traditionally, a Tanh or Sigmoid activation function was used to train the neural network. But these two activation functions suffer from the vanishing gradient problem[37], where during the back-propagation and gradient descent, the gradient of the error function w.r.t the current weight becomes increasing small such that the weights of the neuron do not change anymore and the model stops learning from the training dataset. On the other hand, activation function whose derivatives are too large would lead to an Exploding Gradient problem. Recently, the Rectified Linear Unit (ReLU) activation function or a variant of ReLU was used for the neurons in the input layer and the hidden layers[32] and a Linear activation function is used for the neurons in the output layer. ReLU suffers less from the vanishing gradient

problem[33].

9. Optimizer - Optimizers are algorithms required to change the weights and learning rate of the neural network to reach the global minimum of the error function. The weights are updated using the optimizers, and the error function guides the optimizer in reaching the global minimum of the error function. Optimizers are generally variants of the gradient descent algorithm. Various optimizers are available to train a neural network[29]. The three most commonly used optimizers in a neural network are Adagrad, RMSprop, and Adam. ADAM[1] is a popular gradient descent optimizer. It is generally used because it is robust, computationally efficient, requires less computational memory than the other optimizers, and can effectively solve practical problems. Another reason why ADAM is used as an optimizer is because of its dynamic momentum calculation process[13]. This speeds up the training process, and the neural network arrives at the global minima quickly.

3.7. Genetic Algorithm

A Genetic Algorithm is a kind of evolutionary algorithm, which is a heuristic search technique inspired by Charles Darwin's theory of natural selection and evolution. Like the process of natural selection, it selects the fittest individuals for reproduction, to produce offspring for the next generation. John Holland introduced this concept in 1960 based on the concept of Darwin's theory of evolution, and later his student David E. Goldberg[22] further extended it in 1989[21].

The natural selection process starts with the selection of the fittest individual from a population. The offspring inherits the characteristics of the parents and is then added to the next generation. The fitness of the offspring depends on its parents (individuals from the previous generation). Offspring of the current generations with better fitness values are selected to produce offspring for the next generation. This process continues until we get the required fitness value or reach a particular generation.

A Genetic Algorithm consists of five phases.

Phase one is the Initial Population. This phase consists of a set of individuals of fixed size, and fixed population size. An Individual consists of a set of parameters called Genes. The genes are joined together to form a Chromosome, which can equivalently be called as an individual. The objective function assesses the genes of the individual and then assigns a fitness value to the individual. For example, an objective function can be to maximize or minimize the value of an individual. This is shown in figure 3.5.

Phase two is the Fitness Function. It is an objective function (for example, a maximization or a minimization function) that determines how to fit an individual is. A fitness score is assigned to each individual. The value of the fitness score determines if the individual is going to be selected for reproducing offsprings for the next generation.

Phase three is the Selection. It is used to select individuals for reproduction and pass on their genes to the offsprings. The offsprings, along with the individuals in the previous generation whose fitness is above a specific threshold value, form the population of the next generation. The selection is based on the fitness scores of the individuals. Individuals with higher fitness have a higher probability of getting selected.

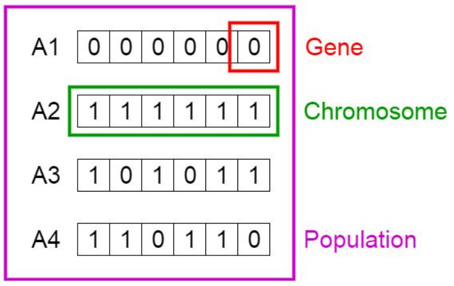
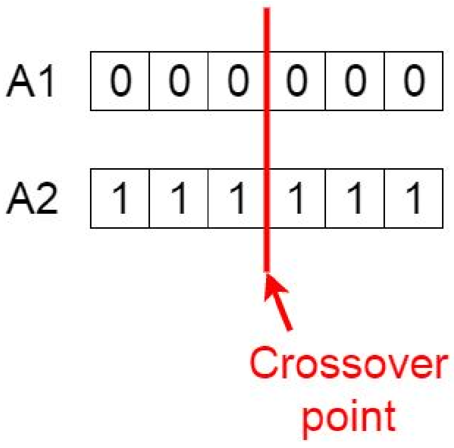
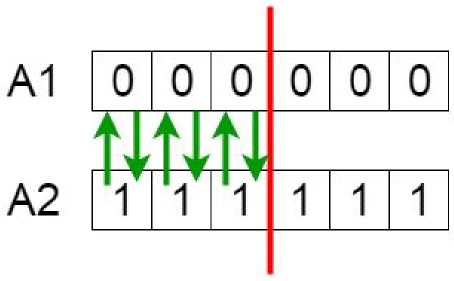


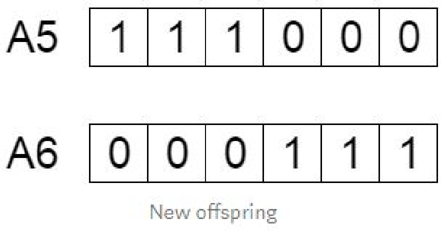
Figure 3.5: The representation of gene, chromosome and population, which are used to implement a genetic algorithm. A gene is the basic element of a chromosome. A collection of genes (more than two) forms a chromosome (Individual). A collection of chromosomes (more than one) is called population. In this figure a gene consists of binary values[8].



(a) Randomly placing the crossover point in the parent individuals.



(b) Gene exchange between parents until the crossover point.



(c) The offspring produced after crossover operation.

Figure 3.6: The crossover phase to produce two offspring[8]. The crossover point is placed randomly. The gene exchange happens between the parent chromosome till the crossover point. This produces offspring, with genes inherited from its parents chromosomes.

Phase four is the Crossover. It is the most important phase. A crossover point is placed randomly within the genes present in the two parents to be mated, as shown in figure 3.6a. The genes are exchanged between the parents until the crossover point is reached, as shown in figure 3.6b. Thus the new set of offspring is produced as shown in figure 3.6c, which are then added to the population.

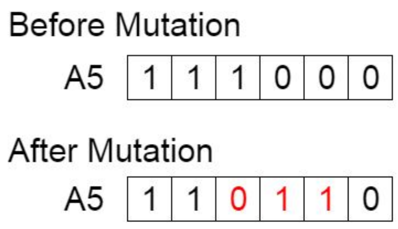


Figure 3.7: Mutation of the genes of the offspring[8]. In this case, it is done by flipping a bit. Mutation is done to maintain diversity in the population and not create a bias

Phase five is the Mutation. In certain offspring, some of their genes are subjected to a random probabilistic change. In the diagram shown in figure 3.7, it means that a random bit in the string of bits is chosen and flipped. The mutation operation is done to maintain diversity in the population and prevent early convergence.

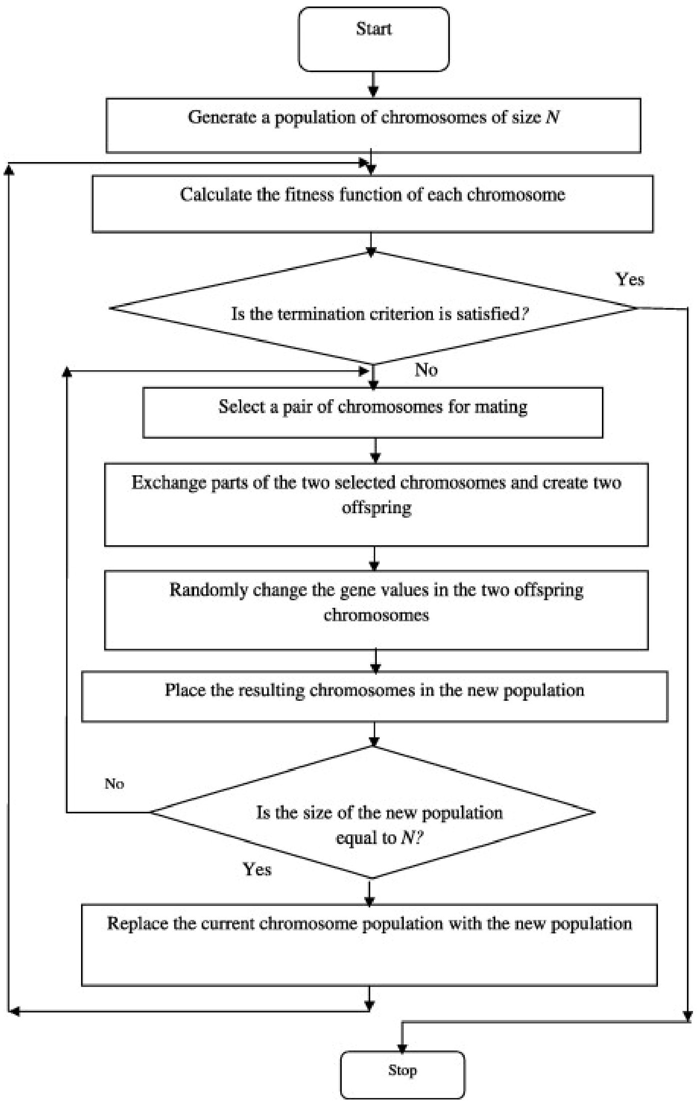

Start
Generate a population of chromosomes of size N
Calculate the fitness function of each chromosome
Is the termination criterion is satisfied?
Yes
No
Select a pair of chromosomes for mating
Exchange parts of the two selected chromosomes and create two offspring
Randomly change the gene values in the two offspring chromosomes
Place the resulting chromosomes in the new population
No
Is the size of the new population equal to N?
Yes
Replace the current chromosome population with the new population
Stop

Figure 3.8: The flow chart for the implementation of a genetic algorithm. It consist of five phases namely initial population, fitness, selection, crossover, and mutation[63]. Algorithm is terminated on reaching a certain fitness value or certain generation.

Finally, the algorithm ends if the required fitness has been reached by an individual, or no change in the fitness value is being observed for each successive generation, or a particular generation number has been reached. A flow chart of the whole process is shown in figure 3.8.

A simple genetic algorithm can optimize a single objective function. If more than two objective functions have to be optimized simultaneously, then a multi-objective evolutionary algorithm has to be used.

Multi-objective Evolutionary Algorithm (MoEA), has to deal with multiple objective functions that are conflicting with each other. Let's take an example where there are two objective functions, and we want to minimize both of them simultaneously. During the optimization process, it is observed that if the value of one objective function is increasing, then the value of the other is decreasing. This does not lead to a global optimum solution (in this case the global minima), as both the objective function cannot be minimized simultaneously due to the trade-off between the objective functions. This gives us a set of possible solutions that might be considered as the set of best possible solutions. In MoEA, a set of solutions that satisfies all the constrain functions and variables, form the Feasible Region (R). Before diving into how the algorithm is going to be used, there are a few other concepts to be defined.

1. Domination - A solution "a" is said to dominate solution "b" if "a" is not worse than b for all the objective functions and "a" is strictly better than "b" in one of its objective function.

2. Non-Dominated Set - From the feasible region "R", a non-dominated set of solutions "R'" refer to the subset of R, such that any member in this subset R' does not dominate over each other. For example, if we consider two solutions "h" and "m" from the region "R'", each of them having two objective functions. Both the objective functions have to be minimized simultaneously. Solutions "h" and "m" can form a non-dominated set, if one of the objective function of "h" is greater than its counterpart in solution "m" and the other objective function of "h" is less than its corresponding counterpart in solution "m".

3. Pareto-optimal Set - The non-dominated set (R') of the feasible space (R) forms the Pareto Optimal Set. An example is shown in figure 3.9. In this figure, the objective functions are the deflection and weight of the beam. Both the objective functions have to be minimized simultaneously. The variables deciding the values of these two objective functions are the length and diameter of the beam. It is observed in this figure that a global solution, i.e., a global minimum, cannot be achieved. The global minimum can be achieved by minimizing the two objective functions simultaneously. As seen in figure 3.9, if the deflection is minimized, then the weight of the beam is maximized. A trade-off has to be made between the deflection and the weight of the beam, which leads to a set of non-dominated solutions forming the Pareto front.

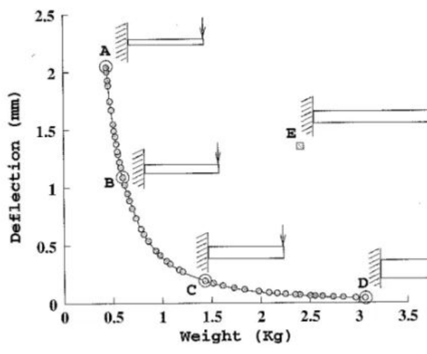


Figure 3.9: The graph shows the Pareto Optimal Front for the deflection of the beam on the vertical axis and the weight of the beam on the horizontal axis. Both the deflection and the weight of the beam have to be minimized. A trade-off has to be made between the deflection and weight of the bean, which results in a Non-Dominated Set. This forms the Pareto Front. A global solution cannot be achieved. Five possible solutions (A/B/C/D/E) of the Pareto Front is shown which is obtained by changing the length and diameter of the beam[30].

Non-dominated Sorting Genetic Algorithm-II (NSGA-II) is an evolutionary algorithm developed by Deb et al.[46] to solve a multi-objective optimization problem. NSGA-II follows the elitist principle, i.e., the best of the population are provided the opportunity to carry forward to the next generation. It also preserves the diversity of the population by calculating the crowding distance. Crowding distance is a mechanism to rank the members in a Pareto front. In the procedure for calculating the crowding distance, members of the Pareto front, present in a less dense region are preferred over members present in a more dense region. The following procedure is followed by NSGA-II to find the best Pareto front.

1. First, it performs the non-dominated sorting of the combination of parent and offspring population. It then categorizes them into fronts, ranking them in descending levels of non-domination. Offsprings are produced by crossover and mutations operations.
2. Secondly, it fills in the new population from the Pareto fronts based on their ranks.
3. If one of the Pareto fronts is partially taken, it selects the members of that front using crowding distance, i.e., populations present in the less dense regions of the front are selected.
4. It again creates the offspring population from this new population via tournament selection, crossover, and mutation. Tournament selection refers to generating a new population by fronts ranking. If the front ranks are equal, then selection of the population is done by applying the crowding distance. These steps are looped till a particular generation has been reached or a certain termination criterion has been reached. The process is shown in the figure 3.10[28].

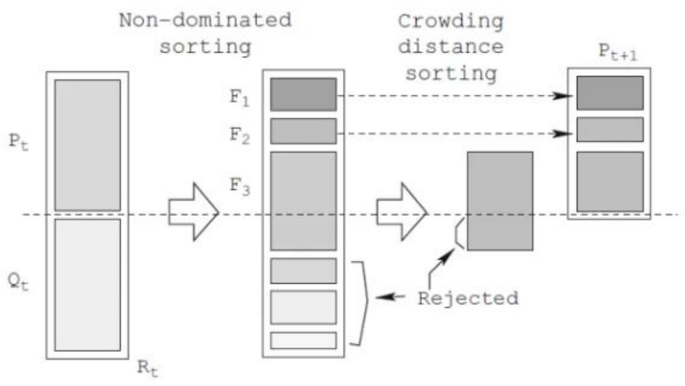


Figure 3.10: The Schematics of an NSGA-II procedure. It first creates the Parent Population (P_t) and the Offspring Population (Q_t). Then using a non-dominated sorting technique, it selects the various fronts (F_1, F_2, F_3) in descending order of their ranks. The population of the partially selected front (F_3) is selected using the crowding distance. Thus a new population (P_{t+1}) is created which then produces the Offspring by Crossover and Mutation[46].

4 Methodology

4.1. Outline of the Steps taken to Obtain the Optimum Design Parameters of Data Center White Space

The final goal of this research is to find the optimum set of design parameters of the data center white space required to maintain the servers below a certain temperature. To achieve this, first using a random sampling technique, i.e., Latin Hypercube Sampling (LHS) technique, a set of random samples are generated for the design variables, which is explained further in section 4.3. Then using the randomly generated values of the design variables, numerous CFD models are designed and simulated in the CFD software 6SigmaRoom to obtain the required results as explained in section 4.4. Using the results (tile flow rate of the floor tiles and maximum cabinet inlet temperature of the cabinets) obtained from the CFD simulations, and the randomly generated values of the design parameters from the LHS technique, an Artificial Neural Network (ANN) is trained on these values to predict the tile flow rate and the cabinet temperature. ANN is programmed in Python using the Keras[25] and Scikit[34] Learn library. This is explained in section 4.5. Finally, using the GA and the ANN, the optimum value of the design parameters are found, which are required to maintain the servers inlet temperature below the maximum limit of the Recommended ASHRAE Temperature, i.e., below $27°C$. The GA is also programmed in Python using the DEAP[18] library. This is shown in section 4.6. The results of the Genetic Algorithm are verified by running CFD simulations using the values of the optimized design parameter of the white space. Then the best result among all of the optimized cases is chosen. In this study, the genetic algorithm was used for two purposes. One, it was used to find the parameters required to obtain the hyperparameters of the neural networks and, two a variant of GA (NSGA-II) was used to find the optimized design parameters of the data center white space. The flow diagram of the methodology is shown in figure 4.1

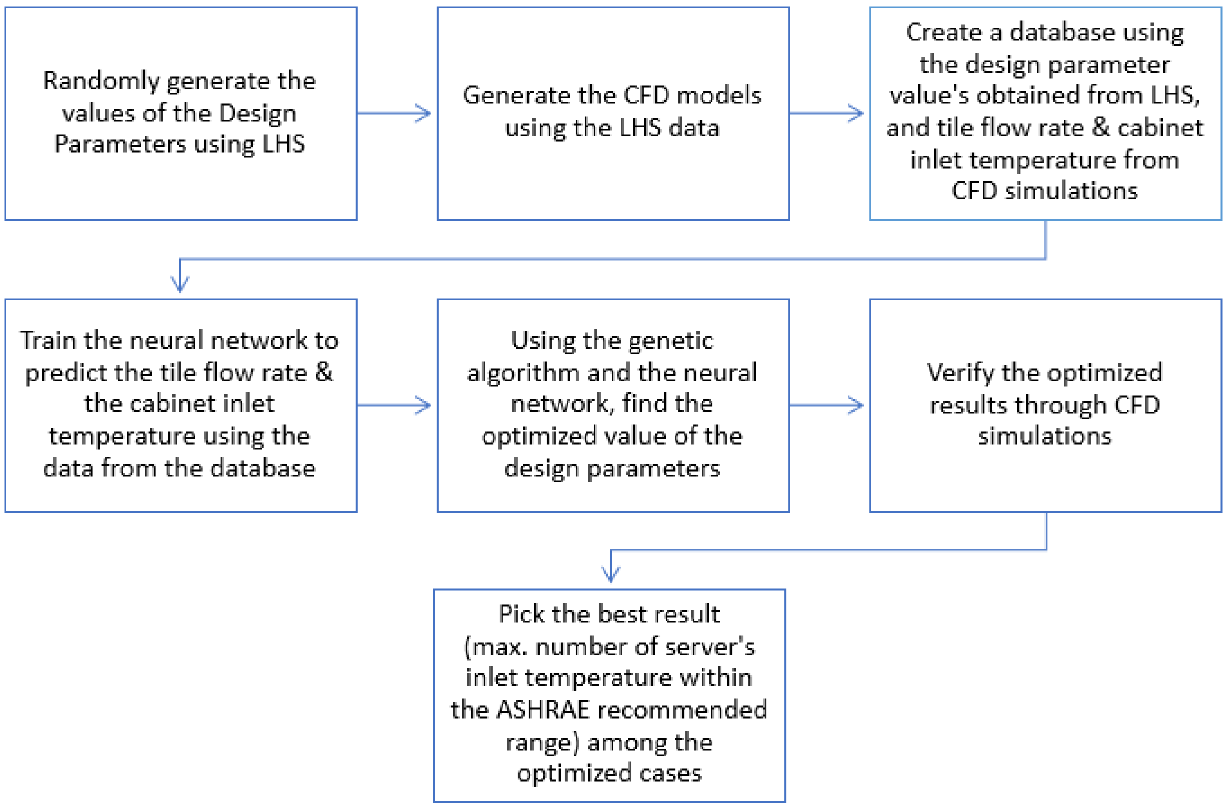
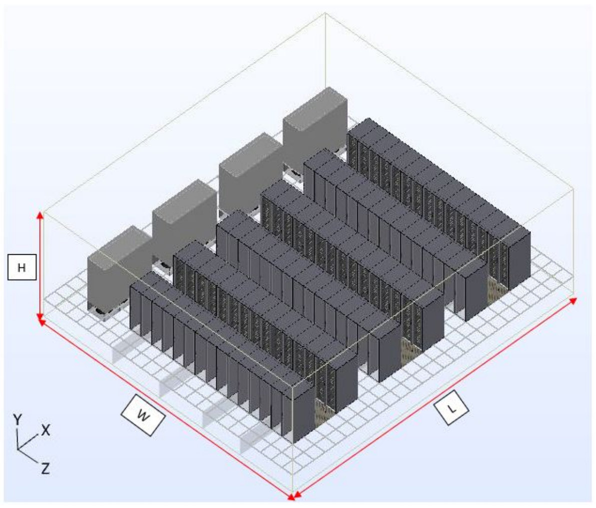

Randomly generate the values of the Design Parameters using LHS
Generate the CFD models using the LHS data
Create a database using the design parameter value's obtained from LHS, and tile flow rate & cabinet inlet temperature from CFD simulations
Train the neural network to predict the tile flow rate & the cabinet inlet temperature using the data from the database
Using the genetic algorithm and the neural network, find the optimized value of the design parameters
Verify the optimized results through CFD simulations
Pick the best result (max. number of server's inlet temperature within the ASHRAE recommended range) among the optimized cases

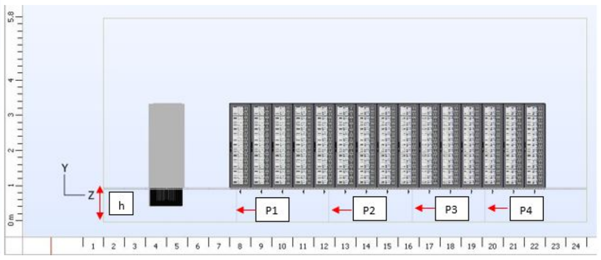
Figure 4.1: The flow-diagram of the methodology followed in this thesis.

4.2. The Data Center White Space Model Setup

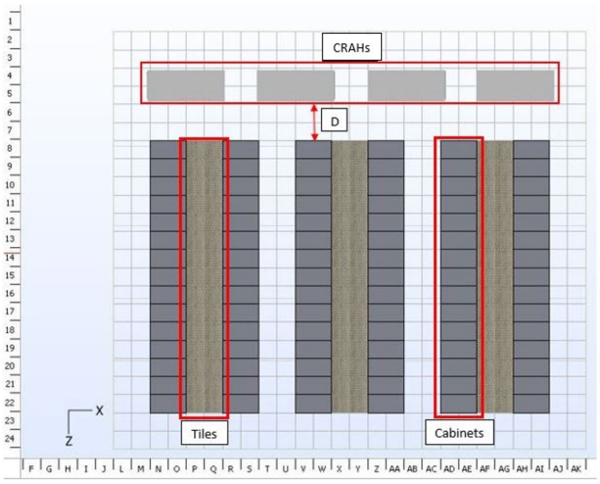
A white space of a data center consist of the area where the IT equipments are placed. A white space mainly houses the servers, network gears, cabinets, air-conditioning units, Power Distribution Units (PDUs), cable trays, and cooling ducts. In this study, a simplified model of a raised floor data center white space is created containing only the cabinets, the servers, the floor tiles, the air-conditioning units (CRAHs), and the perforated plates. This is done to simplify the modeling of the data center white space. There is no containment strategy used in the study.



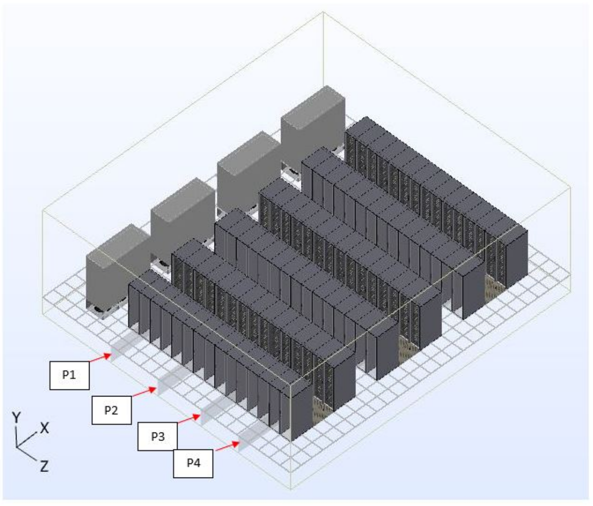
(a) An isometric view of the white space.



(b) The height (h) of the raised floor and positions of the perforated plates (P1 to P4) in the raised floor.



(c) The positions of the CRAHs, floor tiles and cabinets (containing servers) in the white space.



(d) The perforated plates placed below the raised floor.

Figure 4.2: A simplified representation of the data center white space, housing the IT equipments. The dimensions of the white space is shown in (sub-figure (a)); the height of the raised floor in (sub-figure (b)); the position of the four CRAHs, 90 floor tiles and 90 cabinets in (sub-figure (c)) and positions of the four plates (P1, P2, P3, and P4) in (sub-figure (b)) and (sub-figure (d)).

In figure 4.2a, an isometric view of the white space is shown. X-Z is the horizontal plane, and the Y-axis is in the vertical direction. The dimensions (L, W, H) of the white space are shown in the table 4.2. In figure 4.2c, giving a top view of the white space, it is seen that there are four CRAHs, rows of floor tiles to provide cooling air to the inlet of the servers and rows of cabinets containing the servers present in the white space. In total, there are 90 cabinets and 90 tiles present ,i.e., one floor tile for each cabinet. In figure 4.2d and figure 4.2b, the perforated rectangular plates ($P1$, $P2$, $P3$, and $P4$) are positioned in the underfloor. Each plate has numerous circular perforations, each with a diameter of $0.003m$. The amount of perforation of each perforated plate is different. The position of each perforated plate is also different in the Z-direction. In figure 4.3, the dimensions of a plate is shown. The length (P_L) of the plate spans across the room in the X-direction. The height (P_H) of the perforated plate is determined by the height (h) of the raised floor.

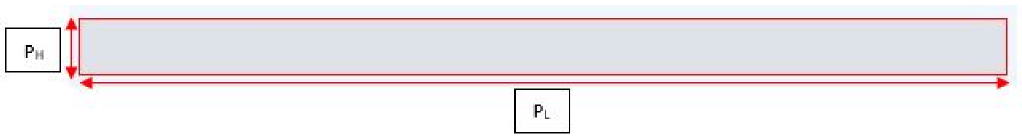
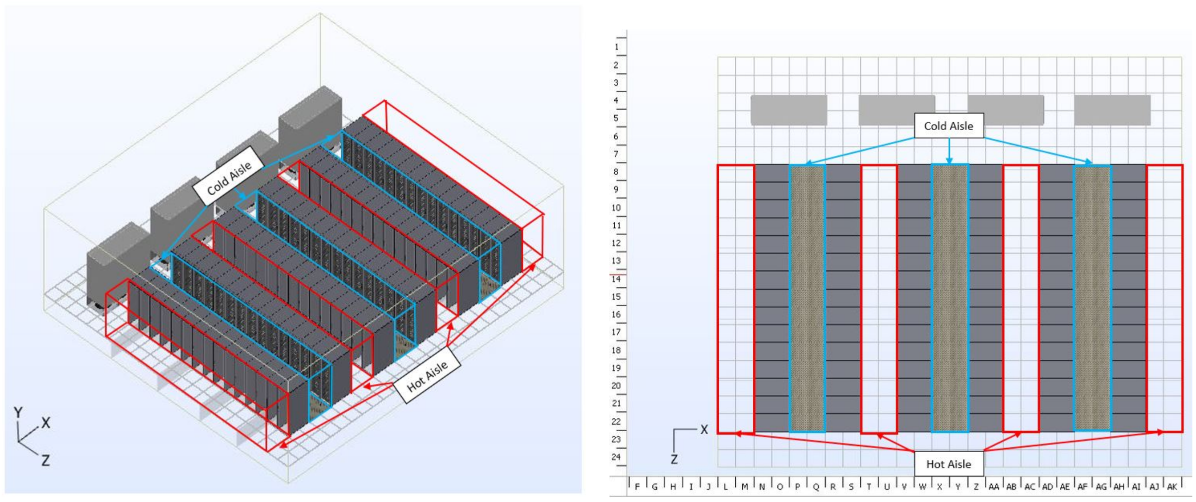


Figure 4.3: The dimensions of the rectangular perforated plate which is placed below the raised floor.

Length L (m)	Width W (m)	Height H (m)
15.6	13.8	5.8

Table 4.1: The dimensions of a simplified data center white space.

In figure 4.4, it is shown how the rows of cabinets in the white space are placed in Hot Aisle (highlighted in red) / Cold Aisle (highlighted in blue) arrangement to reduce the mixing of the hot air with the cold air.

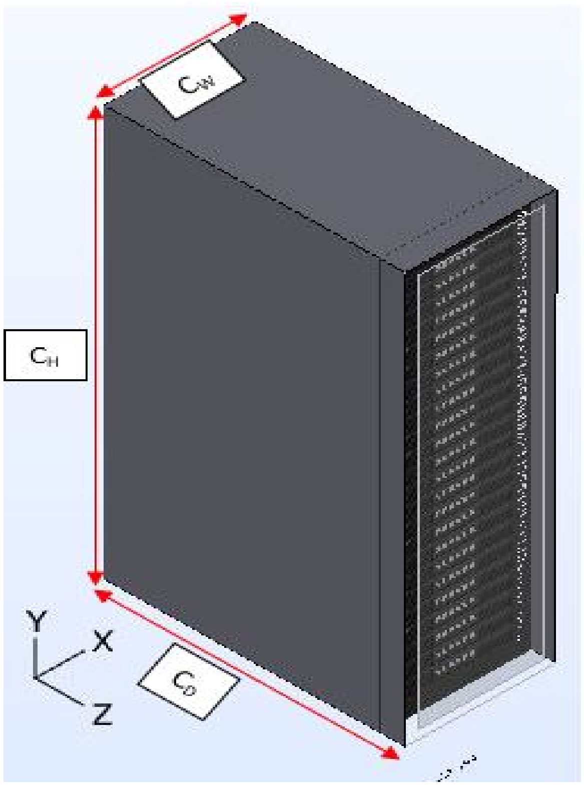


(a) The isometric view of the Hot Aisle / Cold Aisle arrangement.

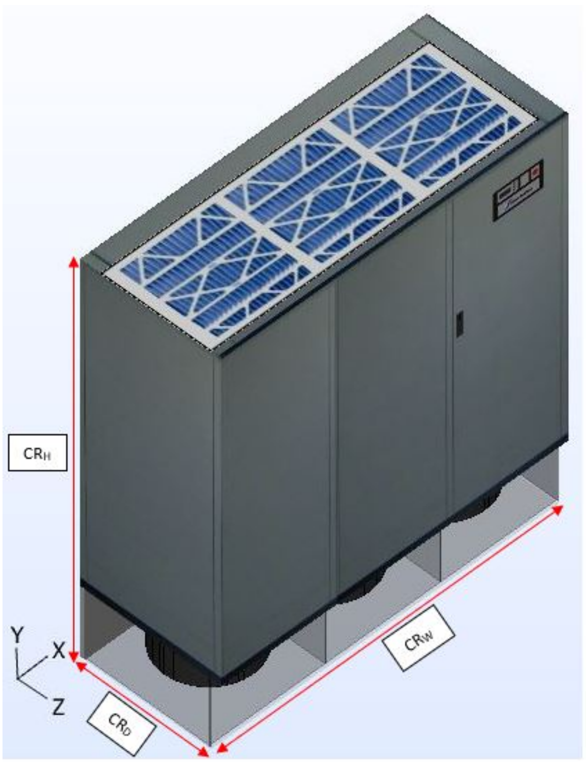
(b) The top view of Hot Aisle / Cold Aisle arrangement.

Figure 4.4: The Hot Aisle (blue) / Cold Aisle (red) arrangement of the cabinets in the data center white space. The cold air emerges from the raised floor through the tiles (present in the Cold Aisle) into the inlet of the servers. The Hot Aisle collects all the hot air from the outlets of the servers and directs it to the CRAHs.

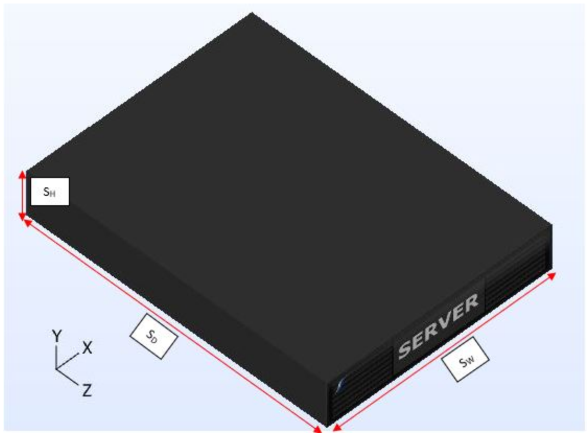
In figure 4.5, we see the standard dimensions of the cabinet, CRAH, server, and floor tile present in the data center white space. These dimensions are determined mainly by the standard components used in the white space. In the white space we study, each cabinet consists of 50 slots of height U ($1U = 0.044m$). The values of these dimensions are presented in the table 4.2. Each floor tile has squared shaped perforations, each of the holes of size $0.035m$.



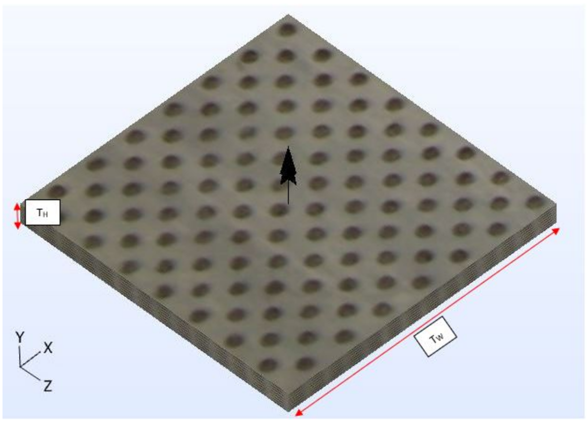
(a) The dimensions of a cabinet.



(b) The dimensions of a CRAH.



(c) The dimensions of a server.



(d) The dimensions of a perforated floor tile.

Figure 4.5: The dimensions of the various components of the white space. The value of these dimensions is shown in the table 4.2

Cabinet Dimensions			Server Dimensions		
Width C_W (m)	Depth C_D (m)	Height C_H (m)	Width S_W (m)	Depth S_D (m)	Height S_H (m)
0.6	1.2	2.4	0.45	0.6	2U (1U = 0.044)
CRAH Dimension			**Floor Tile Dimension**		
Width CR_W (m)	Depth CR_D (m)	Height CR_H (m)	Width T_W (m)	Depth T_D (m)	Height T_H (m)
2.55	1.04	2.4	0.6	0.6	0.041

Table 4.2: The dimensions of the cabinet, server, CRAH and floor tile. These dimensions are constant in all the CFD Simulations.

To create a database of CFD simulations, the combination of the 11 parameters (positions of the 4 perforated plates in the raised floor, amount of perforation in the 4 perforated plates in the raised floor, the height of the raised floor, amount of perforation in the floor tile, and the distance of the CRAH from the first cabinet) presented in the table 4.3, are randomly varied between its lower and upper limit (close bounds). The bounds for these parameters except the position of the perforated plates in the raised floor and its amount of perforation is decided based on industry-standard followed in designing a data center white space[1]. This is further explained in section 4.3.

Parameters		Lower Bound	Upper Bound
Position of Perforated Plates inside the Raised Floor (Z-direction) (m)	Plate 1 (P1)	3.6	5.4
	Plate 2 (P2)	5.4	7.2
	Plate 3 (P3)	7.2	9
	Plate 4 (P4)	9	10.8
Height of the Raised Floor (Y- direction) h (m)		0.8	1
CRAH Distance to the 1^{st} Cabinet of the Row D (m)		1.2	2.4
Amount of Plate Perforation (%)	Plate 1 (P1)	20	100
	Plate 2 (P2)	20	100
	Plate 3 (P3)	20	100
	Plate 4 (P4)	20	100
Floor Tile Perforation (%)		50	100

Table 4.3: The range of the changeable parameters of the white space. A random combination of these parameters would determine the design of the white space, for the CFD simulations. All distances are considered from the origin except the CRAH distance which is considered between the CRAH and the first cabinet of the row.

[1]The bounds for the design parameters were decided based on a discussion with the Lead Mechanical Engineer for Data Center design at RHDHV

4.3. Latin Hypercube Sampling Technique (LHS)

The Latin Hypercube Sampling technique is used to generate the 600 input samples of the 11 design parameters, as shown in table 4.3. These input samples will determine the design of the white space in the CFD simulations. Also, these input samples will constitute the input features of the neural network. The code for LHS is written in Python using the pyDOE[31] library. It is shown in Appendix A.

4.4. Computational Fluid Dynamics (CFD) Simulations of the White Space

4.4.1. Validation of CFD Software

The software to be used for CFD simulations is called 6SigmaRoom (Version R13). Future Facilities[20] has developed this software, which facilitates the creation of 3-Dimensional (3D) data center models and simulate the cooling process in it.

Athavale et al.[42] used the 6SigmaRoom software to experimentally validate a white space CFD model for a raised floor data center having active tiles. Active tiles are perforated floor tiles with integrated local fans, which increases the volume flow rate locally. They found that the average overall difference between the numerical prediction and experimental measurements is less than 4% for the tile flow rate and $1.7°C$ for the server inlet temperature. Alissa et al.[40] used 6SigmaRoom to model the important physical parameters like, for example, raised floor jack hydraulic resistance, the exact location of supply vents, rack detail structure, momentum transfer through the perforated tiles, etc, which affects the accuracy of the simulation. The CFD modeling results were compared with the measurements of the tile flow rate from the floor tiles for different experimental conditions. The numerical and measured tile flow results were in good agreement with each other and had an overall average error of $\sim 3\%$.

Apart from the above two validation studies done by the respective authors, two validation studies have been done in this thesis to validate the software. The following two studies have been chosen for validation study as they involved heat transfer in the domain in turbulent condition. Following the paper on Ventilation of a Room with a Vertical Temperature Gradient[83], a CFD model of forced convection in a cubical room of size $1m*1m*1m$ with a single inlet and a single outlet is analyzed and verified with measurements of the velocities taken in the room along the lines X1 and X2 (see figure 4.6). The lines X1 and X2 are at a different distance from the YZ-Plane. The results were also compared with simulations done in Ansys Fluent 16 by the author (see figure 4.6).

The fluid in the room is air, and the airflow into the room is turbulent. The temperature of the walls and ceiling is kept at $293K$. The floor is kept at a temperature of $308K$. The velocity of the air flowing through the inlet vent is $0.5m/s$. The measurement of velocity is taken along the line X1 and X2. The model is simulated in a steady-state condition using the Standard $k-\epsilon$ model. Buoyancy is included in the model by using Boussinesq approximation. From figure 4.7, it is seen that the numerical results from 6SigmaRoom are in near agreement with the measurements done in the room and also with the results from Ansys Fluent 16. The difference in the numerical results from the two software is due to the different turbulence models used in the software. The RNG $k-\epsilon$ model being used in Fluent by the author to model the turbulent flow, and the Standard $k-\epsilon$ model in used in 6SigmaRoom as this is the default model used by most users in the 6SigmaRoom software. Also, a difference in the solver setting in both the software accounts for the difference in the numerical result. In 6SigmaRoom, the default solver setting is used, which cannot be changed by standard users of the software without special privileges. This is going to be explained in section 4.4.4. The number of grid cells is nearly the same in both the numerical models.

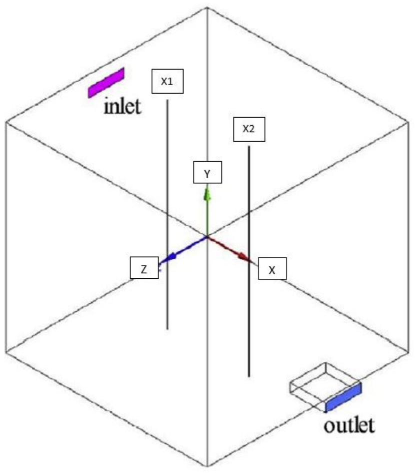
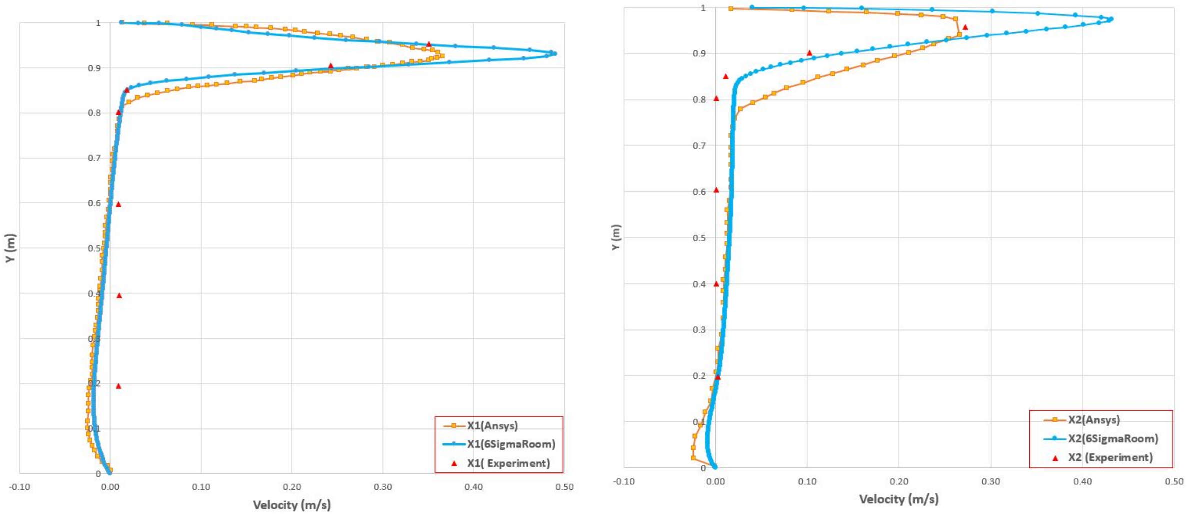


Figure 4.6: The dimensions of the cubical room ($1m * 1m * 1m$). The inlet vent provides an airflow of $0.5m/s$ and the outlet vent extracts an airflow of $0.5m/s$[83].The velocities were measured along line X1 (x = $0.3m$, y = $0 - 1m$, z = $0.5m$) and X2 (x = $0.7m$, y = $0 - 1m$, z = $0.5m$) and compared with the numerical results from Fluent and 6SigmaRoom (see figure 4.7).



(a) X-velocity along the line X1 at location x = $0.3m$, y = $0 - 1m$, z = $0.5m$

(b) X-velocity along line X2 at location x = $0.7m$, y = $0 - 1m$, z = $0.5m$

Figure 4.7: The comparison of simulated and measured X-velocity along the line X1 (sub-figure (a)) and the line X2 (sub-figure (b)). The numerical result from 6SigmaRoom software matches quite well with the result from Ansys Fluent and the measured values. The difference in the numerical results are due to different turbulence model used in respective software. The RNG $k - \epsilon$ turbulence model is used in Fluent by the author[83], and Standard $k - \epsilon$ turbulence model in 6SigmaRoom. Also a difference in the solver settings in both the software accounts for the difference in the numerical result. Given, the limited availability of measurement points in the high velocity gradient zone, it looks like the velocity plot from the 6SigmaRoom gives a better match with the measurements.

The other validation study done in this thesis is on the Turbulent Natural Convection in a Thin Enclosed Channel[43], as shown in figure 4.8. The dimensions of this cavity are $H = 2.18m, W = 0.076m, D = 0.52m$ and the temperature of the cold (left) wall is $15.1°C$ and the hot (right) wall is $34.7°C$. The other walls and ceiling are adiabatic. The property variation of the fluid with temperature is comparatively small. The flow in the core of the cavity is fully turbulent based on the Rayleigh number ($Ra = GrPr = 0.86 * 10^6$), which is calculated based on the width of the cavity and greater than 10^6 for the flow to be in the turbulent regime[45].

The model is simulated in 6SigmaRoom using the steady-state condition and the standard $k-\epsilon$ model. The ideal gas law is used to calculate the density. The temperature and velocity profile is plotted at mid-height ($y = 1.09m$) across the width of the cavity, which is shown in figure 4.9. The temperature and velocity field are almost 2D and do not vary in the z-direction for 95% of the core of the cavity. The simulated results for the velocity and temperature match the experimental values, as shown in figure 4.9.

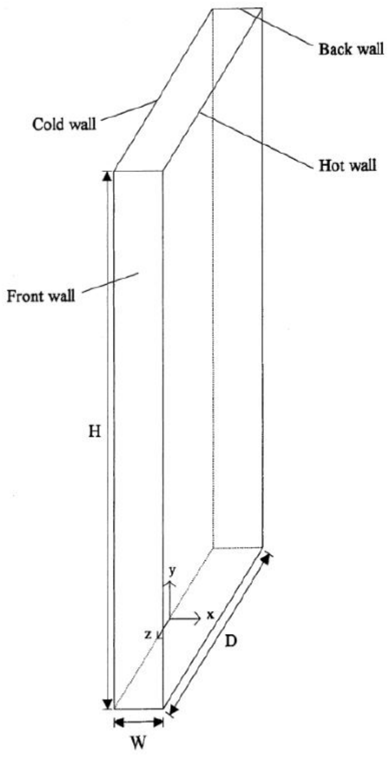
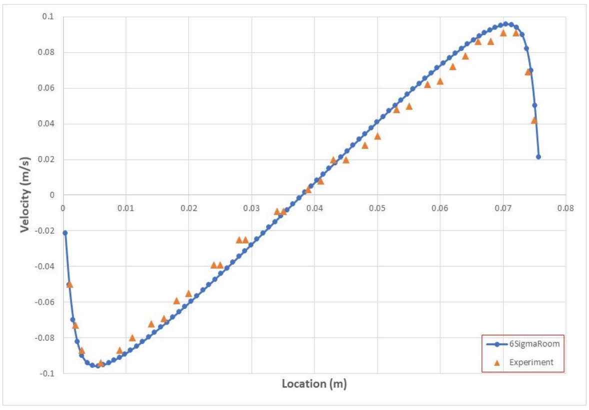
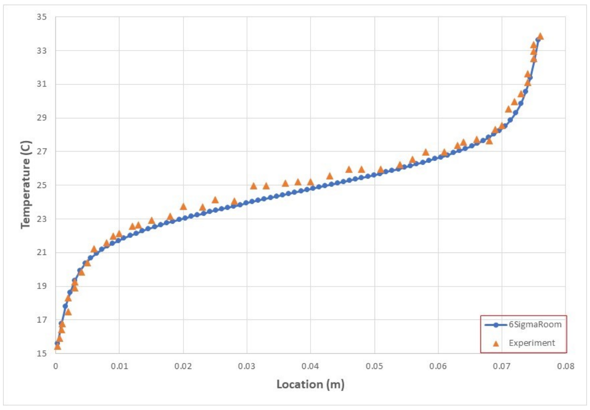


Figure 4.8: The dimensions ($H = 2.18m, W = 0.076m, D = 0.52m$) of the thin enclosed tall cavity. In the test case, the front and back wall are adiabatic. The cold wall is at $15.1°C$ and the hot wall is at $34.7°C$. The flow in the core of the cavity is fully turbulent ($Ra > 10^6$). Property variation of the fluid with temperature is comparatively small.



(a) The velocity profile across the width (W) of the cavity at a height $y = 1.09m$.



(b) The temperature profile across the width (W) of the cavity at a height $y = 1.09m$.

Figure 4.9: The comparison of velocity profile (sub-figure (a)) and temperature profile (sub-figure (b)) along the width (W) of the cavity and at a height ($y = 1.09m$) from the ground. The temperature of the walls is $15.1°C$ and $34.7°C$. The Rayleigh number is $Ra = 0.86 * 10^6$, suggesting a fully turbulent region in the core of the cavity. The temperature and the velocity fields do not vary in the z-direction respectively and can be considered as two dimensional.

Thus, it can be concluded that the 6SigmaRoom software is a well-validated software that can be used for the CFD simulation of the data center white space.

4.4.2. Geometry of the CFD Domain and Meshing

The main components in the simplified data center white space are: the four CRAHs, 90 cabinets, each cabinet containing 25 Servers, 90 perforated floor tiles, and 4 perforated plates placed inside the raised floor. This is shown in figure 4.2a. The geometries are created using the standard objects (IT equipment) available in the software, which populate the domain for the CFD simulation. The dimensions of the white space are ($W = 13.8m, L = 13.8m, H = 5.8m$). The dimensions of the various IT components are shown in the table 4.2. It is assumed there are no cable trays (containing cables), Power Distribution Units (PDUs), and cooling pipes present in the white space to simplify the design. The flow in the servers and CRAHs is not resolved, and they are modeled as "black boxes". The server inlet temperature is calculated by resolving the energy equation 3.3. From equation 4.1, the temperature difference (ΔT) in the server can be found, and thus the outlet temperature of the server can be calculated by adding the temperature difference to the server inlet temperature[2].

$$\Delta T = \frac{Q}{\dot{m} c_p}, \tag{4.1}$$

where Q is the heat dissipated by the server, $\dot{m}$ is the mass flow rate through the server, which is determined by the speed at which the fan in the server is running at, and c_p is the specific heat capacity at constant pressure. The inlet velocity of the server is calculated by solving the RANS equation 3.9. The outlet velocity of the server is calculated from the pressure drop in the server and the inlet velocity. The pressure drop depends on the mass flow rate provided by the fan present in the server.

Modelling of the CRAH is sophisticated as it involves specifying the settings of some internal parameters like mass flow rate of the coolant passing through the CRAH, the temperature of

[2]The inlet temperature of the server is calculated from the energy equation 3.10.

the coolant, maximum sensible cooling curve, airflow control strategy, etc. Most of the options are kept at the default settings.

There is leakage of hot air from the hot aisle through the gaps present in the cabinets. This is assumed to be 5% of the blanked frontal area of the cabinets (see figure 4.10). This means some amount of hot air from the hot aisle would leak into the cold aisle and vice versa. The unrecoverable pressure drop for floor tiles is calculated using the empirical relationships from the Handbook of Hydraulic Resistance[52]. The floor tile is modeled as a thin surface resistance, aligned with the top surface of the tile. According to Future Facilities, "To account for the fact that the tile has many small apertures rather than being one open hole, in addition to the unrecoverable pressure drop, the correct momentum (due to higher velocity) is convected into the cells above the tile. This results in a recoverable pressure drop above the tile that disappears as the jets coalesce, and the jet slows down, or as air is entrained from around the tile by the low pressure. This approach is sometimes referred to as a porous jump model. Another consideration is the impact of the tile on the turbulence in the air stream. Currently, the turbulence in the air is simply convected through the perforated tile, and no additional turbulence is added. To date, there is no research available indicating how turbulence should be addressed". Appropriate boundary conditions are applied ,which is going to be explained further in sub-section 4.4.3.

A structured mesh is generated using the hexahedral element. The meshing is shown in figure 4.12. The computational domain is first divided into rectangular blocks, which are then divided into small hexahedral cells. The aspect ratio of the cells is kept close to one.

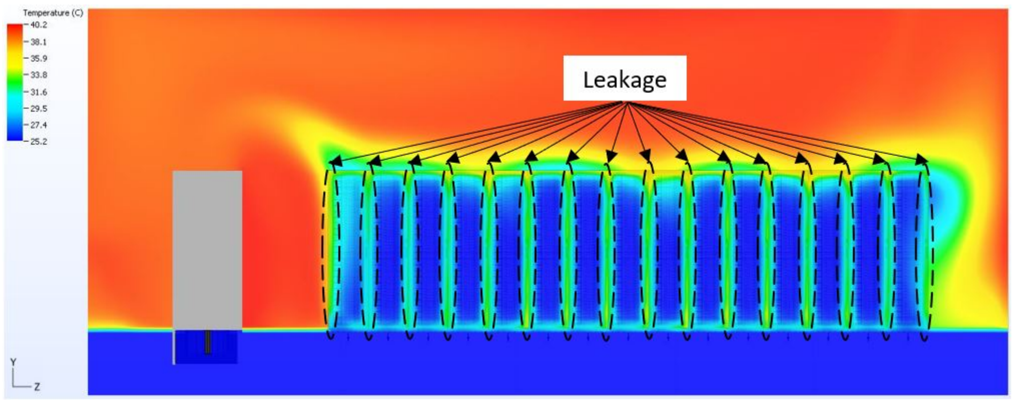


Figure 4.10: The leakage of hot air from the hot aisle to the cold aisle through the side of the cabinets, as seen in this temperature plot taken at the inlet surface of the cabinets in the YZ plane. Due to gaps in the cabinets, the hot air leaks into the cold aisle and mixes with the cold air. This increases the temperature of the supplied cold air and reduces the effectiveness of the supplied cold air provided to the servers. In reality, gaps of 5% of the blanked frontal area of the cabinets are present in the cabinet.

A grid independence study is conducted to ensure that the CFD results are independent of the number of grid cells, as shown in figure 4.11. The termination criteria for the residual is 1. The tile flow rate is considered as the variable for the grid independency test because the temperature in the white space and the servers depends on the tile flow rate provided by the floor tiles. Also, the optimization of the design parameters of the white space is going to be done using the tile flow rate of the floor tiles. The total number of cells required to simulate the flow in the domain is varied from 1.6 Million to 4.8 Million. From the figure 4.11, after 3.0 million cells in the domain, it is seen that the average tile flow rate (average of two adjacent tiles in the X-direction) do not change with the increase in the number of cells in the domain.

Thus it can be concluded that the numerical results become independent from the size of the grid higher than 3.0 million cells.

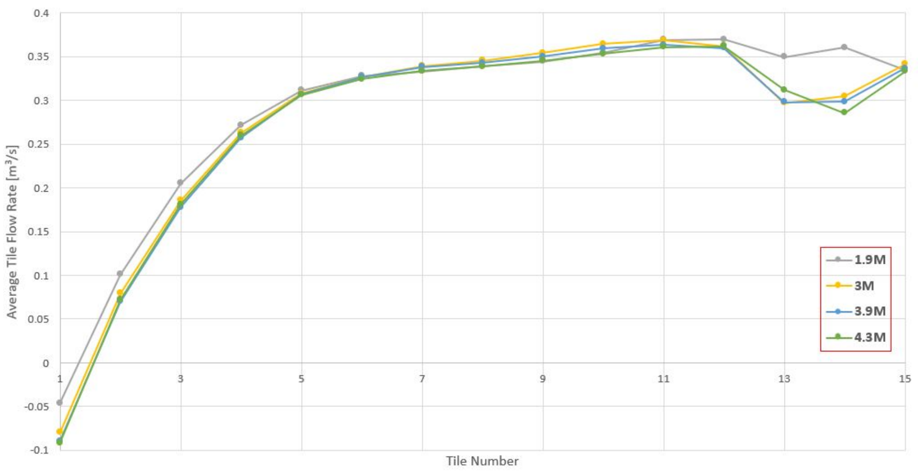
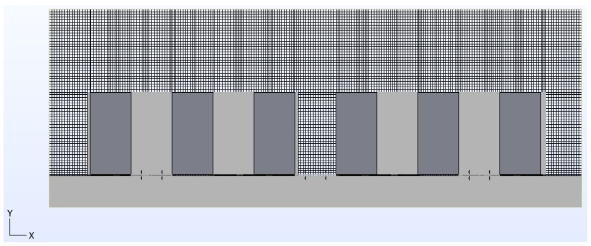
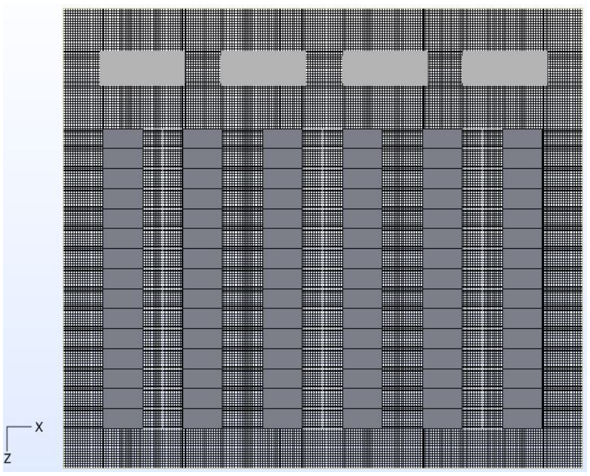


Figure 4.11: The Grid Independency Study to determine the number of cells required to make the numerical solution independent from the size of the grid. The average of the tile flow rate for each tile present in Row X & Row Y (see figure 4.16) is plotted on the y-axis vs. the tile number. Tile number 1 is near the CRAHs and tile number 15 is away from the CRAH. The numerical solution for the tile flow rate becomes Grid Independent after 3.0 million cells.



(a) The meshing of the white space shown along the X-Y plane.



(b) The meshing shown along the X-Z plane.

Figure 4.12: The meshing of the domain of the white space shown along the X-Y plane (sub-figure (a)) and along the X-Z plane (sub-figure(b)). Structured meshing is done using hexahedral elements, because of its simplicity. The CRAHs and cabinets are not meshed, as the flow inside them is not resolved. Thus the CRAHs and cabinets are modeled as "Black Boxes". The maximum size of the cell is $0.05m$ and 3.0 million cells are generated in the domain (after performing the grid independence study).

In a CFD simulation of data center white space, the $k - \epsilon$ turbulence model with standard wall function requires the dimensionless wall distance (y^+) to be within the range 11 to 300[3], i.e., the dimensionless first grid cell height from the wall should be within this range to apply the log-law profile. If the y^+ value is below 11, then the laminar stress-strain relationship is

[3]This is the range of the y^+ value required to apply the log-law wall function in the 6SigmaRoom software. The upper limit of the y^+ is controlled to be below 300 by automatic grid rules implemented in the software.

applied. From figure 4.13, the y^+ value has been plotted in the X-Y plane. It is seen that at the boundary of the solid interface, i.e., the cabinets, raised floor, walls, and ceiling, the y^+ is in the range of 11 to 300 to apply the log-law wall function.

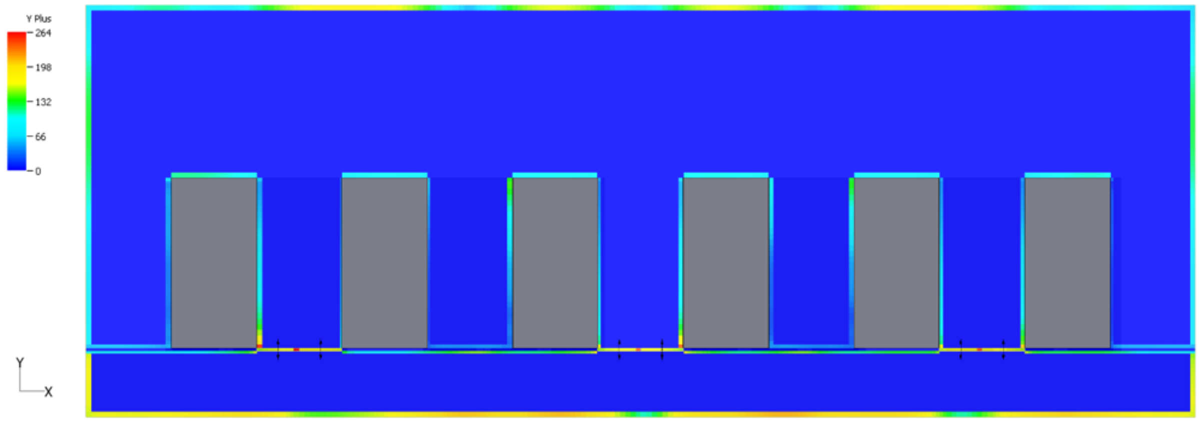


Figure 4.13: The y^+ plotted in the X-Y plane. At the boundary of the cabinets, walls, ceiling, and raised floor, it is seen from the plot that the y^+ is within the range 11 to 300, to apply the log-law profile in order to model the boundary layer.

4.4.3. Boundary Conditions and Assumptions

The floor, walls, and ceiling are adiabatic. It is assumed that there are no cracks/holes through them. Heat loss by radiation is negligible as compared to convection and conduction. The CRAH has two temperature sensors, one of which is placed at its base where the cold air is being supplied and the other at the top of the CRAH where it sucks in the hot air. The CRAHs operate on a return temperature control strategy, as shown in the figure 4.14. It supplies the cold air through the fans present at the base of the CRAH at $25°C$ and is programmed to have a maximum return temperature of $37°C$ while providing a maximum cooling power of $150kW$. The cooling power provided by the CRAH is adjusted based on the return temperature sensed by the CRAH, and the mass flow rate of the cold air provided by the CRAH. The boundary condition of the inlet and outlet of the CRAH is at variable flow condition (depending on the return temperature sensed by the CRAH), i.e., it is a velocity inlet and velocity outlet boundary condition. The mass flow rate of the supplied cold air can vary, and it can provide a maximum flow rate of $10.4m^3/s$. This can be controlled by changing the fan (placed at the bottom of the CRAH) speed of the CRAH. The fans can consume a maximum power of $6.6kW$. The coolant used in the CRAH is chilled water, which runs through it at $20°C$. Technically, the CRAH is programmed to provide a maximum cooling power of $150kW$ at $37°C$, but it can operate beyond this limit for safety reasons, i.e., to provide more cooling power to the white space if it heats up further. But this trade-off comes at a price of smaller operational lifetime of the CRAH.

The flow rate of the server is controlled by the EnergyStar Flow Rate Ratio Curve, as shown in figure 4.15. It is seen that the flow rate through the server remains constant till $27°C$, and then it shoots up linearly as the temperature increases beyond $27°C$ and then reaches a constant value of $100l/s/kW$.

A server acts as a heat source and dissipates $0.266kW$ of heat. This value has been calculated based on the heat dissipation per unit floor area of $2.78kW/m^2$ (based on industry standards[4])

[4]This value for the heat dissipation per unit floor area was decided based on a discussion with the Lead Mechanical Engineer for Data Center design at RHDHV

in the white space of the data center.

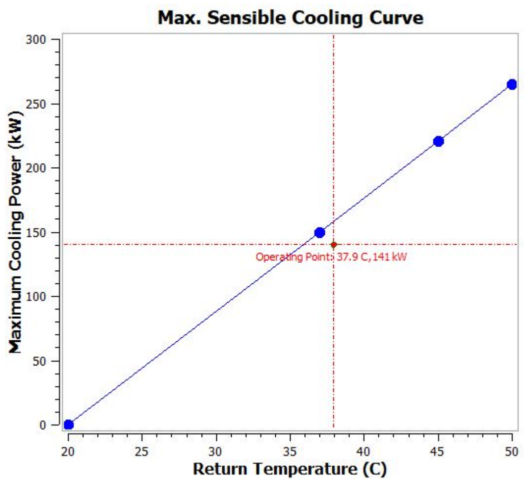


Figure 4.14: The maximum sensible cooling curve of the CRAH. It shows the max. cooling power provided by the CRAH vs. the return temperature ($°C$) of the CRAH. The return temperature is measured by the return temperature sensor placed on the top of the CRAH. The CRAH is programmed to have a maximum return temperature of $37°C$. The cooling power provided by the CRAH is adjusted based on the return temperature sensed by the CRAH and the mass flow rate of the cold air provided by the CRAH. The blue dots indicate the maximum cooling power provided by the CRAH at certain return temperatures. In this figure, the CRAH is operating at $141kW, 37.9°C$ (shown by the red dot). The red dot can move anywhere in the region below the blue curve. If it reaches the region above the blue curve, it is going to violate the CRAH control strategy.

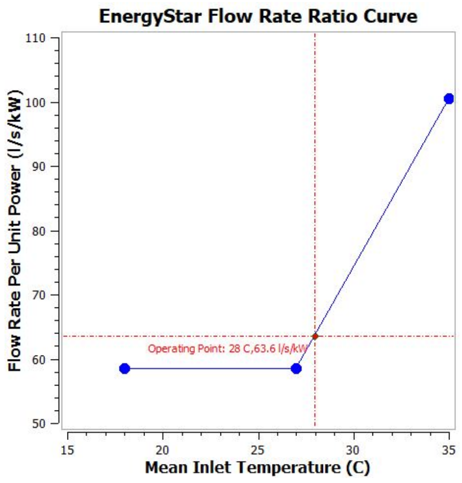


Figure 4.15: The EnergyStar flow rate ratio curve. It shows the flow rate per unit power (l/s/kW) consumed by a server vs. the mean inlet temperature ($°C$) of the server. This determines the flow rate of the air going through the server. The flow rate remains constant at $58l/s/kW$ until mean inlet temperature reaches $27°C$ and then it shoots up linearly as the temperature increases beyond $27°C$, and then reaches a constant value of $100l/s/kW$. In this figure, the server is operating at a mean inlet temperature of $28°C$, and the flow rate of $63.6l/s$ per unit heat dissipated by the server. The blue points indicate the flow rate per unit power provided by the server at the particular mean inlet temperature.

4.4.4. Setup of the CFD Simulation

To confirm if the flow in the white space is in the turbulent region, the average Reynolds number has been calculated based on the floor tile pore size and velocity of airflow through the tile. This is shown in the equation 4.2:

$$Re_D = \frac{\rho_{air} U_{pore} D}{\mu}, \tag{4.2}$$

where U_{pore} is the velocity through the pores of the tiles, D is the pore size. Based on the average velocity of $1.3m/s$ (found from simulations) through the pore and pore diameter of $0.035m$, the Reynolds number turns out to be around 3000 which is more than the critical Reynolds Number ($Re_{cr} = 1000$), for the flow to be in the turbulent regime as suggested by Miller[61]. Thus it can be stated that the flow in the white space is in the turbulent regime.

To determine the type of heat transfer, the Richardson Number (Ri) gives the relative importance of extend of natural convection, mixed convection, and forced convection. It is shown in the equation 4.3:

$$Ri = \frac{g\beta(T_{hot} - T_{bulk})L}{V^2} = \frac{Gr}{Re^2}, \tag{4.3}$$

where β is the thermal expansion coefficient, L the characteristic length, V the characteristic velocity, T_{hot} is the average temperature of the outlet of all the servers, and T_{bulk} is the bulk temperature in the white space (excluding the cold aisle). Based on a characteristic velocity of $1.3m/s$ (velocity of airflow from the floor tiles) and characteristic length scale of $2.4m$ (height of the cabinet), temperature difference of $8.1°C$, and $\beta = 0.003K^{-1}$ the Ri in the white space is around 0.33. This indicates that the convection in the room is mixed convection, mainly dominated by forced convection, as suggested my Mills(1992)[62].

The buoyancy effect is included in the momentum equation by using the Boussinesq approximation as the term $-\beta(T - T_o) << 1$, where T is the temperature in a region and T_o the reference temperature, for all the CFD simulation of the white space.

The CFD simulation was made to run in steady-state condition, and turbulence is modeled using the Standard $k - \epsilon$ model, and Standard Wall Function is used to model the boundary layers. The maximum number of iterations is set to 1000. The Hybrid Cells option is selected[5]. The grid refinement ratio limit is set to 2. The energy equation is solved by selecting the Algebraic MultiGrid (AMG) structured linear solver[68]. The buoyancy term in the RANS equation is modeled using the boussinesq approximation. The default discretization schemes specified in section 3.3 is used. The use of 1^{st} order upwind scheme for space discretization is justified by the grid independency test, as the numerical diffusion is minimized. The termination strategy for a simulation is set using the "residuals only" option. The residual termination criteria is based on a termination factor. A termination factor of 1 indicates that the governing equations have converged.

The reference density of air is set to $1.19kg/m^3$, laminar viscosity to a constant value of $1.8 * 10^{-5}kg/ms$, the conductivity of air to a constant value of $0.026W/m - K$, specific heat to a constant value of $1005J/kg - K$, and expansivity of air to $0.0033C^{-1}$. The reference pressure is $1.01 * 10^5Pa$.

[5]A computational cell can be marked as solid, fluid, or hybrid. A cell is considered solid if its center is within the solid object, and fluid if its center is outside the solid object. A cell which contains both solid and fluid properties are marked as hybrid[68]

4.5. Prediction of the CFD Results using an Artificial Neural Network (ANN)

In this thesis, four separate neural networks are constructed. The input of each of these neural networks consists of 11 parameters as described in the table 4.3. These input neurons are called the input features. The input features are scaled between 0 and 1. Data scaling of the input features is an essential pre-processing step before starting to train the neural network. It speeds up the training process and improves the accuracy of the network. The data (input features) obtained from the Latin Hypercube Sampling (LHS) method contains unscaled values as each input feature has a different range of values. These values are scaled in the range of 0 to 1 before the training of the network starts. The scaling of the input features is shown in equation 4.4

$$X_{scaled} = \frac{X - \overline{X}.min}{\overline{X}.max - \overline{X}.min}, \tag{4.4}$$

where $\overline{X}$ is one of the unscaled input feature, X_{scaled} is the scaled variable of the input feature, X is the corresponding unscaled variable, $\overline{X}.min$ is the minimum value of the unscaled input feature, and $\overline{X}.max$ is the maximum value of the unscaled input feature. The output layer of each neural network consists of 15 neurons. The 4 neural networks are used to predict the 4 rows highlighted in the figure 4.16. Each row consists of 15 IT components, i.e., row X and row Y consist of 15 floor tiles each and row W and row Z consist of 15 cabinets each. Only the middle row of cabinets and floor tiles are being considered for prediction by the neural network because predicting the temperature of all the cabinets and tile flow rate of all the floor tiles present in the white space would increase the complexity of the neural network. It would require more training time to reach a generalized[6] solution and the neural network is going to require a large amount of training data. Also, the flow in the white space is not symmetrical (will be explained in section 5.2). Therefore, it is not possible to replace the other rows of cabinets and floor tiles with a symmetry boundary condition in the YZ plane. Thus the other rows of cabinets and floor tiles are put into the domain of the simulation, beside the rows highlighted. The values for the input feature are obtained from the Latin Hypercube Sampling Method, and the corresponding output (tile flow rate and cabinet temperature) of these input features are obtained from generating 600 CFD simulations of the white space.

In order to reduce the complexity in designing the neural networks, 4 separate neural networks are used to predict each of the 4 rows, instead of 1 single neural network to predict all the variables. Every row (row W/X/Y/Z) is assigned a neural network to predict its output variable, i.e., the tile flow rate of the floor tiles in row X and row Y respectively and maximum cabinet inlet temperature of the cabinets in row W and row Z respectively. Since the output of these neural networks consist of continuous real values (i.e., floating numbers) and not discrete value (i.e., ones and zeros), this neural network falls under the category of regression and not classification.

[6]A neural network is said to be generalized if it is able to predict accurately on new unseen data, which is drawn from the same distribution used to train the neural network.

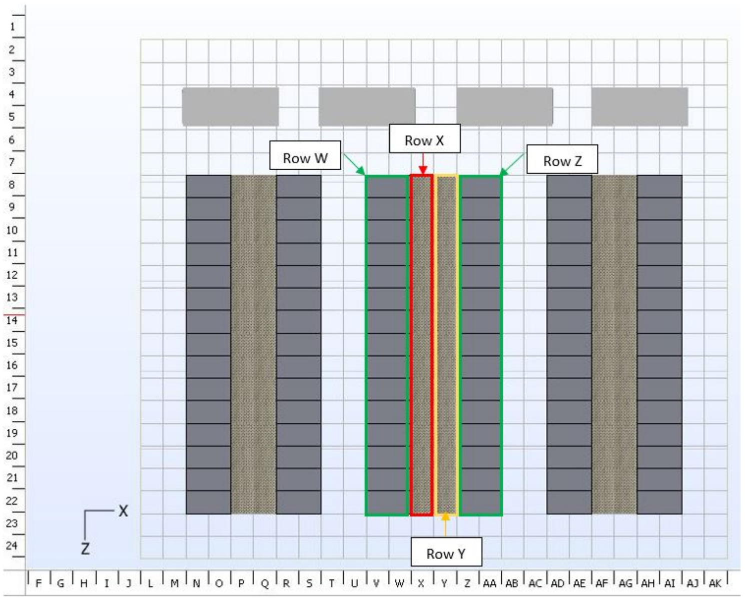


Figure 4.16: The top view of the data center white space. The rows highlighted in green (row W and row Z), are the cabinets (15 cabinets in each row). The row highlighted in red (row X), and yellow (row Y) are the floor tiles (15 in each row). These rows are going to provide the output data, which the neural networks are going to predict.

A dataset, consisting of the 11 input features and the ground truth values (tile flow rate and max. cabinet inlet temperature) is created. Before the neural networks start training on the dataset, the hyper-parameters of the neural network have to be specified. Mean Squared Error (see equation 3.21) is used for all the four neural networks to calculate the prediction error, as it is easier to compute the gradient of the error function of the neural network as compared to using MAE (see equation 3.22). The train test split used here is $80\% - 20\%$ as the dataset size is limited to 600 samples. LeakyRelu[26, 33] is used as an activation function for the input, and the hidden layers (as it does not suffer from the vanishing gradient problem) and Linear Activation function is used for the output layer.

There are many ways to find the hyper-parameters of a neural network. They are Hit and Trial Method (Manual Search), Grid Search, Random Search, Bayesian Optimization (HyperOpt), and Genetic Algorithm[23, 51]. Using the Keras library[25], the structure of the neural network is created. A genetic algorithm is used to find the optimum hyper-parameters of the neural network to achieve a high testing accuracy. The fitness function of the genetic algorithm is set to the testing accuracy of the neural network, which has to be maximized. The crossover probability and the mutation probability is set to 0.25 and 0.05 respectively. The population size is kept to 20 neural network model, and the maximum number of generations is 10. The termination criteria for the genetic algorithm is triggered when the testing accuracy is more than a R^2 of 0.99. Two genetic algorithm models are created, one to find the hyper-parameters of the neural network predicting the tile flow rate and the other to find the hyper-parameters of the neural network predicting the cabinet temperature. The python code for finding the hyper-parameters using the genetic algorithm is shown in Appendix B and C. After the hyper-parameters are found for the respective neural networks, the neural networks are trained on the training dataset to predict the tile flow rate of row X and row Y and the maximum cabinet inlet temperature of row W and row Z respectively. This is shown in Appendix D, E, F, and G respectively.

4.6. Optimization of Design Parameters of the white space using NSGA-II (Genetic Algorithm)

Each server requires an airflow rate of approximately $60l/s/kW$, to maintain its mean inlet temperature below $27°C$ (see figure 4.15). In each cabinet (in the white space of the CFD simulation), there are 25 servers present in it. Thus, in order to maintain the mean inlet temperature of all the servers present in a cabinet below $27°C$, a cabinet would require at-least $1500l/s/kW$ (see equation 4.5) of cold air (from $25°C$ to $27°C$) at its inlet. Each server dissipates $0.266kW$ of heat. Thus each cabinet would then require $0.4m^3/s$ of cold air at its inlet surface (see equation 4.6 for calculation).

$$\begin{aligned}\tilde{v}_c = \tilde{v}_s * n_s &= 60\frac{l/s}{kW} * 25 \\ &= 1500\frac{l/s}{kW},\end{aligned} \tag{4.5}$$

$$\begin{aligned}\dot{v}_c &= \tilde{v}_c * Q * 0.001\frac{m^3/s}{l/s} \\ &= 1500\frac{l/s}{kW} * 0.266kW * 0.001\frac{m^3/s}{l/s} \\ &= 0.4\frac{m^3}{s},\end{aligned} \tag{4.6}$$

where, $\tilde{v}_c$ is the flow-rate rate required by a cabinet per unit heat dissipated, $\tilde{v}_s$ is the flow-rate required by a server per unit heat dissipated, n_s is the number of servers in a cabinet, and Q is the heat dissipated by a server.

In this study, we focus on 30 cabinets, i.e. 15 in row W and 15 in row Z (see figure 4.16). Each cabinet requires a total airflow of $0.4m^3/s$, to maintain the mean inlet temperature of all the servers present in it below $27°C$. Each of the cabinets (in consideration) requires an airflow rate of $0.4m^3/s$. Thus, the distribution of this airflow rate is uniform. The average value of this airflow distribution is $0.4m^3/s$, and the standard deviation of this distribution is $0m^3/s$ (as the distribution is uniform). For two distributions to be the same, their average and standard deviation should be the same. The 30 floor tiles present in row X and row Y should be able to provide the airflow required by all the 30 cabinets present (15 in row W and 15 row Z), in order to maintain the mean inlet temperature of the servers present in a cabinet below $27°C$. In other words, the average and the standard deviation of the tile flow rate distribution of the 30 floor tiles, i.e., 15 in row X and 15 in row Y (see figure 4.16), has to match with the average and standard deviation of the airflow rate distribution, required by the 30 cabinets. This leads to a multi-objective optimization problem. The objective functions are presented in equation 4.7 and equation 4.8:

$$Objective_1 = |\overline{\dot{v}_t} - \overline{\dot{v}_c}|, \tag{4.7}$$

$$Objective_2 = |\sigma_{\dot{v}_t} - \sigma_{\dot{v}_c}|, \tag{4.8}$$

where, $\overline{\dot{v}_t}$ is the average of the tile flow rate distribution, $\overline{\dot{v}_c}$ is the average of the cabinet airflow rate distribution, $\sigma_{\dot{v}_t}$ is the standard deviation of the tile flow rate distribution, and $\sigma_{\dot{v}_c}$ is the standard deviation of the cabinet airflow rate distribution. In both objectives, the absolute difference is calculated. In order to match both the distributions, objective_1 and objective_2 have to be minimized to zero simultaneously. In other words, the difference between the averages of these two distributions, and the difference between the standard deviations of these two distributions have to be reduced to zero simultaneously. This is going to require a multi-objective optimization algorithm to solve it.

The average and the standard deviation of the tile flow rate distribution of the 30 floor tiles (15 in row X and 15 in row Y) are calculated from the neural networks predicting the tile flow rates. The average and the standard deviation of the ideal airflow rate distribution of the 30 cabinets (15 in row W and 15 in row Z) is known. The absolute difference of the average and standard deviation, respectively, of the two distributions (tile flow rate distribution and cabinet airflow requirement distribution) are found. This is done multiple number of times for several runs of the neural network. This then forms the solution space for the NSGA-II algorithm. The absolute difference of the average and the absolute difference of the standard deviation respectively have to be minimized to zero, which then forms the fitness (objective) function. Initially, a parent population size of 2000 and an offspring population size of 1000 is formed by running the neural networks for the tile flow rate 3000 number of times. The crossover probability is set to 0.8, and the mutation probability is set to 0.01. The maximum number of generations until which the NSGA-II algorithm is run is till 500.

The NSGA-II algorithm gives us the optimized values of the 11 design variables, namely positions of four perforated plates, amount of perforation of these four perforated plates, tile perforation, raised floor height, and CRAH distance. These optimized values of the design variables are going to ensure that a maximum number of servers have their mean inlet temperature below $27°C$. The python code for the implementation of the NSGA-II algorithm is applied using the DEAP Library[18]. This is shown in Appendix H.

5 Results

5.1. Residuals of the CFD Simulation

The residual of the temperature, turbulent kinetic energy (KE), turbulent dissipation rate (EP), X velocity, Y velocity, Z velocity, and pressure equation in the CFD simulation of the white space is shown in the figure 5.1. The residual[1] is plotted on the vertical axis and the no. of iterations on the horizontal axis. A residual of 1 indicates that the termination criteria has been achieved, meaning the sum of the residual errors for the variable has fallen below the termination criteria, which by default is the 10^{-3} times the incoming mass, momentum, energy, etc. as appropriate. A user-set termination factor is multiplied with the termination criteria to give the user control. It is seen that the residual decreases with an increase in the number of iterations. The CFD simulation of the white space continues until the residual for the temperature, turbulent kinetic energy, turbulent dissipation rate, X velocity, Y velocity, Z velocity, and pressure reaches 1, as shown in the figure 5.1. The CFD solution converges before reaching the maximum iteration number of 1000. Reducing the termination factor to 0.5 changes the tile flow rate of the floor tiles by less than 1% of the original value. This is not a significant change. Keeping the termination criteria to a termination factor of 0.5 requires more computational time for the solutions to converge.

[1]The vertical axis should be labeled as normalized residual errors. It is labeled as residuals purely for the simplicity for most users. This information is provided by Future Facilities

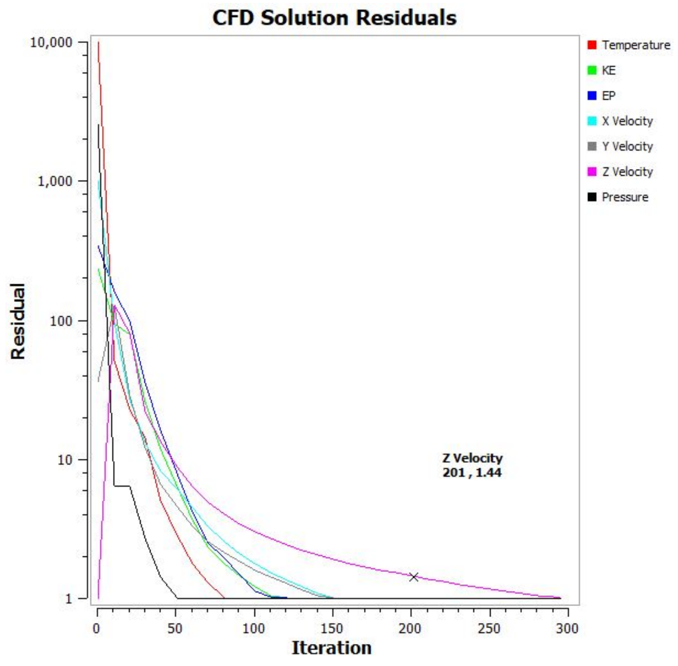


Figure 5.1: The plot of the residual (the vertical axis should be labeled as normalized residual) of the CFD simulation on the vertical axis vs. the number of iterations on the horizontal axis. The termination factor is multiplied with the default termination criteria. A residual of 1 indicates that the termination criteria has been achieved, i.e., the sum of the residual errors for the variable has fallen below the termination criteria, which by default is the 10^{-3} times the incoming mass, momentum, energy, etc. as appropriate. The cross (x) on the pink curve is indicating that the residual for Z velocity at 201^{th} iteration is 1.44.

5.2. Non-Symmetrical Flow Domain

If we look at a server from the inlet side, i.e., in the direction of the red arrow, as shown in the figure 5.2, the outlet of the server is located towards the left side of the server. Each server has a fan located internally, which regulates the airflow through it. This can be verified using the velocity plot in the XZ plane, as shown in the figure 5.2. In this figure, we focus our attention on the middle two rows of cabinets (highlighted by the yellow dotted box). The inlet of all the servers is facing the cold aisle. In this yellow box, the row of cabinets present on the right (highlighted by the black dotted box) contains the servers expelling their heat from their left-back portion (indicated by the red arrows which are shifted towards the CRAH). Similarly, the row of cabinets present on the left (highlighted by the white dotted box) contains the servers expelling their heat from their left-back portion too (indicated by the red arrows which are shifted away from the CRAH). This can also be verified through the magnitude of the velocity plot at the outlet of the server, as seen in figure 5.2. At the outlet of the server, the magnitude of the velocity is relatively higher towards the left side of the server (if we look in the direction of the red arrow) as compared to the right side of the server. Similar phenomena are taking place in the other rows present in the white space. From this, we can conclude that although there is a geometrical symmetry in the YZ plane passing through the middle of the yellow box, the flow is asymmetrical along this plane. Thus a symmetrical boundary condition cannot be applied to simulate the flows in the data center. The flow in a server is not resolved (see section 4.4.2). The outlet of the server is shifted towards the left side (as viewed from the inlet of the server). Thus the velocity of air is higher near the left side of the server (as indicated by the red arrow in figure 5.2). This location of the outlet vents of the server causes a non-symmetrical flow in the data center white space.

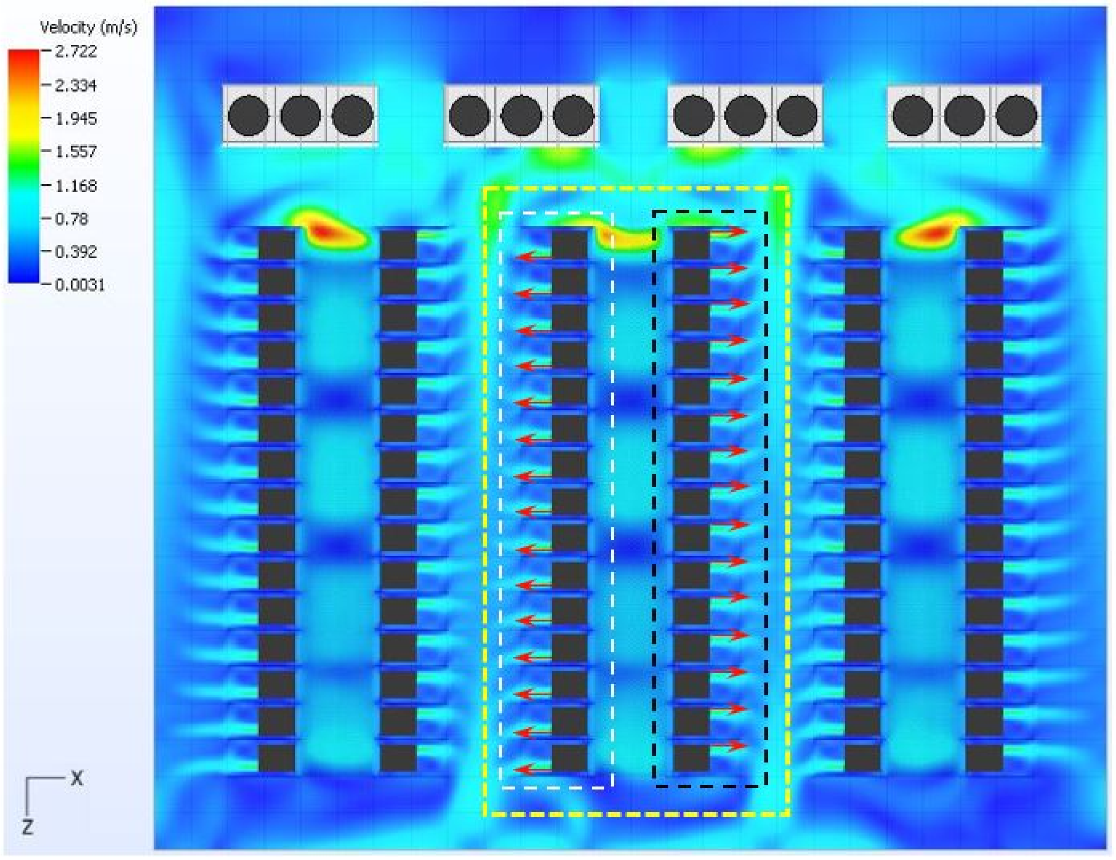


Figure 5.2: The demonstration of the non-symmetrical flow field shown by the velocity plot in the X-Z plane taken at the mid-height of the cabinet. The server's inlet are facing the cold aisle. In the rows present in the middle of the white space (highlighted in yellow), the row of cabinets present on the right (highlighted by the black), contains the servers expelling theirs heat from their left-back portion (indicated by the red arrows which are shifted towards the CRAH). Similarly, for the row of cabinets present on the left (highlighted by the white dotted box) it contains the servers which are expelling their heat from their left-back portion too (indicated by the red arrows which are shifted away from the CRAH). A similar flow field can be seen in the other rows present in the white space. Thus there is geometrical symmetry along the plane (parallel to the YZ Plane) and passing through the middle of the yellow box, but the flow symmetry is not present along the XZ plane.

5.3. Neural Network Predictions

The results of scaling the input features are shown in figure 5.3. In this figure, the frequency of occurrence of the variable is shown on the y-axis and the value of the input feature on the x-axis. From this figure, the 11 input features are almost uniformly distributed (see figure 5.3), which indicates that there is no bias in the range of these input features.

The output variables to be predicted by the neural network are obtained from the CFD simulations. They consist of the tile flow rate of the floor tiles present in row X and row Y, respectively, and the maximum cabinet inlet temperature of the cabinets present in row W and row Z (see figure 4.16). The output variables are unscaled as the neural network does not require them to be scaled. The distribution of these variables is shown in figure 5.4.

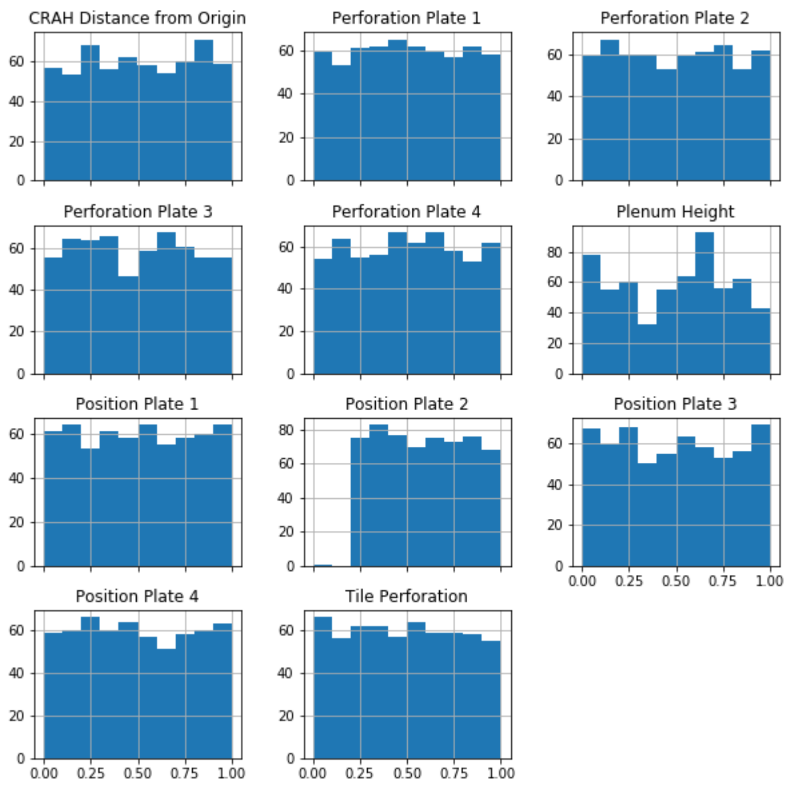
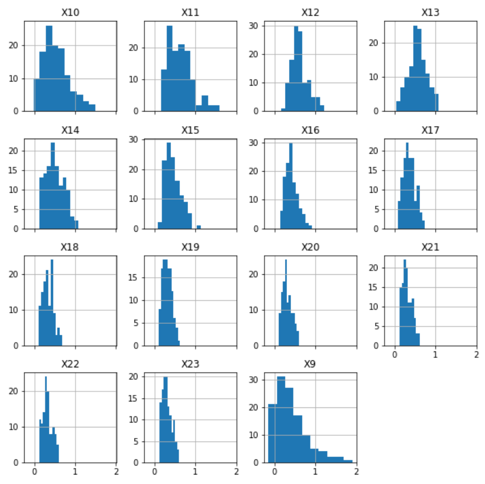


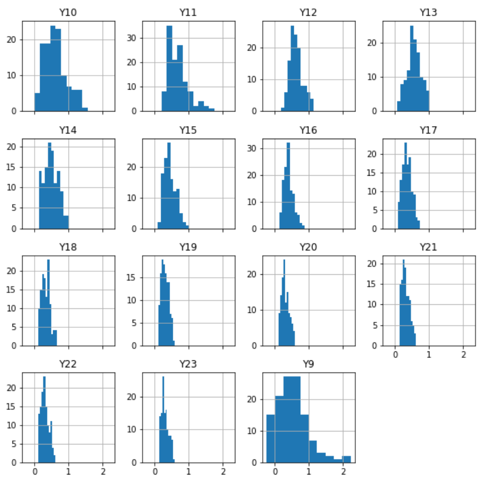
Figure 5.3: The distribution of the 11 input features of the neural network. The plot for each feature shows the frequency of occurrence on the Y-axis and the value of the feature on the X-axis. The input features have been scaled in the range of 0 to 1. The input features show a uniform distribution.

The genetic algorithm is applied to find the optimal set of hyper-parameters of the neural network predicting the tile flow rate of row X. Then the neural network is trained to predict the tile flow rate of row X. This same set of hyper-parameters are used for training the neural network to predict the tile flow rate of row Y. This is done because the distribution of the tile flow rate in row X and row Y is nearly the same. Similarly, the genetic algorithm is applied to find the optimal set of hyper-parameters of the neural network predicting the max. cabinet inlet temperature of row W. These hyper-parameters are then used to train the neural networks. predicting variables in row W and row Z respectively, as their distribution is also nearly the same. The optimal set of hyper-parameters are shown in table 5.1. The neural networks runs for 10000 epochs.

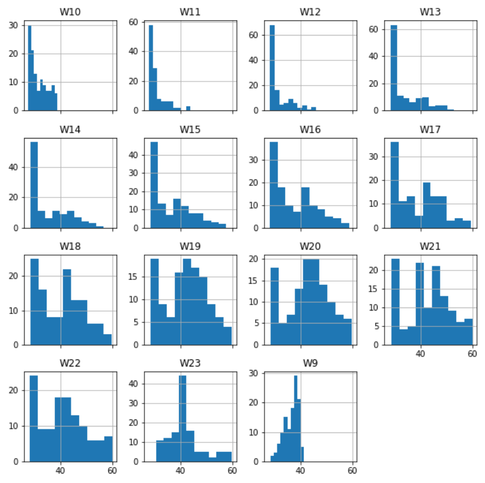
Using the input features obtained from the LHS method and its corresponding output from the 600 CFD simulations, these values are fed to the neural network to predict the output values (tile flow rate and maximum cabinet inlet temperature). The whole dataset is split into the training and testing dataset in the ratio of 4 : 1.



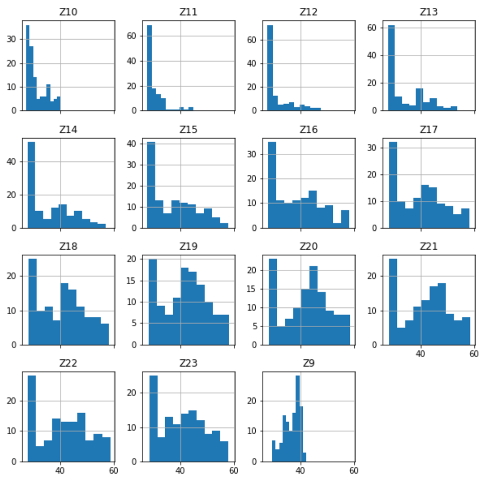
(a) The ground truth (actual values) of tile flow rate (Row X).



(b) The ground truth (actual values) of tile flow rate (Row Y).



(c) The ground truth (actual values) of maximum cabinet inlet temperature (Row W).



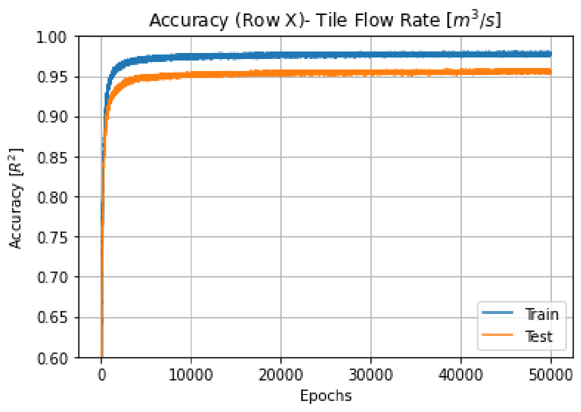
(d) The ground truth (actual values) of maximum cabinet inlet temperature (Row Z).

Figure 5.4: The ground truth (actual values) obtained from the CFD simulations, corresponding to the input features obtained from LHS. The distribution of the tile flow rate of row X and row Y are shown in sub-figure (a) and (b) respectively. The distribution of the maximum cabinet inlet temperature of row W and row Z are shown in sub-figure (c) and (d). The variables are unscaled as the neural network do not require the output variables to be scaled. There are no outliers present in the distribution.

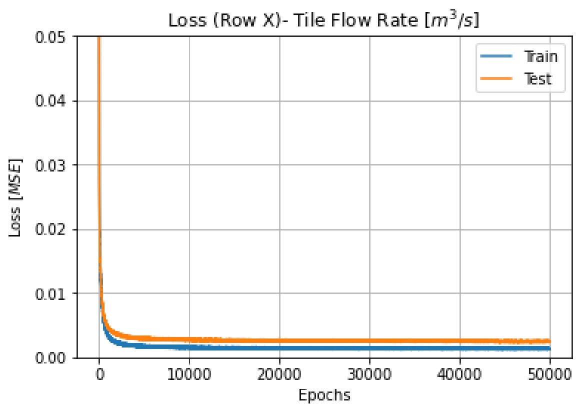
The training and testing R^2 accuracy (eq. 3.23) and the corresponding MSE (eq. 3.21) loss of the neural networks predicting the tile flow rate of the floor tiles present in row X and row Y is shown in the figure 5.5. It is observed in this figure that the R^2 accuracy of the networks increases rapidly and then becomes asymptotic to 1. Similarly, the MSE loss decreases rapidly and then becomes asymptotic to 0. There is no overfitting and underfitting of the training data as the testing accuracy/loss matches the training accuracy/loss almost completely. Thus the neural nets are well generalized[2].

Hyperparameters	Row X & Row Y	Row W & Row Z
Batch Size	128	128
Learning Rate	0.0005	0.0005
No. of Hidden Layers	2	3
No. of Neurons per Hidden Layers	236, and 188	236, 250, and 300
Dropout	0.297 and 0.005	0.297, 0.1, and 0.1

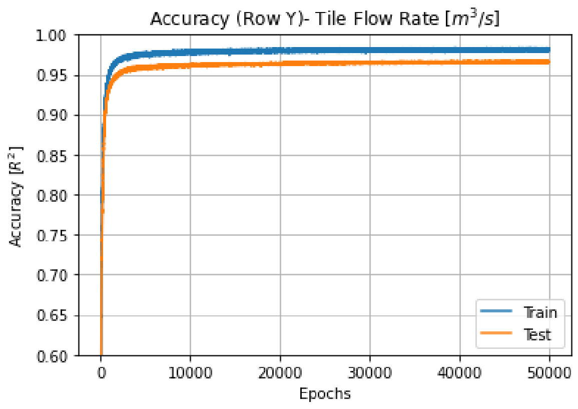
Table 5.1: The hyper-parameters used by the neural networks to predict the output variables. The neural networks predicting the tile flow rate of the floor tile present in row X, and row Y uses the same set of hyper-parameters. This is done as the distribution of the variable in row X and row Y are nearly similar. Similarly, the neural networks predicting the max. cabinet inlet temperature of the cabinets present in row W and row Z is using another similar set of hyper-parameters



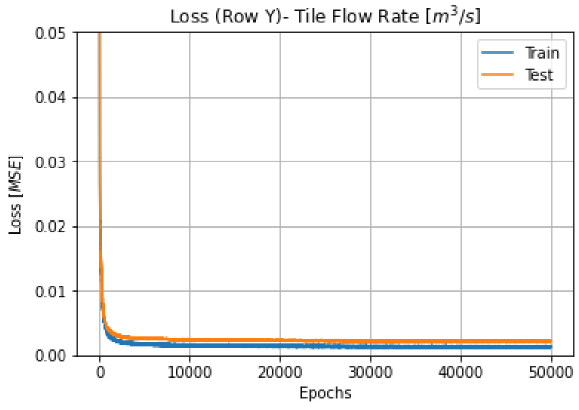
(a) The R^2 accuracy of the tile flow rate prediction (Row X).



(b) The MSE loss of the tile flow rate prediction (Row X).



(c) The R^2 accuracy of the tile flow rate prediction (Row Y).



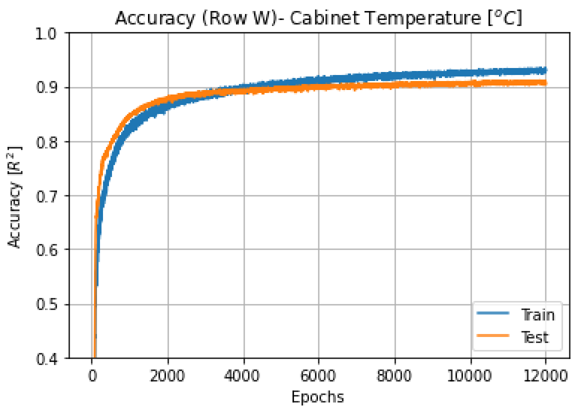
(d) The MSE loss of the tile flow rate prediction (Row Y).

Figure 5.5: The performance of the neural networks predicting the tile flow rate of row X and row Y respectively. The R^2 accuracy and MSE loss is plotted for row X in sub-figure (a) and (b) respectively and row Y in sub-figure (c) and (d) respectively. From the graph, it is seen that the training and testing accuracy and its loss are almost equal to each other. The neural nets are well generalized. The training of the networks can be stopped at 50000 epoch as there is no significant improvement in the performance henceforth. The R^2 accuracy reaches almost 1.

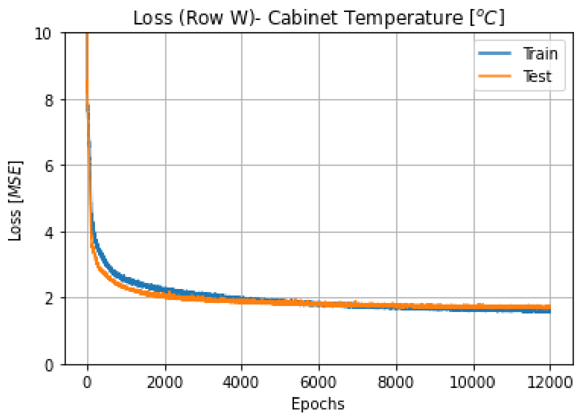
The training and testing R^2 accuracy and the corresponding MSE loss for the neural networks

[2]A neural network is said to be generalized if it can predict accurately on new unseen data, which is drawn from the same distribution used to train the neural network.

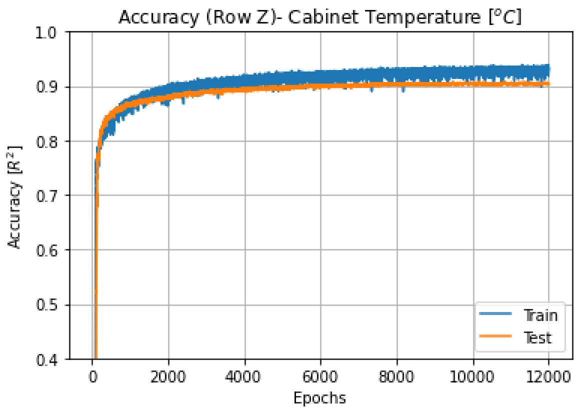
predicting the maximum cabinet inlet temperature of the cabinets present in row W and row Z are shown in the figure 5.6. It is observed in this figure that the R^2 accuracy of the networks increases rapidly, becomes asymptotic near to 1 and then starts to overfit. Similarly, the MSE loss decreases rapidly and then becomes asymptotic closer to 0. There is overfitting of the training data after 12000 epochs as the testing accuracy starts to diverge after 12000 epochs. The possible cause of this overfitting is due to the limited size of the training dataset. The training of the model should be stopped at 12000 iterations.



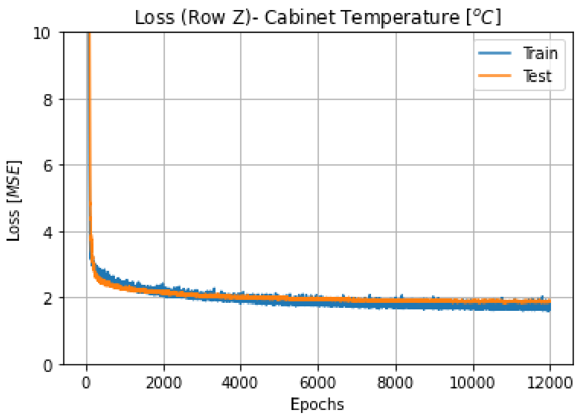
(a) R^2 Accuracy of maximum cabinet inlet temperature (Row W)



(b) MSE Loss of maximum cabinet inlet temperature (Row W)



(c) R^2 Accuracy of maximum cabinet inlet temperature (Row Z)

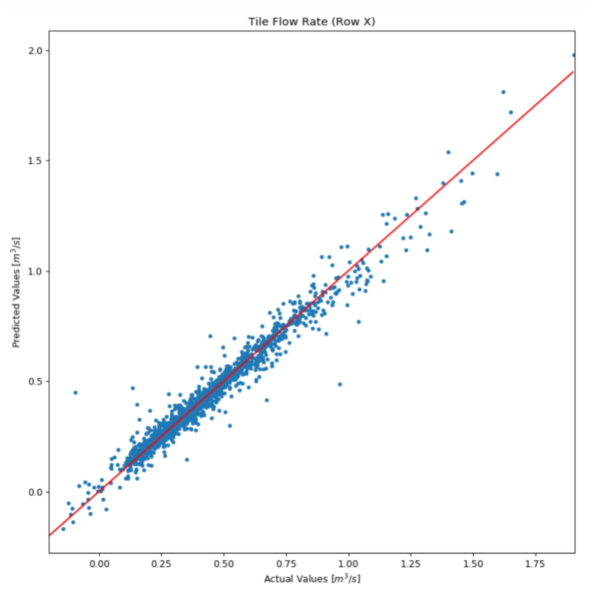


(d) MSE Loss of maximum cabinet inlet temperature (Row Z)

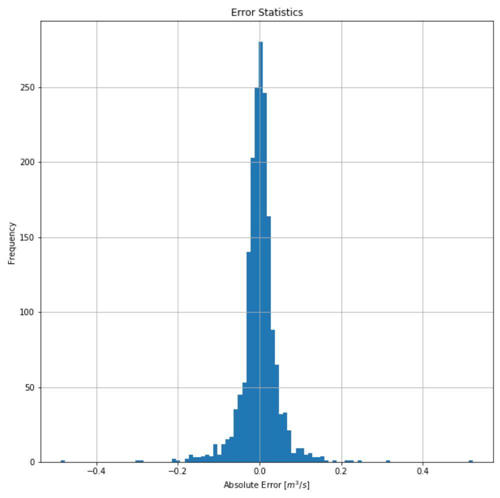
Figure 5.6: The performance of the neural networks predicting the max. inlet temperature of cabinets of row X and row Y respectively. The R^2 accuracy and MSE loss is plotted for row X in sub-figure (a) and (b) and row Y in sub-figure (c) and (d). From the the training and testing accuracy graphs, the neural nets starts to overfit after 12000 epochs. The neural networks are not generalizing well. The training of the networks should be stopped at 12000 epochs.

In figure 5.7, the regression plot for the predicted vs. the actual tile flow rate for row X and row Y is plotted. It is seen that the points lie close to the line $(Y = X)$, which indicates that the predicted values almost resemble the real ones. The absolute error for these two rows is also shown in the same figure. It is observed that the mean of the absolute error is centered at $0m^3/s$, and most of the error is occurring around it. This indicates that most of the error is around zero. The range of the majority of these errors is within $-0.2m^3/s$ to $0.2m^3/s$.

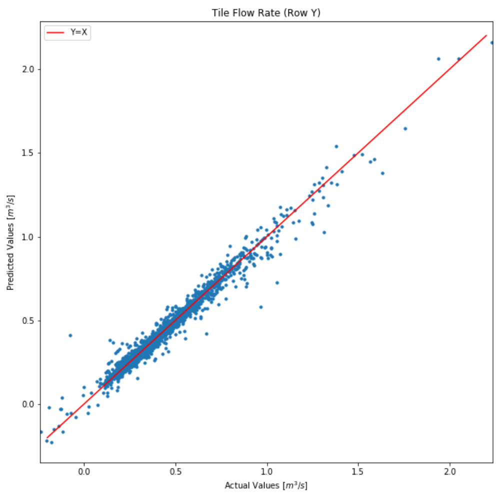
The performance of the neural network (R^2 accuracy and MSE loss) predicting the row W/X/Y/Z is shown in the table 5.2.



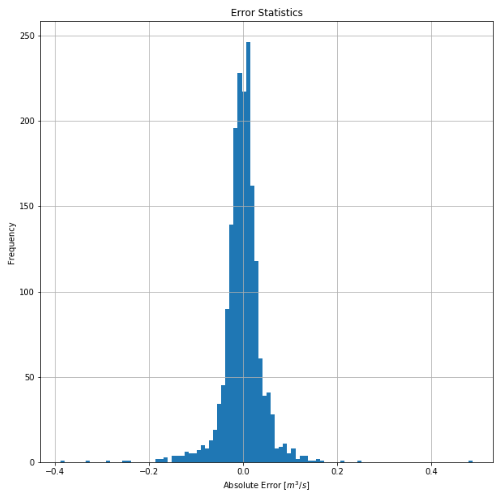
(a) The regression plot for predicted vs actual tile flow rate (Row X).



(b) The error histogram of tile flow rate (Row X).



(c) The regression plot for predicted vs actual tile flow rate (Row Y).



(d) The error histogram of tile flow rate (Row Y).

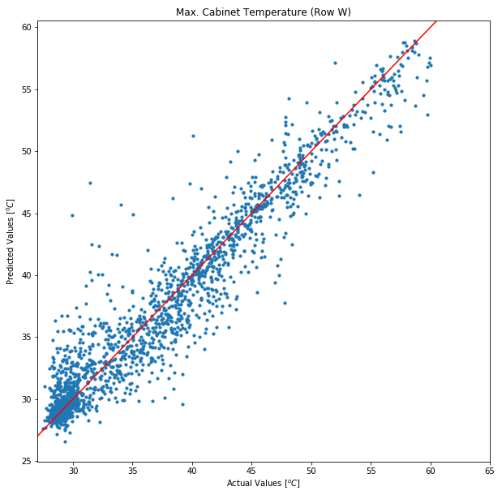
Figure 5.7: The regression plot for the predicted vs the actual tile flow rate is shown in sub-figures (a) and (c). It is seen in these sub-figures that most of the points lie close to the line ($Y = X$), highlighted in red. This indicates that the predicted results are close to the actual ones. The error histogram of row X and row Y in sub-figures (b) and (d) shows that the mean of the absolute error is centered at $0m^3/s$. The spread of majority of this errors is narrow and centered around $0m^3/s$, indicating the neural network is well trained.

	R^2 Accuracy		MSE Loss	
	Training	Testing	Training	Testing
Row W	0.933	0.910	1.574	1.709
Row X	0.978	0.962	0.0013	0.0022
Row Y	0.981	0.965	0.0013	0.0021
Row Z	0.936	0.904	1.63	1.858

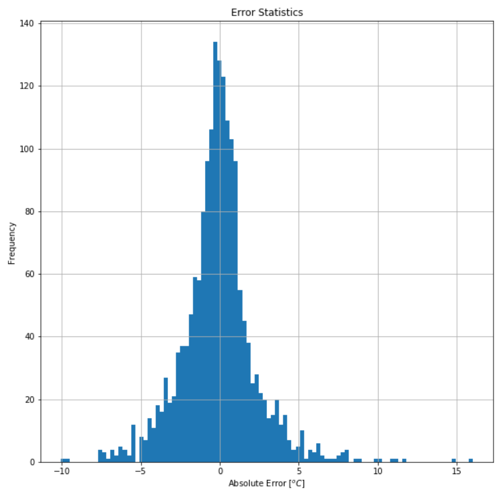
Table 5.2: The R^2 accuracy and MSE loss of the neural networks predicting the variables in row W/X/Y/Z. The tile flow rate of the floor tiles present in row X and row Y is being predicted by the neural net. Similarly, the max. cabinet inlet temperature of the cabinets present in row W and row Z is being predicted.

In figure 5.8, the regression plot for the predicted vs. the actual max. cabinet inlet temperature for row W and row Z is plotted. It is seen that the points are spread out around the line ($Y = X$), which indicates that the predicted values don't quite well resemble the actual ones. The absolute error for these two rows is also shown in the same figure. It is observed that the mean of the absolute error is centered at $0°C$, and most of the error is occurring around it. This indicates that most of the error is around zero. The spread of the absolute error is wider than the errors from the neural nets predicting the tile flow rate. Thus the neural nets predicting the cabinet temperature have to be trained further with more training data to reduce this spread of the absolute error of the max. cabinet inlet temperature. The range for the majority of these errors is within $-5°C$ to $5°C$, with a maximum frequency of occurrence of these errors centered around $0°C$. The neural network is going to need more training data to improve the accuracy of the cabinet temperature prediction and thus reduce the spread of the absolute error around $0°C$.

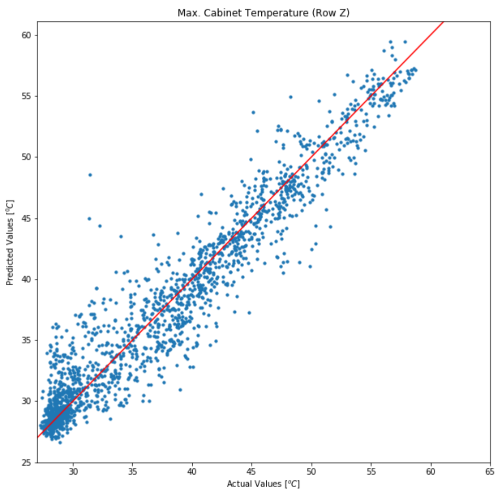
The error statistics of the tile flow rate of each floor tiles present in row X and row Y, and for the cabinet inlet temperature of each cabinet present in row W and row Z are shown in Appendix I respectively. It is seen in figure I.1 and figure I.2, that the individual floor tiles and the individual cabinets contribute unequally to the total absolute error in predicting the respective quantities. This means that the individual errors are all centered at 0 units, but the spread of the error for each quantity is different.



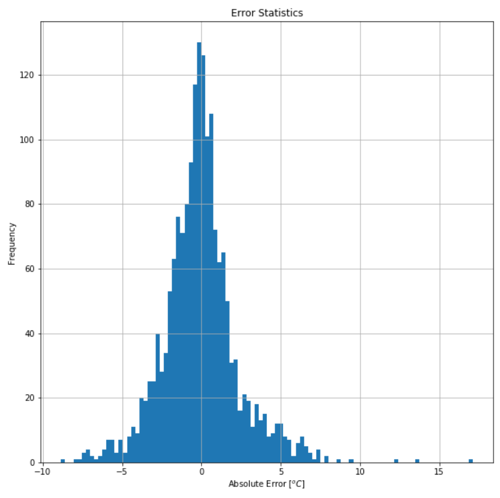
(a) The regression plot for the predicted vs the actual max. cabinet inlet temperature (Row W).



(b) The error histogram of the max. cabinet inlet temperature (Row W).



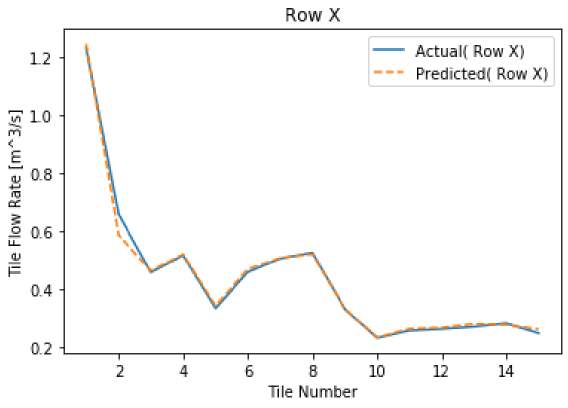
(c) The regression plot for the predicted vs the actual max. cabinet inlet temperature (Row Z).



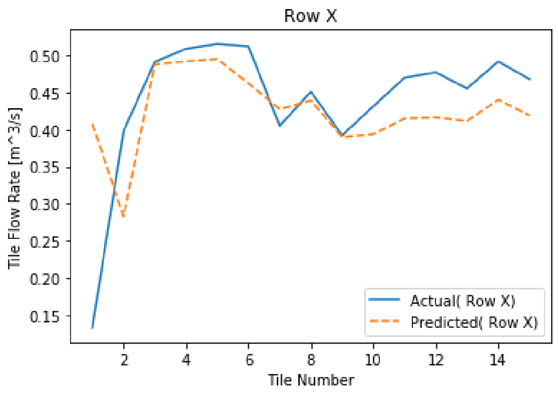
(d) The error histogram of the max. cabinet inlet temperature (Row Z).

Figure 5.8: The regression plot for the predicted vs the actual max. cabinet inlet temperature is shown in sub-figure (a) and (c). It is seen in these sub-figures that the point are spread out around the line ($Y = X$), highlighted in red. This indicates that the predicted results are not representing the actual ones with good accuracy. The error histogram of row W and row Z in sub-figure (b) and (d) shows that the mean of the absolute error is centered at $0m^3/s$ and it is having a wider spread around the mean. The range of majority of these error is within $-5°C$ to $5°C$.

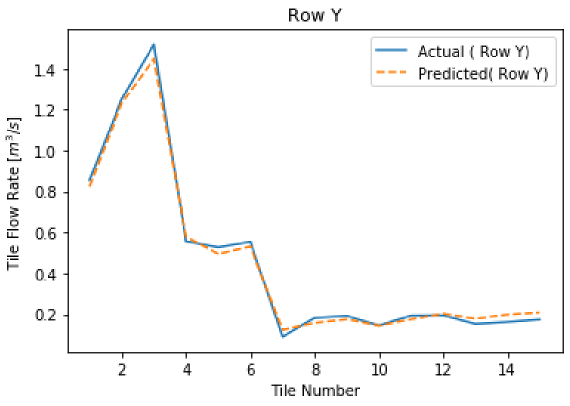
In figure 5.9 and figure 5.10, few validation of the prediction results from the testing dataset is shown. For the tile flow rate prediction of row X and row Y, it is seen that the predicted results follow the trend set by the actual values, and the neural network predicts them quite accurately. For the max. cabinet inlet temperature prediction of row W and row Z, the predicted results follow the trend from the actual values, but it does not predict them accurately. The performance of the max. cabinet inlet temperature prediction can be improved further by exposing the neural network to more training dataset or improving the architecture of the neural networks.



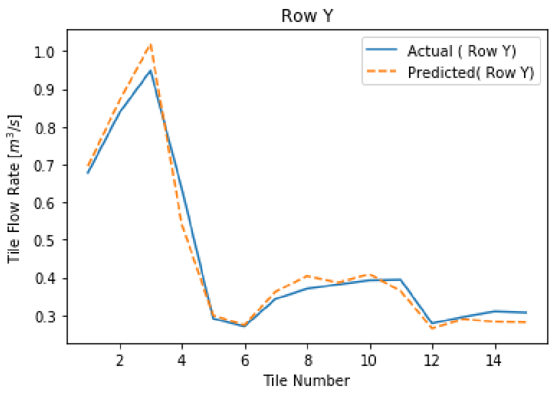
(a) The NN prediction of the tile flow rate (Row X).



(b) The NN prediction of the tile flow rate (Row X).

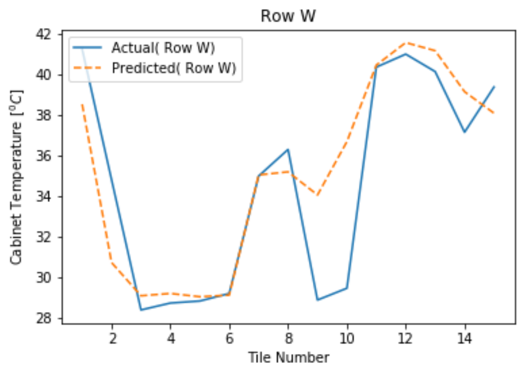


(c) The NN prediction of the tile flow rate (Row Y).

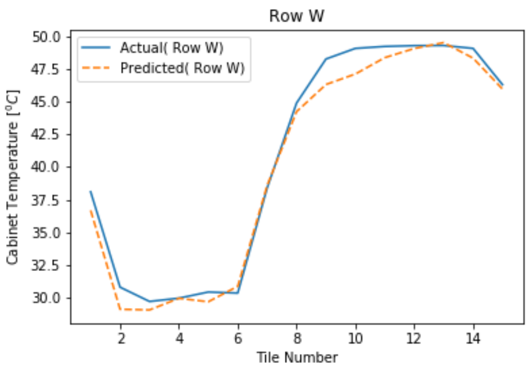


(d) The NN prediction of the tile flow rate (Row Y).

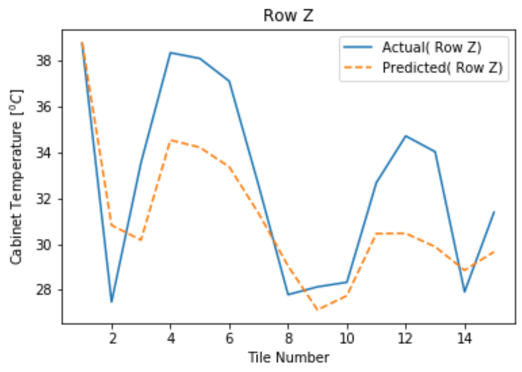
Figure 5.9: The Neural network (NN) predictions of the tile flow rate of the floor tiles present in row X (sub-figure (a)) and (sub-figure (b)), and the floor tiles present in row Y (sub-figure (c)) and (sub-figure (d)) from the testing dataset. The samples are picked randomly from the testing dataset. The tile flow rate $[m^3/s]$ is plotted on the vertical axis and the tile number on the horizontal axis. Tile no. 1 is near the CRAH and tile no. 15 away from the CRAH. The predicted results matches with the actual result except in (sub-figure (b)). Overall, the prediction result follows the trend of the actual result.



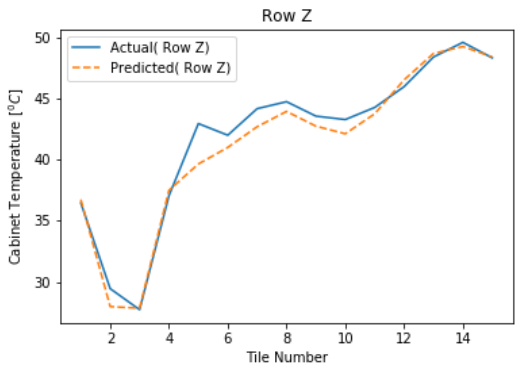
(a) The NN prediction of the max. cabinet inlet temperature (Row W).



(b) The NN prediction of the max. cabinet inlet temperature (Row W).



(c) The NN prediction of the max. cabinet inlet temperature (Row Z).

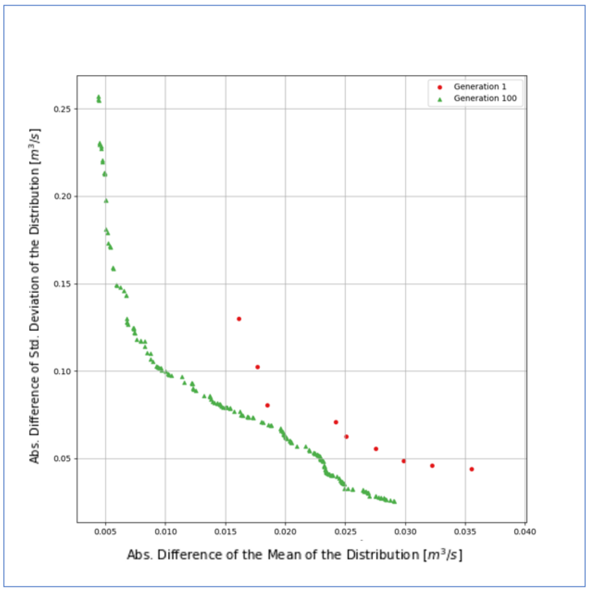


(d) The NN prediction of the max. cabinet inlet temperature (Row Z)

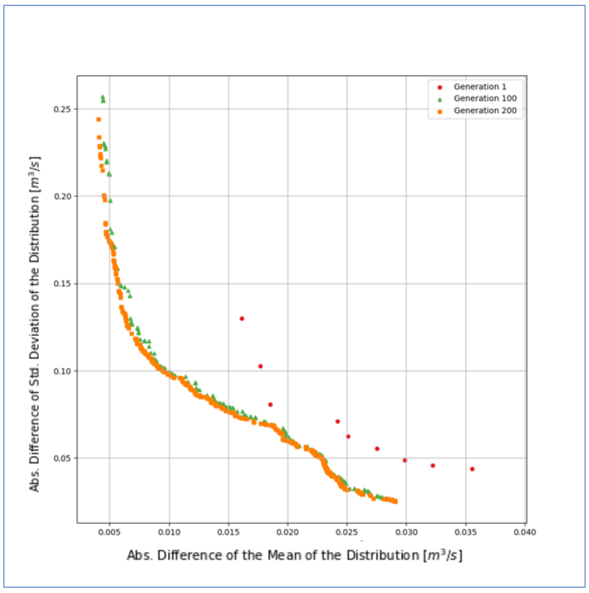
Figure 5.10: The Neural Network (NN) predictions of the max. cabinet inlet temperature of the cabinets present in row W (sub-figure (a)) and (sub-figure (b)), and the cabinets present in row Z (sub-figure (c)) and (sub-figure (d)) from the testing dataset. The cabinet temperature $[^{\circ}C]$ is plotted on the vertical axis and the cabinet number on the horizontal axis. Cabinet no. 1 is near the CRAH and cabinet no. 15 away from the CRAH. The predicted results follows the trend of the actual results, but do not predict it quite accurately.

5.4. The Optimized Design Parameters of the White Space using NSGA-II Algorithm

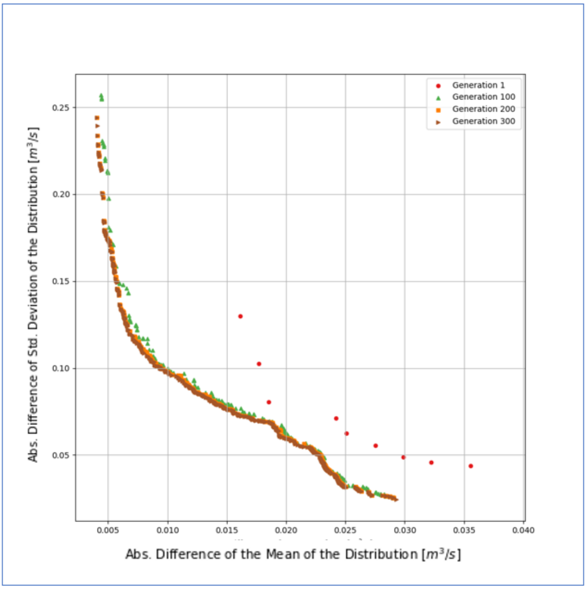
The NSGA-II algorithm (see section 4.6) is used to find the 11 optimum design parameter of the white space. This is shown in the figure 5.11. In this graph, for each generation the Pareto front (containing the non-dominated solutions) with the highest rank is plotted. The highest rank is 1 and in this case it is assigned to the Pareto front which contains the best set of non-dominated solutions in each generations i.e. the set of non-dominated solutions which are present more towards the origin (0,0). In figure 5.11, it is seen that as the number of generations increases, the population of each Pareto front increases and moves towards the origin $(0,0)$. This shows that the objectives are being minimized. After the 300^{th} generation , there is no significant change in the Pareto front and thus the algorithm can be terminated at the 300^{th} generation . The set of non-dominated solutions from the Pareto front at the 300^{th} generation is obtained. This will give the optimum value of the 11 design parameters of the white space.



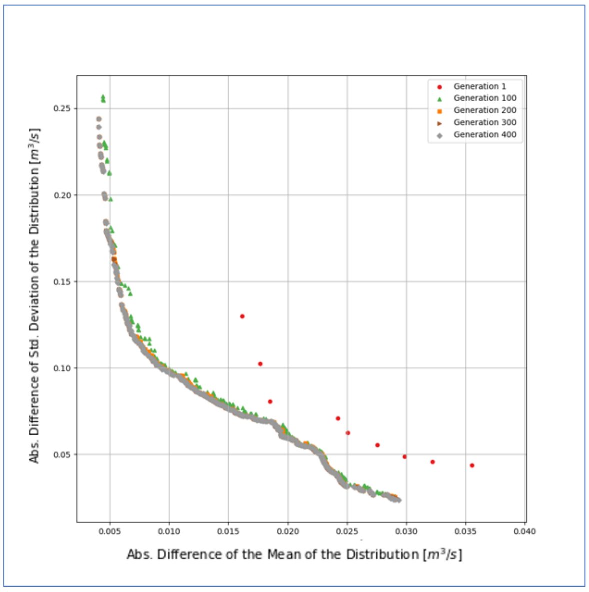
(a) The Pareto fronts of gen. 1 and 100.



(b) The Pareto fronts of gen. 1, 100, and 200.



(c) The Pareto fronts of gen. 1, 100, 200, and 300.



(d) The Pareto fronts of gen. 1, 100, 200, 300, and 400.

Figure 5.11: The graph shows the Pareto fronts of each generations till the 400^{th} generation. The absolute difference of the mean of the two distributions is plotted on the vertical axis and the absolute difference of the standard deviation of the two distributions is plotted on the horizontal axis. For each generation, the Pareto front with the highest rank is shown. Both the objectives, i.e. the mean and the standard deviation are being minimized as indicated by the convex shape of the Pareto front as viewed from the origin $(0, 0)$. After the 300^{th} generation, the minimization of the objectives are not significant anymore as the Pareto front for the 400^{th} generation starts to overlap with the 300^{th} generation. The optimal set of design parameters is obtained from the Pareto front of the 300^{th} generation.

A comparison of the performance of the NSGA-II algorithm (which itself is relying on the tile flow rate neural network to create the solution space) against the tile flow rate neural network to find the optimum design solution of the white space is shown in the figure 5.12. The data-points for the neural network have been obtained from the testing dataset. It is seen that the NSGA-II algorithm performs well in finding the optimum solution of the 11 design parameters of the white space as compared to the neural network. But it would be unfair to make this comparison, as the task of the neural network is to predict the ground truth value and not to find the optimal solution. In the process of predicting the ground truth values, the neural network might find the optimal solutions by luck, and it might perform better than the genetic algorithm. But the chances of this happening are meager.

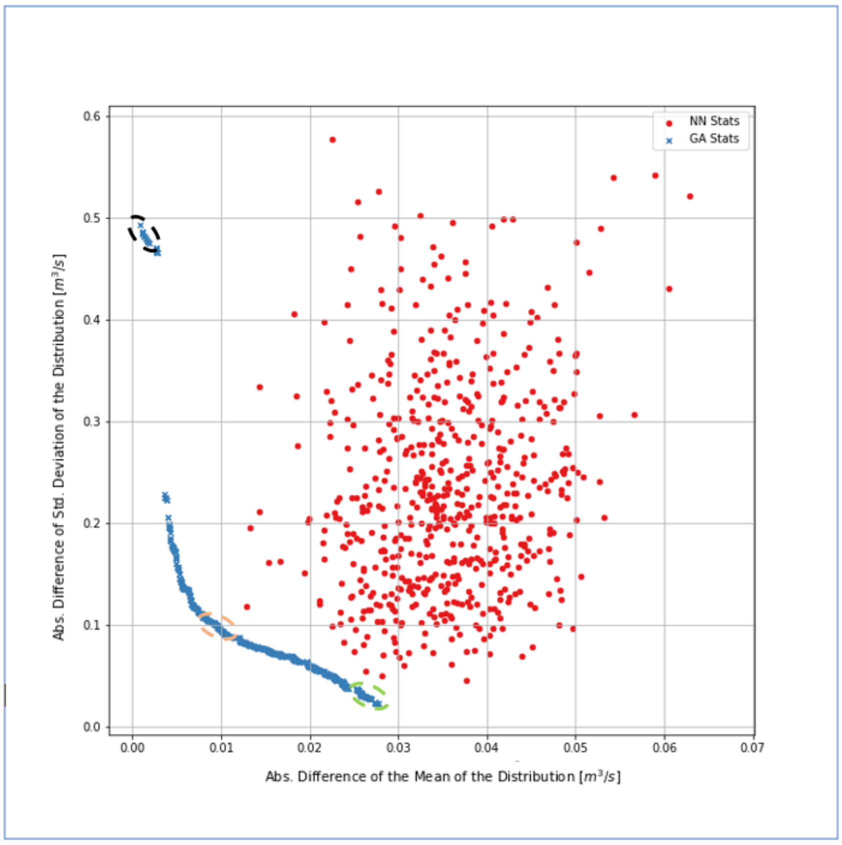
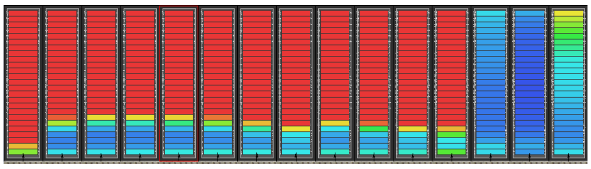
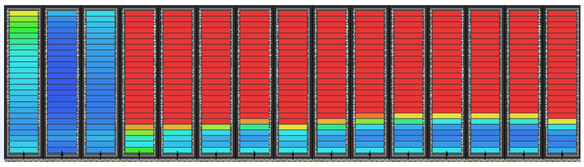
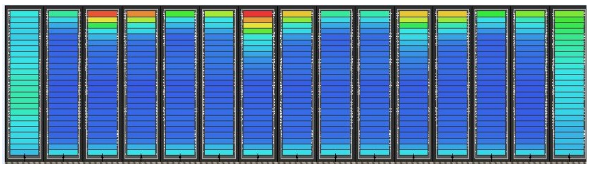


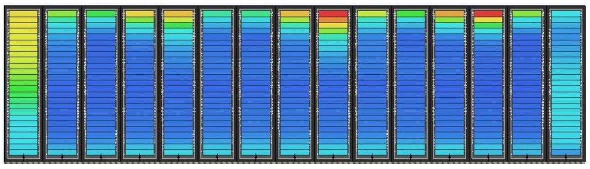
Figure 5.12: The comparison of NSGA-II with the tile flow-rate neural network. The Pareto front (in blue) of 300^{th} generation obtained from the GA is compared with the predictions (in red) from the neural net. The absolute difference of the mean of the two distributions (Tile Flow Rate Distributions and Cabinet Airflow Requirement Distribution) is plotted on the vertical axis, and the absolute difference of the standard deviation of the two distributions is plotted on the horizontal axis. The NSGA-II performs better in obtaining the optimal design space solutions as compared to the neural net. The neural network is primarily tasked to predict the ground truth values. But in the process, it may find the optimal solution. The points highlighted (by the black, orange and green oval shapes) in the Pareto front are the set of solutions taken to decide the best optimal design solution.

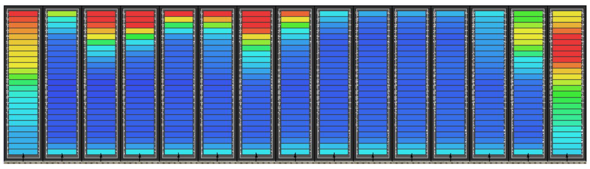
In figure 5.12, the Pareto Front is highlighted by the black, orange, and green zones. These are the zones from which optimized sample points are taken to be compared with each other. From each zone, the best sample point is chosen. This is shown in figure 5.13. In this figure, the range of the mean inlet temperature of the servers is kept from $25°C$ to $32°C$. The upper limit has been chosen to be $32°C$ as we are interested in limiting the server temperature to less than $32°C$. Thus this is going to give us an indication of how many servers have an inlet temperature greater than or equal to $32°C$. It is not suitable to have the server mean inlet temperature above $32°C$, as the servers start to overheat. In figure 5.13, the contour, which is indicating a temperature of more than $32°C$, indicates that it is better not to put the server in that location. It is seen in this figure that the solution on the Pareto front from the orange zone (figure 5.13c and figure 5.13d), has a better mean server inlet temperature, i.e., most of the servers have their inlet temperature lower than $27°C$ and very few of them beyond $32°C$, as compared to the solutions on the Pareto front from the black zone and green zone. The actual range of temperatures of these servers is shown in Appendix J.

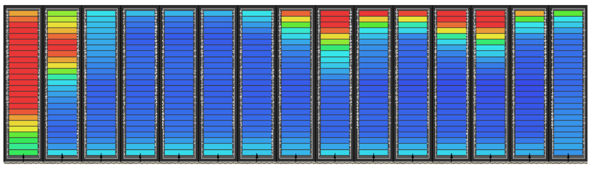

(a) Mean server inlet temperature of row W (Black Zone)


(b) Mean server inlet temperature of row Z (Black Zone)


(c) Mean server inlet temperature of row W (Orange Zone)


(d) Mean server inlet temperature of row Z (Orange Zone)


(e) Mean server inlet temperature of row W (Green Zone)


(f) Mean server inlet temperature of row Z (Green Zone)

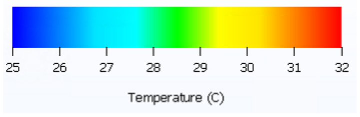


Figure 5.13: The plot of mean server inlet temperature present in row W and row Z respectively. Solution points from the Pareto front is taken from the black, orange and green zones and the best solution out of these zones is taken for comparison. The range of the mean inlet temperature of the servers is kept from $25°C$ to $32°C$. The upper limit has been chosen to be $32°C$ as we are interested in limiting the server temperature to less than $32°C$. It is seen that the solution from the orange zone gives the better optimal solution as most of the server has a mean inlet temperature below $27°C$ and very few beyond $32°C$. Thus the solution shown in (sub-figure (c)) and (sub-figure (d)) can be considered as an optimal solution.

Thus, the optimal values of the 11 design parameters of the white space required to keep most of the servers below the mean inlet temperature of $27°C$ are as follows.

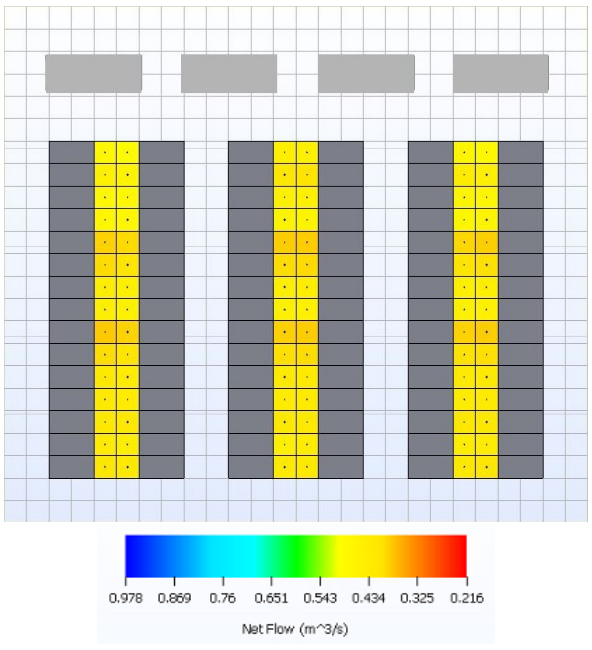
- Position of Perforated Plate 1 (P1) $= 3.79m$
- Position of Perforated Plate 2 (P2) $= 6.42m$
- Position of Perforated Plate 3 (P3) $= 8.81m$
- Position of Perforated Plate 4 (P4) $= 10.8m$
- Height of the Raised Floor (h) $= 0.97m$
- CRAH Distance $= 2.4m$
- Percentage of Perforation of Plate 1 (P1) $= 84.4\%$
- Percentage of Perforation of Plate 2 (P2) $= 68.19\%$
- Percentage of Perforation of Plate 3 (P3) $= 96.53\%$

- Percentage of Perforation of Plate 4 (P4) = 33.66%
- Floor Tile Perforation = 50.16%

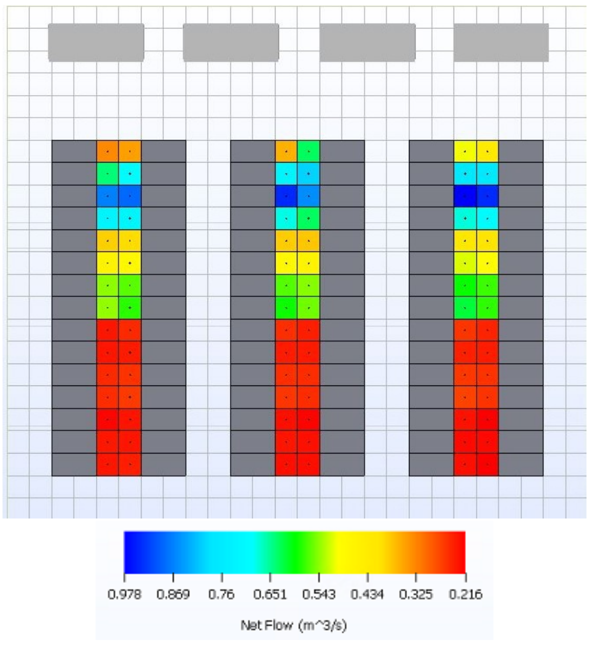
The position of the perforated plates and CRAH distance is calculated perpendicular to the X-axis in the positive Z-direction (see figure 4.4 for alignment of the axis). The height of the raised floor is calculated perpendicular to the X-axis in the positive Y-direction. From the values of these 11 parameters shown above, a small conclusion can be made that the presence of perforated plate 1 and plate 3 would not be required under the raised floor as they are highly perforated. A proper conclusion can be made by finding the interaction effect of these plates with each other on the flow distribution and the temperature field in the white space. The CRAH should be placed as close as possible (2-floor tiles away from the wall) to the first cabinet of the row but not too close to cause a suction effect in the raised floor. The height of the raised floor is around $1m$, which is in alignment with the plot shown in figure 2.11a. The value of the floor tile perforation suggests having a higher perforation as compared to the plot shown in figure 2.11b. From figure 2.11b, it is suggested to have a tile perforation around $15\% - 20\%$. A higher floor tile perforation would lead to non-uniform flow across the floor tiles present in the raised floor. But this non-uniformity of the tile flow can be counteracted by the higher height of the raised floor and presence of perforated plates in the raised floor.

A comparison of the optimized case has also been made with a non-optimized case (with plates) and the white space without the perforated plates in the raised floor. This is shown in the following figures. In figure 5.14, a comparison of the tile flow rate has been made among the optimized case, one of the non-optimized case, and the case without the perforated plates in the underfloor. It is seen that in the optimized case (figure 5.14a), there is a uniform distribution of the tile flow rate of the floor tile around the value of $0.42m^3/s$. In the non-optimized case (figure 5.14b), there is a variation of the value of the tile flow rate. In the case without the perforated plates (figure 5.14c) in the raised floor, there is a negative tile flow rate for the floor tiles present near the CRAH, and it increases (the pressure drop across the floor tile increases as we move away from the CRAH) as the distance from the CRAH increases. Thus the optimized case achieves the requirement of a uniform tile flow rate necessary to keep as many servers below the mean inlet temperature of $27°C$.

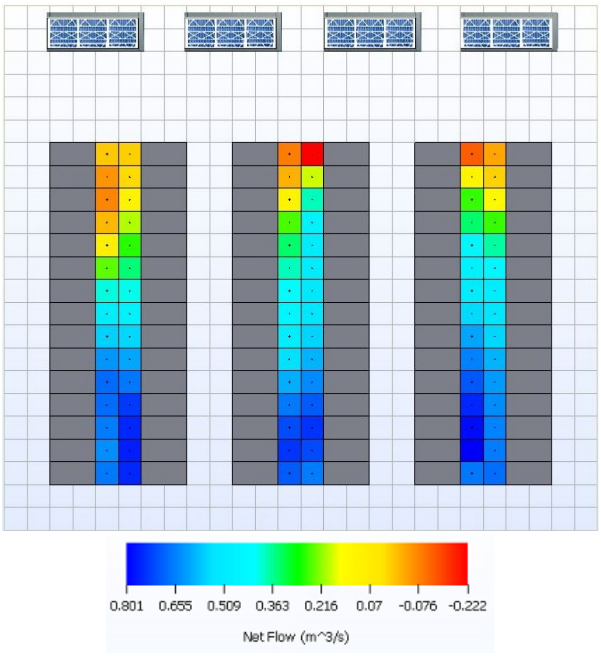
The effect of the distribution of the tile flow rate on the supply (cold) temperature through the floor tile is shown in the figure 5.15. In the optimized case (figure 5.15a), the distribution of the floor tile's temperature is uniform and around $(25°C)$, as required by the cabinets. In the non-optimized case figure 5.15b, there is a slight variation in this temperature, more towards the floor tiles present near the CRAH. This too can be attributed to the entry of the hot air from the surroundings into the cold aisle from the sides of the rows containing the floor tiles. This can lead to an increase in the supply temperature of the airflow provided to the cabinets. In the case without the plates in the raised floor (figure 5.15c, the variation of the supply temperature is significant and more pronounced in the floor tiles present near the CRAH. This is due to the negative tile flow rate (see figure 5.14c) of the floor tiles present near the CRAH. This negative tile flow rate causes the hot air from the surrounding to be pulled into the raised floor and thus raising the supply temperature.



(a) The tile flow rate of the floor tiles in the optimized design condition.

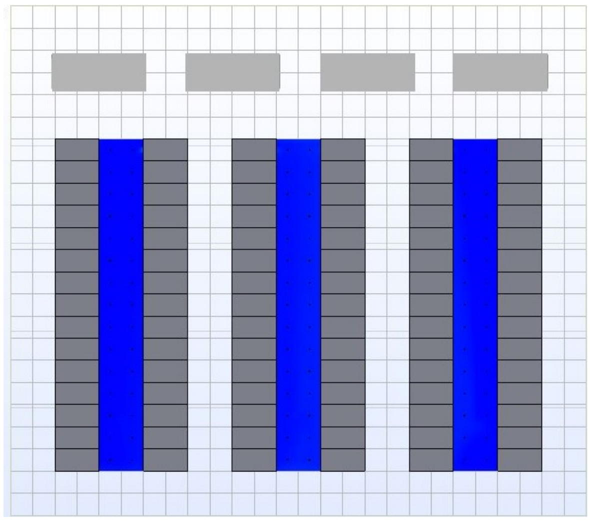


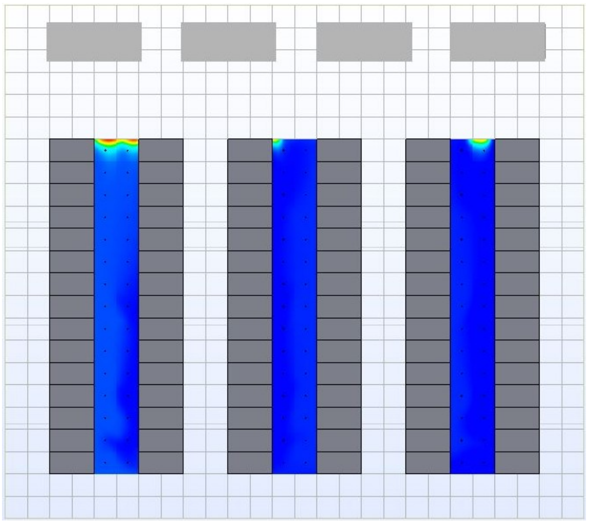
(b) The tile flow rate of the floor tiles in the non-optimized design condition.

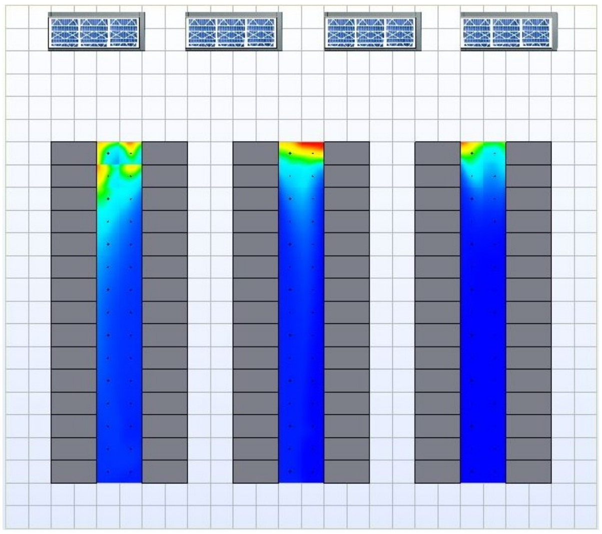


(c) The tile flow rate of the floor tiles in the white space without perforated plates.

Figure 5.14: The contour of the tile flow rate of the floor tiles present in the white space. In the optimized case (sub-figure (a)), the tile flow rate is nearly uniform at around $0.42m^3/s$. In the non-optimized case (sub-figure (b)) there is a variation in the tile flow rate as we move away from the CRAH. In the case without perforated plates in the raised floor (sub-figure (c)), there is a negative tile flow rate in the floor tiles present near the CRAH, and then the tile flow rate increases.


(a) The supply air temperature of the floor tiles in the optimized design condition.


(b) The supply air temperature of the floor tiles in the non-optimized design condition


(c) The supply air temperature of the floor tiles in the white space without perforated plates.

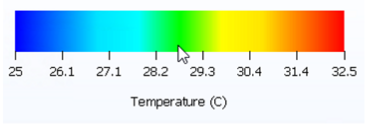
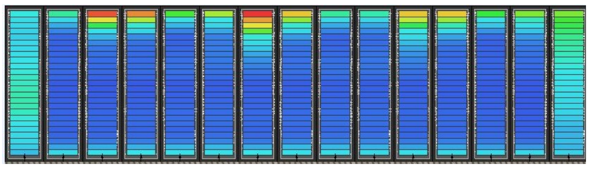
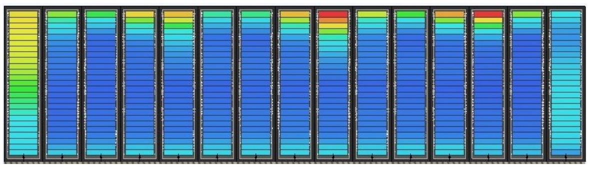
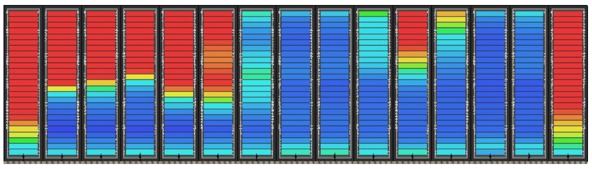


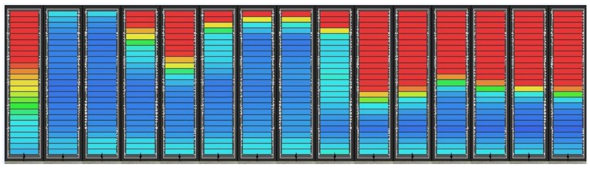
Figure 5.15: The contour of the supply air temperature of floor tiles present in the white space. In the optimized case (sub-figure (a)), the supply air temperature is nearly uniform around $25°C$. In the non-optimized case (sub-figure (b)), there is a slight variation in the supply air temperature of floor tiles. The tiles present near the CRAH have a higher temperature due to end-effects (hot air entering into the cold aisle from the edge of the aisle). In the case without perforated plates in the raised floor (sub-figure (c)), there is a significant variation of the floor tile temperature present near the CRAH. This is also due to the negative tile flow rate, which sucks in the hot air from the surroundings.

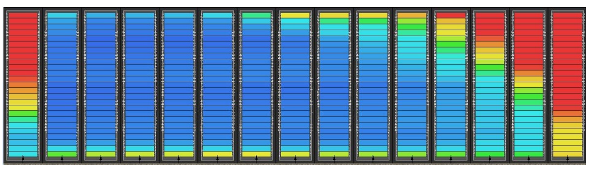
In figure 5.16, the mean server inlet temperature is shown for the servers present in row W and row Z. It is seen in the optimized case (figures 5.16a and 5.16b) most of the servers are within the temperature of $27°C$ and very few above $32°C$. In the non-optimized case (figures 5.16c and 5.16d) and the case without the perforated plates in the raised floor (figures 5.16e and 5.16f), more number of the servers inlet temperature lies above the temperature of $32°C$ as compared to the optimized case. The non-optimum case with the perforated plates in it, does not seem to perform better than the case without the perforated plates. The actual ranges of the temperature of the servers in the different cases (optimized, non-optimized, and without plates) are shown in Appendix J.2.

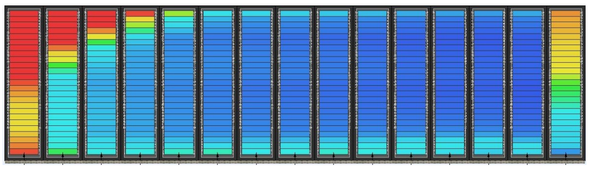

(a) The mean server inlet temperature of row W (Optimized).


(b) The mean server inlet temperature of row Z (Optimized).


(c) The mean server inlet temperature of row W (Non-Optimized).


(d) The mean server inlet temperature of row Z (Non-Optimized).


(e) The mean server inlet temperature of row W (Without Perforated Plates).


(f) The mean server inlet temperature of row Z (Without Perforated Plates).

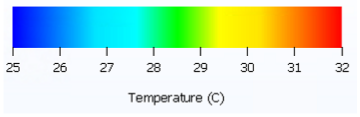
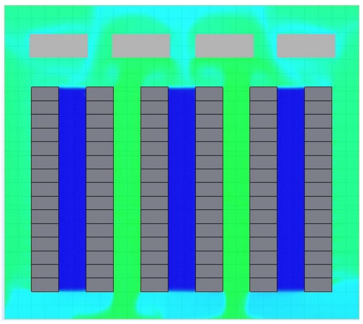
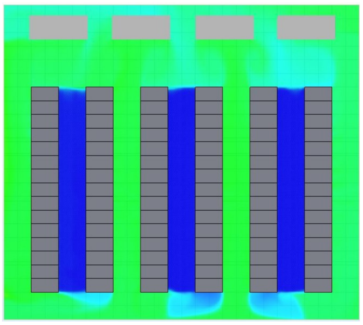
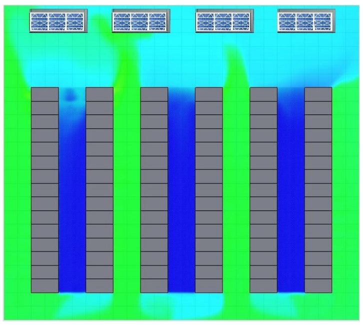


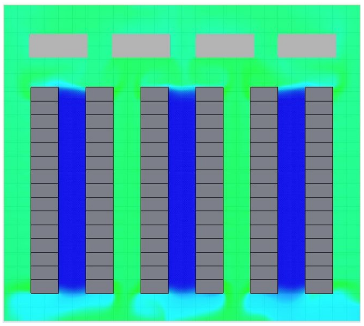
Figure 5.16: The plot of mean server inlet temperature present in row W and row Z, respectively, and shown for the optimized case (sub-figure (a)) and (sub-figure (b)), non-optimized case (sub-figure (c)) and (sub-figure (d)), and without plates (sub-figure (e)) and (sub-figure (f)). For the servers present in the optimized case, most of them are within the temperature of $27°C$ and very few above $32°C$. In the non-optimized case, most of the server temperatures are above $32°C$, and in the case without the perforated plates, the server temperature lies within $27°C$. However, in the servers present in the corner cabinets, the temperature exceeds $32°C$. Thus servers in case (sub-figure (a)) and (sub-figure (b)) have a better mean server inlet temperature.

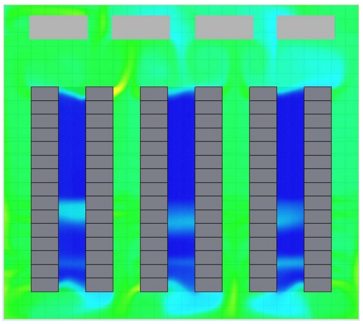
The effect of the optimization of the design parameters of the white space can be seen through the temperature plots in figure 5.17 and figure 5.18. In these two figures, the temperature plot for the optimized case with the perforated plates in the raised floor has been plotted in the left column, the non-optimized case with plates in the raised floor in the middle column and the case without plates in the right column. In figure 5.17, the temperature plot is taken along the height of the cabinet in which the supplied cold air is being optimally provided till the top of the cabinet, whereas in the non-optimized case and case without the perforated plates, there is infiltration of hot air from the side of the row and presence of re-circulation regions. This causes the inlet temperature of the airflow provided to the server to increase. In figure 5.18, the temperature plots are taken along the length of the row of cabinets (in the Z-direction) starting from the cabinets present near the CRAH. In this figure it is seen that in the optimized case, the supplied cold air is being optimally utilized by all the servers in the cabinets. There is no wastage of the cold air to the surroundings. In the non optimized case and the case without the perforated plates, there is wastage of cold air to the surrounding at certain distance away from the CRAH and presence of re-circulation regions at other distances causing an increase in the supply airflow temperature.

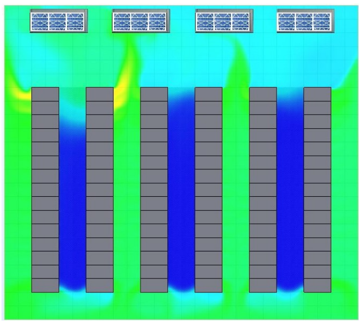

(a) (OC)-0.1m

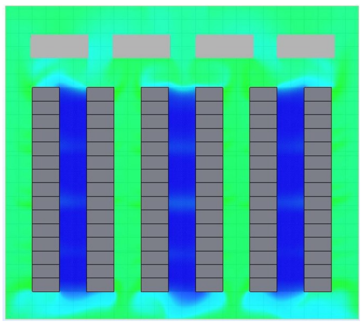

(b) (NOC)-0.1m

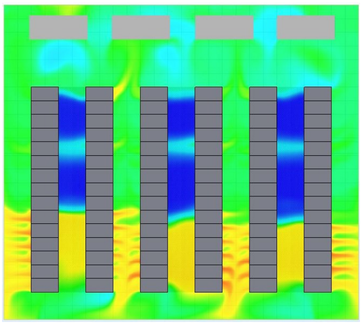

(c) (NOCw/oP)-0.1m

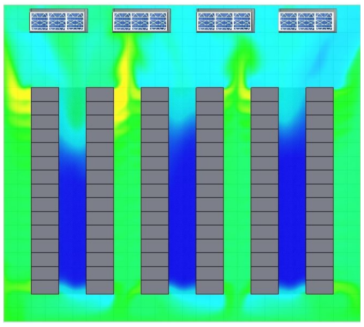

(d) (OC)-1.02m

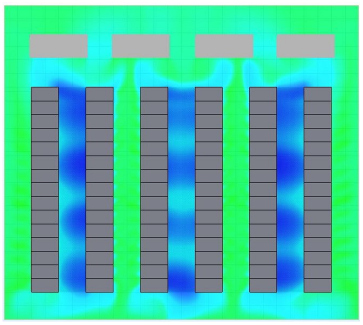

(e) (NOC)-1.02m

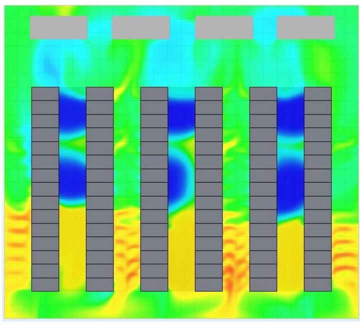

(f) (NOCw/oP)-1.02m

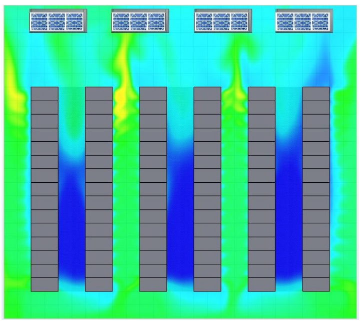

(g) (OC)-1.94m


(h) (NOC)-1.94m


(i) (NOCw/oP)-1.94m


(j) (OC)-2.39m


(k) (NOC)-2.39m


(l) (NOCw/oP)-2.39m

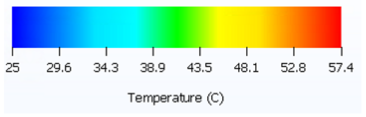
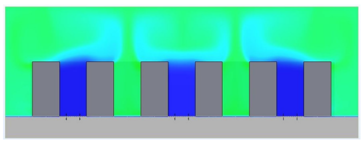
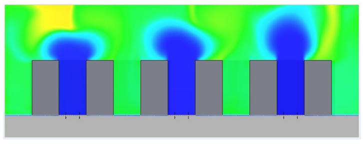
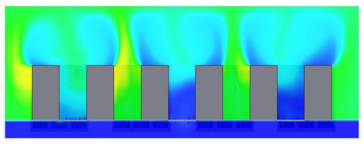
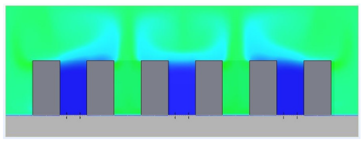


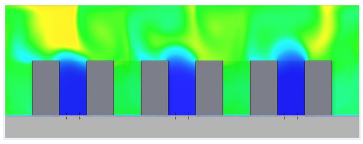
Figure 5.17: The figures shows the temperature contours of three cases. The first one is the Optimized Case (OC) shown in the first column from the left, Non-Optimized Case (NoC) shown in the second column, and the Non-Optimized Case without perforated Plates (NOCw/oP) shown in the third column. The result planes are taken at several distances from the raised floor, starting from the raised floor. In OC, as all the 11 design parameters are optimized, there is not much infiltration of hot air from the sides and sufficient amount of cold air is supplied till the top of the cabinet (sub-figure (j)). In NOC, there is infiltration of hot air from the sides and prominent presence of re-circulation regions as the distance increases from the raised floor. Similarly in NOCw/oP, there is more infiltration of hot air from the sides as compared to the other cases.

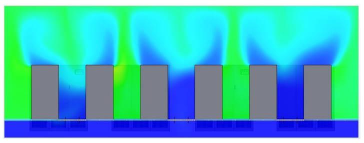

(a) (OC)-5.8m

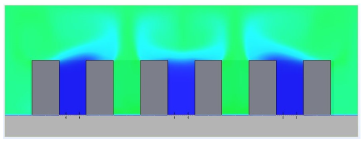

(b) (NOC)-5.8m

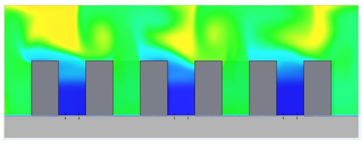

(c) (NOCw/oP)-5.8m

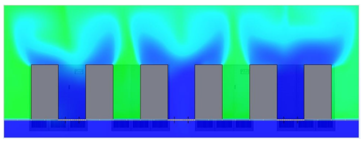

(d) (OC)-6.8m

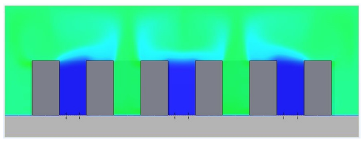

(e) (NOC)-6.8m

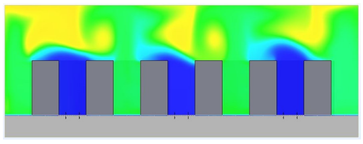

(f) (NOCw/oP)-6.8m

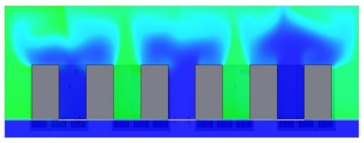

(g) (OC)-7.8m

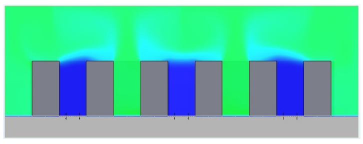

(h) (NOC)-7.8m

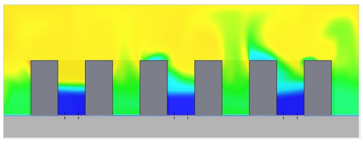

(i) (NOCw/oP)-7.8m

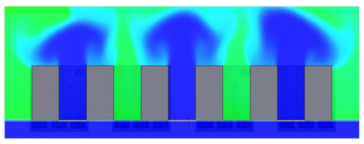

(j) (OC)-8.8m

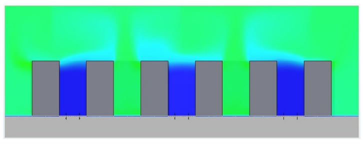

(k) (NOC)-8.8m

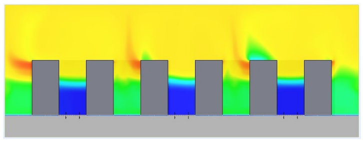

(l) (NOCw/oP)-8.8m

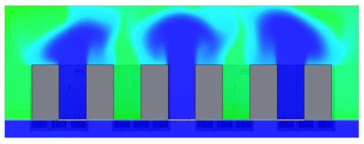

(m) (OC)-10.8m

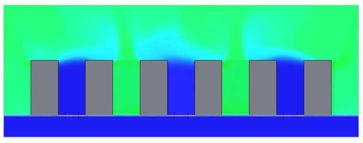

(n) (NOC)-10.8m

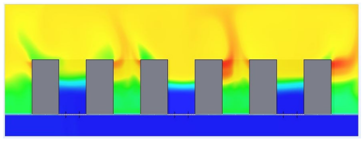

(o) (NOCw/oP)-10.8m

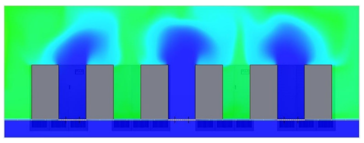

(p) (OC)-11.8m

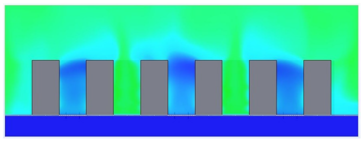

(q) (NOC)-11.8m

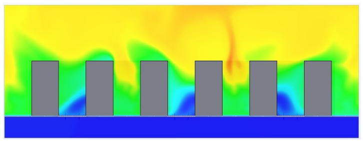

(r) (NOCw/oP)-11.8m

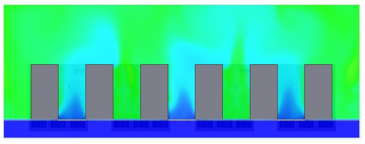

(s) (OC)-12.8m


(t) (NOC)-12.8m


(u) (NOCw/oP)-12.8m


(v) (OC)-13.79m


(w) (NOC)-13.79m


(x) (NOCw/oP)-13.79m

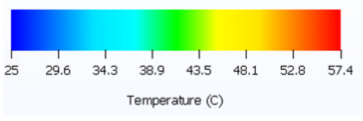


Figure 5.18: The figures shows the temperature contours of three cases. The first one is the Optimized Case (OC) shown in the first column from the left, Non-Optimized Case (NoC) shown in the second column, and the Non-Optimized Case without perforated Plates (NOCw/oP) shown in the third column. The result planes are taken at several distance from the wall which is parallel to XY plane, starting from the cabinets present near the CRAH and till the last cabinet. In OC, the supplied cold air is being optimally used by the cabinets and there is no wastage of it to the surrounding. In NOC, there is a slight wastage of cold air to the surrounding and presence of re-circulation regions. In NOCw/oP there is significant amount of cold air being wasted to the surroundings.

Thus, it can be stated that the NSGA-II algorithm, along with the neural network are a useful tool to obtain the optimal set of design parameters of the white space. These tools can be used to maintain as many servers as possible within the temperature limit of $27°C$. Better solutions than the stated optimized case can be obtained with the help of these tools (genetic algorithm and neural networks) by modifying the hyper-parameters of these tools, generating more training data, and designing a better optimization tool.

6 Conclusion and Recommendations

6.1. Conclusion

The ultimate goal of the study performed in this thesis is to find out the optimized design parameters of the data center white space, which are required to maintain the server's inlet temperature within the ASHRAE recommended temperature range, for a fixed cooling load. The task of finding these optimum values is done by using CFD simulations, neural networks, and a genetic algorithm. In the process of finding the optimized design variables, we are going to check if there is a reduction of the electrical power consumption by the CRAHs for cooling the servers in the white space. A comparison in terms of the required computational time is made between the CFD simulations and the combination of a neural network and a genetic algorithm to find the optimum solution.

Each floor tile should be able to provide the required airflow rate in a specific temperature range of $25°C$ to $27°C$, needed by a cabinet (containing servers) to maintain the server's inlet temperature within the ASHRAE recommended temperature range. It is assumed that a sufficient uniformly distributed tile flow rate, with the tile airflow supply temperature at $25°C$, is going to maintain all the server's inlet temperature in the ASHRAE temperature range. Perforated plates are introduced in the raised floor to control the tile flow rate. 11 design variables are considered, which influences the tile flow rate. These are the four perforated plates in the raised floor, the amount of perforation of these four perforated plates, the distance of the CRAH from the cabinets, the raised floor height, and the floor tile perforation. It is assumed that these variables affect the tile flow rate.

The Latin Hypercube Sampling (LHS) technique is used to generate a random combination of values for the 11 design variables. The corresponding tile flow rates and cabinet inlet temperatures, for each sample of the 11 design variables, are found from CFD simulations. 6SigmaRoom, a customized CFD software for data center white space simulations, is used. A simple design of the white space is considered for the CFD simulations. Using the data obtained from the LHS and CFD simulations, a database is created, which is used to train the neural networks. The artificial neural network is used to predict the tile flow rate of the floor tiles and the cabinet's inlet temperature separately. The results of the neural networks predicting the tile flow rate are used by the NSGA-II (a variant of the genetic algorithm) algorithm to find the optimized value of the 11 design variables of the data center white space.

The LHS technique generates a uniform distribution for all the 11 design variables. The CFD simulations provide a converged solution. In this thesis, the verification of the CFD simulations cannot be done as the experimental setup for the same is not available. In order to estimate the numerical error, a conclusion can be made based on the study done by Athavale et al. in the paper "Artificial Neural Network Based Prediction of Temperature and Flow Profile in Data Centers"[42]. The authors used the 6SigmaRoom software to verify their white space model with their experimental setup. They found that the error for the tile flow rate is less than 4% and $1.7°C$ for the server inlet temperature. Thus, the errors for the tile flow rate and server inlet temperature in the white space model, used in this study, can be estimated around these values.

The predictions by the neural networks for the tile flow rates are with an average R^2 accuracy of 0.97 and prediction error of less than 5%. For the cabinet inlet temperature, it is with an average R^2 accuracy of 0.92 and an prediction error of less than $2°C$. The errors from the CFD simulation and the neural networks are comparable. Thus, if the neural network were to predict from experimental data, then it would give nearly the same amount of error as given by the CFD simulations. The neural network quite accurately predicts the data obtained from the CFD simulation. The errors of the neural networks can be reduced further if it is trained on more training data. The tile flow rate neural network is used to create the sample space for the NSGA-II algorithm as its accuracy is higher than the neural network predicting the cabinet temperature. The optimized values of the 11 design variables obtained from the NSGA-II algorithm are verified by plugging in these values into the 6SigmaRoom software and running the CFD simulation. The distribution of the tile flow rate is obtained from the CFD simulation and compared with the distribution from the neural network. The error for this distribution is around 5%.

From the CFD simulation, it is seen in the optimized case that the tile flow rate distribution is nearly uniform, and more number of servers have their mean inlet temperature between $25°C$ to $27°C$ (within the ASHRAE temperature range), as compared to the non-optimized cases. There is no wastage of cold air and recirculation zones in the optimized case. From the 6SigmaRoom software, it is also possible to find the power required to run the fans of the CRAHs. There is a reduction of electrical power consumed by the CRAHs to cool the white space, by 10% for the optimized case as compared to the power consumed by the CRAHs in the non-optimized case.

A HP mobile workstation, with processor specification Intel(R) Core(TM) i7-6820HQ CPU @ 2.70GHz and 32.0GB RAM, is used for the running the CFD simulations, the neural networks, and the genetic algorithm. The CFD software used 4 physical cores. It took on an average of 40 minutes to obtain the converged CFD results. Once the neural network was well trained, it took less than 1 minute to predict the CFD results. The genetic algorithm took around 10 minutes to find the optimal solution. Thus in terms of computational time required, the combination of a neural network and a genetic algorithm takes less time for prediction and optimization, respectively, as compared to the time required to perform repeated CFD simulations. This provides an advantage but comes at the cost of the prediction and optimization accuracy. There is a possibility to use these tools (a neural network and a genetic algorithm) for real-time prediction and optimization of the temperature and flow fields if their accuracy can be improved further.

The research performed in this study is preliminary and a proof of concept of the applications of the neural network and the genetic algorithm in predicting and optimizing the temperature distribution and flow fields of a data center white space. A more in-depth study needs to be done to determine how these tools can be implemented at the industry level and work in synergy with the CFD software.

6.2. Recommendations

Using the CFD simulation, the neural network, and a genetic algorithm, a study can be done to find out the optimum number of perforated plates required in the raised floor to maintain a uniform tile flow rate distribution. A similar study can be done using the CFD, the neural network, and the genetic algorithm to see the effect of CRAH/CRAC failure on the tile flow distribution in the white space having perforated plates in the raised floor. An interesting CFD study can be done to see the effect of the vertically inclined perforated partitions in the raised floor on the floor tile's tile flow rate. The prediction accuracy of the neural network can be improved with more training data and by finding better algorithms to find the hyper-parameters of the neural network. Sensitivity analysis can be performed to find the relative importance of the 11 design parameters on the tile flow rate and cabinet temperature. This can improve the performance of the neural network model. The power consumption by the fans of the CRAH can also be considered as one of the design parameter for future studies. This can help in the optimization process and also in the prediction by the neural network. In this thesis, Python was used to design the neural network using the available libraries. The "Deep Learning" toolbox in MATLAB can also be used to design the neural network to compare which software (Python/MATLAB) gives a better design of the neural network. Several other variants of the genetic algorithm can be explored to obtain the optimized design parameter of the white-space. In this study, the optimization of the white space layout is done based on the tile flow rate of the floor tiles. The optimization can also be done depending on the server inlet temperatures. In this study, the white space modeling was simplified. The complexity of the white space modeling should be increased by including objects like ducts, cable trays, power distribution units, etc, which are usually found in the data center white space. Thus by increasing the complexity of the model, a realistic representation of the data center white space can be achieved.

A

Appendix-A

Python Script for Latin Hypercube Sampling Technique

```python

#Import required Librarys
from smt.sampling_methods import LHS

import pandas as pd
from pandas import ExcelWriter
import numpy as np
from openpyxl import load_workbook


#Load Excel File
wb = load_workbook('./Calculation_for_heat_load.xlsx')
#print(wb.get_sheet_names())


#Load Worksheet from Excel
sheet = wb.get_sheet_by_name('NN_Input_Para')
#sheet.title


#Get Parameter Maximum and Minimum values from Excel
Pow_min = sheet['C4'].value
Pow_max = sheet['D4'].value

Pos_P1_min = sheet['C5'].value
Pos_P1_max = sheet['D5'].value
Pos_P2_min = sheet['C6'].value
Pos_P2_max = sheet['D6'].value
Pos_P3_min = sheet['C7'].value
Pos_P3_max = sheet['D7'].value
Pos_P4_min = sheet['C8'].value
Pos_P4_max = sheet['D8'].value

HP_min = sheet['C10'].value
HP_max = sheet['D10'].value

CRAH_dist_min = sheet['C11'].value
CRAH_dist_max = sheet['D11'].value

Perf_P1_min = sheet['C12'].value
```

```
Perf_P1_max = sheet['D12'].value
Perf_P2_min = sheet['C13'].value
Perf_P2_max = sheet['D13'].value
Perf_P3_min = sheet['C14'].value
Perf_P3_max = sheet['D14'].value
Perf_P4_min = sheet['C15'].value
Perf_P4_max = sheet['D15'].value
#Perf_P5_min = sheet['C16'].value
#Perf_P5_max = sheet['D16'].value

Tile_Perf_min = sheet['C17'].value
Tile_Perf_max = sheet['D17'].value


# Execute Lattice Hypercube Sampling Method
# [Pow_min, Pow_max],
xlimits = np.array([[Pos_P1_min, Pos_P1_max],
                    [Pos_P2_min, Pos_P2_max],
                    [Pos_P3_min, Pos_P3_max],
                    [Pos_P4_min, Pos_P4_max],
                    [HP_min, HP_max],
                    [CRAH_dist_min, CRAH_dist_max],
                    [Perf_P1_min, Perf_P1_max],
                    [Perf_P2_min, Perf_P2_max],
                    [Perf_P3_min, Perf_P3_max],
                    [Perf_P4_min, Perf_P4_max],
                    [Tile_Perf_min, Tile_Perf_max]])
sampling = LHS(xlimits = xlimits)
print(xlimits)

print('*********')
N = 500             #Number of Samples
x = sampling(N)
x = np.round(x,2)
print('*********')
print(x.shape)
print('*********')


a = 215.28                                        # Area of the floor in m^2
Total_Load = x[:,0]*a                             # Total IT Load
R = 6                                             # Number of rows
Row_Load = np.round((Total_Load/R),2)             # IT Load per row
Cabinet = 15                                      # Number of Cabinets
Cabinet_Load = np.round((Row_Load/Cabinet),2)     # IT Load per Cabinet
Servers = 25                                      # Number of 2U Servers
Server_Load = np.round((Cabinet_Load/Servers),2)*1000  # IT Load per Server
Height_plate = x[:,4]-0.04                      # Height of Plate (mm)
Height_plate = Height_plate*1000
CRAH_Dist_Origin = 4.8 - 1.04 - x[:,5]            # CRAH Distance from Origin

#Store data in a Excel File
# 'Load per Cabinet (kW)': Cabinet_Load,
# 'Load per Server (W)': Server_Load,
# 'Power Dissipation per floor area (kW/m^2)':x[:,0],
# 'CRAH Distance (m)':x[:,6],
df = pd.DataFrame({'Serial Number':np.linspace(1,N, N),
                   'Position Plate 1 (m)':x[:,0],
                   'Position Plate 2 (m)':x[:,1],
                   'Position Plate 3 (m)':x[:,2],
                   'Position Plate 4 (m)':x[:,3],
                   'Height of Plenum (m)': x[:, 4],
                   'Height of Plates (mm)': Height_plate,
                   'Perforation Plate 2 (%)': x[:, 7],
                   'Perforation Plate 3 (%)': x[:, 8],
                   'Perforation Plate 4 (%)': x[:, 9],
                   'Perforation Plate 1 (%)': x[:, 6],
                   'Tile Perforation (%)': x[:, 10],
                   'ÇRAH Distance from Origin (m)': CRAH_Dist_Origin,
                    })

writer = ExcelWriter('Input499to998.xlsx') #Latin_Hypercube_Sampling(LHS)
df.to_excel(writer,'Sheet1',index=False)
writer.save()
```

Listing A.1: Latin Hypercube Sampling Technique

B

Appendix-B

Python Script to find the Neural Network Architecture for Tile Flow Rate using Genetic Algorithm[1]

```python

# Import Libraries
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

import numpy as np
import pandas as pd

from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.layers import LeakyReLU

import random

print("Passed")

#Load Dataset
dataset = pd.read_csv('LessFeatureAnalysis.csv', sep=',')

# Input Varaibles
X                        = dataset.iloc[: ,0: 11]
dataset.iloc[: ,11:41] = dataset.iloc[: ,11:41 ] *0.000472

# Output Variable
Y                        = dataset.iloc[: ,11:26]

X_data_columns           = X.columns
Y_data_columns           = Y.columns


test_split                         = 0.2
X_train, X_test, Y_train, Y_test = train_test_split(X, Y , shuffle=True,
                                                    random_state=9 ,test_size=
    test_split)

Scaler                   = MinMaxScaler(feature_range=(0, 1))
```

[1]The procedure to implement the genetic algorithm to find the hyper-parameters of the neural network is inspired from the following references [6] and [7].

```
X_Scaler               = Scaler.fit(X_train)
X_train1               = X_Scaler.transform(X_train)
X_test1                = X_Scaler.transform(X_test)

X_train1        = pd.DataFrame(data =X_train1, columns = X_data_columns)
X_test1         = pd.DataFrame(data =X_test1, columns  = X_data_columns)

#Round to 2 decimal place
X_train1        = X_train1.round(2)
X_test1         = X_test1.round(2)

Y_train        = Y_train.round(2)
Y_test         = Y_test.round(2)

population           = 20                     # Population of each generation
generations          = 10                     # Number of Generations

threshold            = 0.99                   # Validation accuracy threshold
epochs               = 10000
chance_to_mutate     = 0.05                   # Chances of mutation
fit_chromosomes      = 0.25                   # Number of fit chromosomes to take
feable_chromosomes   = 0.15                   # Number of unfit chromosomes to take
chromosomes_to_mutate = 1 - fit_chromosomes - feable_chromosomes


def r_square(y_true, y_pred):
    from keras import backend as Ks
    SS_res = Ks.sum(Ks.square(y_true - y_pred))
    SS_tot = Ks.sum(Ks.square(y_true - Ks.mean(y_true)))
    return (1 - SS_res / (SS_tot + Ks.epsilon()))


def serve_model( units1, dr1, units2, dr2,
                 xtrain, ytrain, xtest, ytest,
                 batch_size, summary=False):

    print(" Units1:  "+str(units1)+" Dr1: "+str(dr1)+
          " Units2:  "+str(units2)+" Dr2: "+str(dr2)+
          " Batch_size:  "+str(batch_size))
    #adam  = optimizers.adam(lr=lr)
    model = Sequential()
    model.add(Dense(units1, input_shape=[11,]))
    model.add(LeakyReLU())
    model.add(Dropout(dr1))
    model.add(Dense(units2))
    model.add(LeakyReLU())
    model.add(Dropout(dr2))
    model.add(Dense(15, activation='linear'))
    model.compile(loss='mean_squared_error', optimizer='adam', metrics=[r_square])
    if summary:
        model.summary()

    model.fit(xtrain, ytrain, batch_size=batch_size, epochs=epochs,
              validation_data=(xtest, ytest), shuffle=True,  verbose=0)

    return model

class Network():
    def __init__(self):

        #self._layers        = np.random.choice([1,2,3,4,5])
        self._units1        = np.random.randint(1, 500)
        self._units2        = np.random.randint(1, 500)
        self._dr1           = np.random.random()
        self._dr2           = np.random.random()

        #self._lr             = random.choice([0.001, 0.0001, 0.00001])
        self._batch_size    = random.choice([4,8,16,32, 64, 128])
        self._accuracy = 0

    def init_hyperparams(self):
        hyperparams = {
            #'layers'    :self._layers
            'units1'    : self._units1,
            'dr1'       : self._dr1,
            'units2'    : self._units2,
            'dr2'       : self._dr2,
            #'lr'        : self._lr,
            'batch_size': self._batch_size
        }
        return hyperparams

def init_networks(population):
    print("initialize")
    return [Network() for _ in range(population)]

def fitness(networks):
    print("fitness")
    for network in networks:
        hyperparams = network.init_hyperparams()
        #layers     = hyperparams['layers']
        units1     = hyperparams['units1']
        dr1        = hyperparams['dr1']
        units2     = hyperparams['units2']
        dr2        = hyperparams['dr2']
        #lr         = hyperparams['lr']
        batch_size = hyperparams['batch_size']
        try:
            model = serve_model( units1, dr1, units2, dr2,
                                 X_train1, Y_train, X_test1,
                                 Y_test, batch_size)
            accuracy = model.evaluate(X_test1, Y_test, verbose=0)[1]
            network._accuracy = accuracy
            print('Accuracy: {}'.format(network._accuracy))
        except:
            network._accuracy = 0
            print('Build failed.')

    return networks

def selection(networks):
    print("Selection")
    networks = sorted(networks, key=lambda network: network._accuracy, reverse=
    True)
    temp     = networks
    networks = networks[:int(fit_chromosomes * len(networks))]
    print("Top Selection:   ", len(networks))

    lower_rank = random.sample(population=temp[int(fit_chromosomes * len(temp)):],
     k=int(feable_chromosomes*len(temp)))
    networks.extend(lower_rank)
    print("Low Selection:   ", len(networks))
    return networks

def crossover(networks):
    print("crossover")
    offspring = []
    print("Number of babies to produce: ", int((population - len(networks))))
    while len(offspring) <  int((population - len(networks))):
        length  = len(networks)-1
        male    = random.randint(0, length)
        female  = random.randint(0, length)

        if male !=female:
            male    = networks[male]
            female  = networks[female]

            child1   = Network()
            child2   = Network()

            child1._units1     = male._units1
            child1._dr1        = female._dr1
            child1._units2     = male._units2
            child1._dr2        = female._dr2
            #child1._lr          = female._lr
            child1._batch_size = male._batch_size
            offspring.append(child1)

            child2._units1     = female._units1
            child2._dr1        = male._dr1
            child2._units2     = female._units2
            child2._dr2        = male._dr2
            #child2._lr          = male._lr
            child2._batch_size = female._batch_size
            offspring.append(child2)

    networks.extend(offspring)
    print("crossover:   ", len(networks))
    return networks

def mutate(networks):
    print("mutate")

    mutate_network = random.sample(population=list(enumerate(networks[int(
    chromosomes_to_mutate * len(networks))-1:])),
                                k=int(chance_to_mutate * len(networks)))
    for index, network in mutate_network:
        print("Index:   ", index)
        print("*****************************************************************")
        print(network)
        print("*****************************************************************")
        hyperpara = network.init_hyperparams()
        mutation  = random.choice(list(hyperpara.keys()))

        if mutation             == 'units1':
            print("units1")
            network._units1     = hyperpara[mutation]
        elif mutation           == 'dr1':
            print("dr1")
            network._dr1        = hyperpara[mutation]
        elif mutation           == 'units2':
            print("units2")
            network._units2     = hyperpara[mutation]
        elif mutation           == 'dr2':
            print("dr2")
            network._dr2        = hyperpara[mutation]
        # elif  mutation          == 'lr':
        #     print("lr")
        #     network._lr         = hyperpara[mutation]
        elif mutation           =='batch_size':
            print("batch_size")
            network._batch_size = hyperpara[mutation]

        networks[index+8] = network
        print(networks[index])
    return networks

def main():
    networks = init_networks(population)

    for gen in range(generations):
        print('Generation {}'.format(gen + 1))

        networks = fitness(networks)
        networks = selection(networks)
        networks = crossover(networks)
        networks = mutate(networks)

        for network in networks:
            if network._accuracy > threshold:
                print('Threshold met')
                print(network.init_hyperparams())
                print('Best accuracy: {}'.format(network._accuracy))
                exit(0)

    networks = sorted(networks, key=lambda network: network._accuracy, reverse=
    True)
    for network in networks:
        print("*****************************************************************")
        print("Hyperparameters: ")
        print(network.init_hyperparams())
        print("Accuracy       : {}".format(network._accuracy))
        print("*****************************************************************")

if __name__ == '__main__':
    main()
```

Listing B.1: Neural Network Architecture for Tile Flow Rate using Genetic Algorithm.

C

Appendix-C

Python Script to find the Neural Network Architecture for Cabinet Temperature using Genetic Algorithm[1]

```

# Import Libraries

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler

import numpy as np
import pandas as pd

from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras import optimizers
from keras.layers import LeakyReLU

import random

print("Passed")

#Load Dataset
dataset = pd.read_csv('LessCabinetTemp-II.csv', sep=',')

# Input Varaibles
X                     = dataset.iloc[: ,0: 11]
dataset.iloc[: ,11:41] = dataset.iloc[: ,11:41 ] *0.000472

# Output Variable
Y                     = dataset.iloc[: ,56:71]

X_data_columns        = X.columns
Y_data_columns        = Y.columns

Scaler                = MinMaxScaler(feature_range=(0, 1))
# X_Scaler              = Scaler.fit(X_train)
# X_train1              = X_Scaler.transform(X_train)
X1                    = Scaler.fit_transform(X)
```

[1]The procedure to implement the genetic algorithm to find the hyper-parameters of the neural network is inspired from the following references [6] and [7].

```
#Y1                     = Scaler.fit_transform(Y)

test_split                        = 0.2
X_train, X_test, Y_train, Y_test = train_test_split(X1, Y , shuffle=True,
                                                   random_state=9 ,test_size=
    test_split)

# Scaler                 = MinMaxScaler(feature_range=(0, 1))
# X_Scaler               = Scaler.fit(X_train)
# X_train1               = X_Scaler.transform(X_train)
# X_test1                = X_Scaler.transform(X_test)

#Normalize Y
# Standard_Scaler = StandardScaler()
# Y_Standard      = Standard_Scaler.fit(Y_train)
# Y_train1        = Y_Standard.transform(Y_train)
# Y_test1         = Y_Standard.transform(Y_test)

X_train1        = pd.DataFrame(data =X_train, columns = X_data_columns)
X_test1         = pd.DataFrame(data =X_test, columns  = X_data_columns)

Y_train1        = pd.DataFrame(data=Y_train, columns=Y_data_columns)
Y_test1         = pd.DataFrame(data=Y_test, columns=Y_data_columns)

#Round to 2 decimal place
# X_train1        = X_train1.round(2)
# X_test1         = X_test1.round(2)

# Y_train        = Y_train.round(2)
# Y_test         = Y_test.round(2)

population             = 20                    # Population of each generation
generations            = 10                    # Number of Generations

threshold              = 0.95                  # Validation accuracy threshold
epochs                 = 5000
chance_to_mutate       = 0.05                  # Chances of mutation
fit_chromosomes        = 0.25                  # Number of fit chromosomes to take
feable_chromosomes     = 0.15                  # Number of unfit chromosomes to take

# Choose from the set to child population for mutation
chromosomes_to_mutate = 1 - fit_chromosomes - feable_chromosomes


def r_square(y_true, y_pred):
    from keras import backend as Ks
    SS_res = Ks.sum(Ks.square(y_true - y_pred))
    SS_tot = Ks.sum(Ks.square(y_true - Ks.mean(y_true)))
    return (1 - SS_res / (SS_tot + Ks.epsilon()))



def serve_model( units1, dr1,units2, dr2, units3, dr3,units4, dr4,
                 lr, xtrain, ytrain, xtest, ytest, batch_size, summary=False):

    print(" Units1:  "+str(units1)+" Dr1: "+str(dr1)+" Units2:  "+str(units2)+"
    Dr2: "+str(dr2)+
          " Units3:  "+str(units3)+" Dr3: "+str(dr3)+" Units4:  "+str(units4)+"
    Dr4: "+str(dr4)+
          " Lr: "+str(lr)+" Batch_size:  "+str(batch_size))
    adam  = optimizers.adam(lr=lr)
    model = Sequential()
    model.add(Dense(units1, input_shape=[11,]))
    model.add(LeakyReLU())
    model.add(Dropout(dr1))
    model.add(Dense(units2))
    model.add(LeakyReLU())
    model.add(Dropout(dr2))
    model.add(Dense(units3))
    model.add(LeakyReLU())
    model.add(Dropout(dr3))
    model.add(Dense(units4))
    model.add(LeakyReLU())
    model.add(Dropout(dr4))
    model.add(Dense(15, activation='linear'))
    model.compile(loss='mean_squared_error', optimizer=adam, metrics=[r_square])
    if summary:
        model.summary()

    model.fit(xtrain, ytrain, batch_size=batch_size, epochs=epochs,
              validation_data=(xtest, ytest), shuffle=True,  verbose=0)

    return model


class Network():
    def __init__(self):

        #self._layers        = np.random.choice([1,2,3,4,5])
        self._units1        = np.random.randint(1, 500)
        self._units2        = np.random.randint(1, 500)
        self._units3        = np.random.randint(1, 500)
        self._units4        = np.random.randint(1, 500)

        self._dr1           = np.random.random()
        self._dr2           = np.random.random()
        self._dr3           = np.random.random()
        self._dr4           = np.random.random()

        self._lr            = random.choice([0.001, 0.0005, 0.0001, 0.00001])
        self._batch_size    = random.choice([4, 8, 16, 32, 64, 128, 256])

        self._accuracy = 0

    def init_hyperparams(self):
        hyperparams = {
            #'layers'    :self._layers
            'units1'    : self._units1,
            'dr1'       : self._dr1,
            'units2'    : self._units2,
            'dr2'       : self._dr2,
            'units3'    : self._units3,
            'dr3'       : self._dr3,
            'units4'    : self._units4,
            'dr4'       : self._dr4,
            'lr'        : self._lr,
            'batch_size': self._batch_size
        }
        return hyperparams


def init_networks(population):
    print("initialize")
    return [Network() for _ in range(population)]


def fitness(networks):
    print("fitness")
    for network in networks:
        hyperparams = network.init_hyperparams()
        #layers     = hyperparams['layers']
        units1     = hyperparams['units1']
        dr1        = hyperparams['dr1']
        units2     = hyperparams['units2']
        dr2        = hyperparams['dr2']
        units3     = hyperparams['units3']
        dr3        = hyperparams['dr3']
        units4     = hyperparams['units4']
        dr4        = hyperparams['dr4']
        lr         = hyperparams['lr']
        batch_size = hyperparams['batch_size']
        try:
            model = serve_model( units1, dr1, units2, dr2,
                                 units3, dr3, units4, dr4,
                                 lr, X_train1, Y_train1,
                                 X_test1, Y_test1, batch_size)
            accuracy = model.evaluate(X_test1, Y_test1, verbose=0)[1]
            network._accuracy = accuracy
            print('Accuracy: {}'.format(network._accuracy))
        except:
            network._accuracy = 0
            print('Build failed.')

    return networks


def selection(networks):
    print("Selection")
    networks = sorted(networks, key=lambda network: network._accuracy, reverse=
    True)
    temp     = networks
    networks = networks[:int(fit_chromosomes * len(networks))]
    print("Top Selection:   ", len(networks))

    lower_rank = random.sample(population=temp[int(fit_chromosomes * len(temp)):],
     k=int(feable_chromosomes*len(temp)))

    networks.extend(lower_rank)
    print("Low Selection:   ", len(networks))
    return networks


def crossover(networks):
    print("crossover")
    offspring = []
    print("Number of babies to produce: ", int((population - len(networks))))
    while len(offspring) <  int((population - len(networks))):
        length  = len(networks)-1
        male    = random.randint(0, length)
        female  = random.randint(0, length)

        if male !=female:
            male    = networks[male]
            female  = networks[female]

            child1   = Network()
            child2   = Network()

            child1._units1     = male._units1
            child1._dr1        = male._dr1
            child1._units2     = male._units2
            child1._dr2        = male._dr2
            child1._units3     = male._units3

            child1._dr3        = female._dr3
            child1._units4     = female._units4
            child1._dr4        = female._dr4
            child1._lr         = female._lr
            child1._batch_size = female._batch_size
            offspring.append(child1)

            child2._units1     = female._units1
            child2._dr1        = female._dr1
            child2._units2     = female._units2
            child2._dr2        = female._dr2
            child2._units3     = female._units3

            child2._dr3        = male._dr3
            child2._units4     = male._units4
            child2._dr4        = male._dr4
            child2._lr         = male._lr
            child2._batch_size = male._batch_size
            offspring.append(child2)

    networks.extend(offspring)
    print("crossover:   ", len(networks))
    return networks

def mutate(networks):
    print("mutate")

    mutate_network = random.sample(population=list(enumerate(networks[int(
    chromosomes_to_mutate * len(networks))-1:])),
                                   k=int(chance_to_mutate * len(networks)))
    for index, network in mutate_network:
        print("Index:   ", index)
        print("****************************************************************")
        print(network)
        print("****************************************************************")
        hyperpara = network.init_hyperparams()
        mutation  = random.choice(list(hyperpara.keys()))

        if mutation             == 'units1':
            print("units1")
            network._units1     = hyperpara[mutation]
        elif mutation           == 'dr1':
            print("dr1")
            network._dr1        = hyperpara[mutation]
        elif mutation           == 'units2':
             print("units2")
             network._units2     = hyperpara[mutation]
        elif mutation           == 'dr2':
             print("dr2")
             network._dr2        = hyperpara[mutation]
        elif mutation           == 'units3':
            print("units3")
            network._units3     = hyperpara[mutation]
        elif mutation           == 'dr3':
            print("dr3")
            network._dr3        = hyperpara[mutation]
        elif mutation           == 'units4':
            print("units4")
            network._units4     = hyperpara[mutation]
        elif mutation           == 'dr4':
            print("dr4")
            network._dr4        = hyperpara[mutation]
        elif  mutation          == 'lr':
            print("lr")
            network._lr         = hyperpara[mutation]
        elif mutation           =='batch_size':
            print("batch_size")
            network._batch_size = hyperpara[mutation]

        networks[index+8] = network
        print(networks[index])
    return networks

def main():
    networks = init_networks(population)

    for gen in range(generations):
        print('Generation {}'.format(gen + 1))

        networks = fitness(networks)
        networks = selection(networks)
        networks = crossover(networks)
        networks = mutate(networks)

        for network in networks:
            if network._accuracy > threshold:
                print('Threshold met')
                print(network.init_hyperparams())
                print('Best accuracy: {}'.format(network._accuracy))
                exit(0)

    networks = sorted(networks, key=lambda network: network._accuracy, reverse=
    True)
    for network in networks:
        print("*****************************************************************")
        print("Hyperparameters: ")
        print(network.init_hyperparams())
        print("Accuracy       : {}".format(network._accuracy))
        print("*****************************************************************")


if __name__ == '__main__':
    main()
```

Listing C.1: Neural Network Architecture for Cabinet Temperature using Genetic Algorithm

D

Appendix-D

Python Script for the Neural Network Predicting the Tileflow Rate of Row X

```

# Import Libraries

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

import pandas as pd
import matplotlib.pyplot as plt

from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras import optimizers
from keras.layers import LeakyReLU
from keras.callbacks import ModelCheckpoint
print("Passed")

#Load Dataset
dataset = pd.read_csv('LessFeatureAnalysis.csv', sep=',')

# Input Varaibles
X                       = dataset.iloc[: ,0: 11]
dataset.iloc[: ,11:41] = dataset.iloc[: ,11:41 ] *0.000472

# Output Variable
Y                       = dataset.iloc[: ,11:26]

X_data_columns          = X.columns
Y_data_columns          = Y.columns

Scaler                  = MinMaxScaler(feature_range=(0, 1))
X1                      = Scaler.fit_transform(X)

test_split                        = 0.2
X_train, X_test, Y_train, Y_test = train_test_split(X1, Y , shuffle=True,
                                                   random_state=9 ,test_size=
    test_split)


X_train1        = pd.DataFrame(data =X_train, columns = X_data_columns)
```

```
X_test1          = pd.DataFrame(data =X_test, columns  = X_data_columns)

Y_train1         = pd.DataFrame(data =Y_train, columns = Y_data_columns)
Y_test1          = pd.DataFrame(data =Y_test, columns  = Y_data_columns)

# coefficient of determination (R^2) for regression
def r_square(y_true, y_pred):
    from keras import backend as Ks
    SS_res  = Ks.sum(Ks.square(y_true - y_pred))
    SS_tot  = Ks.sum(Ks.square(y_true - Ks.mean(y_true)))
    return (1 - SS_res / (SS_tot + Ks.epsilon()))

n_batchsize = 128            # Size of each batch 64
lr          = 0.0005         # Learning rate for optimizer

epochs      = 100000         # No. of each epochs
verbose     = 2              # Type of display of the accuracy and loss of the
    network

def NN_baseline(epochs, lr):
    adam  = optimizers.adam(lr=lr)

    model = Sequential([

        Dense(236, input_shape=(11,)),
        LeakyReLU(),
        Dropout(0.29780690374246566),
        Dense(188),
        LeakyReLU(),
        Dropout(0.0054370255928052336),

        Dense(15, activation='linear')
    ])

    model.compile(loss='mean_squared_error', optimizer=adam, metrics=[r_square])

    # Checkpoint
    filepath       = "Tile-X.best.hdf5"
    checkpoint     =  ModelCheckpoint(filepath, monitor='val_r_square', verbose=1,
     save_best_only=True, mode='max')
    callbacks_list = [checkpoint]
    history        = model.fit(X_train1, Y_train1, epochs=epochs, batch_size=
    n_batchsize,
                               validation_data=(X_test1, Y_test1), shuffle=True,
    verbose=verbose,
                               callbacks=callbacks_list)

    return model, history


model, history = NN_baseline(epochs, lr)
print(model.summary())

plt.figure(1)
plt.plot(history.history['r_square'])
plt.plot(history.history['val_r_square'])
plt.title('Accuracy (Row X)- Tile Flow Rate $[m^3/s]$')
plt.ylabel('Accuracy $[R^2]$')
plt.xlabel('Epochs')
plt.ylim(0, 1)
plt.legend(['Train', 'Test'], loc='lower right')
plt.grid()

plt.figure(2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Loss (Row X)- Tile Flow Rate $[m^3/s]$')
plt.ylabel('Loss $[MSE]$')
plt.xlabel('Epochs')
plt.legend(['Train', 'Test'], loc='upper right')
plt.grid()
plt.show()
```

Listing D.1: (Row X) - Tile Flow Rate Prediction

E

Appendix-E

Python Script for the Neural Network Predicting the Tileflow Rate of Row Y

```

# Import Libraries
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

import pandas as pd
import matplotlib.pyplot as plt

from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras import optimizers
from keras.layers import LeakyReLU
from keras.callbacks import ModelCheckpoint
print("Passed")

#Load Dataset
dataset = pd.read_csv('LessFeatureAnalysis.csv', sep=',')

# Input Varaibles
X                       = dataset.iloc[: ,0: 11]
dataset.iloc[: ,11:41] = dataset.iloc[: ,11:41 ] *0.000472

# Output Variable
Y                       = dataset.iloc[: ,26:41]

X_data_columns          = X.columns
Y_data_columns          = Y.columns

Scaler                  = MinMaxScaler(feature_range=(0, 1))
X1                      = Scaler.fit_transform(X)

test_split                        = 0.2
X_train, X_test, Y_train, Y_test = train_test_split(X1, Y , shuffle=True,
                                                    random_state=9 ,test_size=
    test_split)


X_train1         = pd.DataFrame(data =X_train, columns = X_data_columns)
X_test1          = pd.DataFrame(data =X_test, columns  = X_data_columns)
```

```

Y_train1        = pd.DataFrame(data =Y_train, columns = Y_data_columns)
Y_test1         = pd.DataFrame(data =Y_test, columns  = Y_data_columns)

# coefficient of determination (R^2) for regression
def r_square(y_true, y_pred):
    from keras import backend as Ks
    SS_res  = Ks.sum(Ks.square(y_true - y_pred))
    SS_tot  = Ks.sum(Ks.square(y_true - Ks.mean(y_true)))
    return (1 - SS_res / (SS_tot + Ks.epsilon()))

n_batchsize = 128            # Size of each batch 64
lr          = 0.0005         # Learning rate for optimizer

epochs      = 100000         # No. of each epochs
verbose     = 2              # Type of display of the accuracy and loss of the
    network

def NN_baseline(epochs, lr):
    adam  = optimizers.adam(lr=lr)

    model = Sequential([

        Dense(236, input_shape=(11,)),
        LeakyReLU(),
        Dropout(0.29780690374246566),
        Dense(188),
        LeakyReLU(),
        Dropout(0.0054370255928052336),

        Dense(15, activation='linear')
    ])

    model.compile(loss='mean_squared_error', optimizer=adam, metrics=[r_square])

    # Checkpoint
    filepath       = "Tile-Y.best.hdf5"
    checkpoint     =  ModelCheckpoint(filepath, monitor='val_r_square', verbose=1,
     save_best_only=True, mode='max')
    callbacks_list = [checkpoint]
    history        = model.fit(X_train1, Y_train1, epochs=epochs, batch_size=
    n_batchsize,
                               validation_data=(X_test1, Y_test1), shuffle=True,
    verbose=verbose,
                               callbacks=callbacks_list)

    return model, history

model, history = NN_baseline(epochs, lr)
print(model.summary())

plt.figure(1)
plt.plot(history.history['r_square'])
plt.plot(history.history['val_r_square'])
plt.title('Accuracy (Row Y)- Tile Flow Rate $[m^3/s]$')
plt.ylabel('Accuracy $[R^2]$')
plt.xlabel('Epochs')
plt.ylim(0, 1)
plt.legend(['Train', 'Test'], loc='lower right')
plt.grid()

```

```python
plt.figure(2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Loss (Row Y)- Tile Flow Rate $[m^3/s]$')
plt.ylabel('Loss $[MSE]$')
plt.xlabel('Epochs')
plt.legend(['Train', 'Test'], loc='upper right')
plt.grid()
plt.show()
```

Listing E.1: (Row Y) - Tile Flow Rate Prediction

F
Appendix-F

Python Script for the Neural Network Predicting the Cabinet Temperature of Row W

```python

# Import Libraries
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

import pandas as pd
import matplotlib.pyplot as plt

from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras import optimizers
from keras.layers import LeakyReLU

from keras.callbacks import ModelCheckpoint

print(”Passed”)

#Load Dataset
dataset = pd.read_csv('LessFeatureAnalysis.csv', sep=',')

# Input Varaibles
X                        = dataset.iloc[: ,0: 11]
dataset.iloc[: ,11:41]   = dataset.iloc[: ,11:41 ] *0.000472

# Output Variable
Y                        = dataset.iloc[: ,41:56]

X_data_columns          = X.columns
Y_data_columns          = Y.columns

Scaler                  = MinMaxScaler(feature_range=(0, 1))
X1                      = Scaler.fit_transform(X)

test_split                        = 0.2
X_train, X_test, Y_train, Y_test = train_test_split(X1, Y , shuffle=True,
                                                    random_state=9 ,test_size=
    test_split)

X_train1        = pd.DataFrame(data =X_train, columns = X_data_columns)
```

```
X_test1         = pd.DataFrame(data =X_test, columns  = X_data_columns)

Y_train1        = pd.DataFrame(data=Y_train, columns=Y_data_columns)
Y_test1         = pd.DataFrame(data=Y_test, columns=Y_data_columns)

# coefficient of determination (R^2) for regression
def r_square(y_true, y_pred):
    from keras import backend as Ks
    SS_res = Ks.sum(Ks.square(y_true - y_pred))
    SS_tot = Ks.sum(Ks.square(y_true - Ks.mean(y_true)))
    return (1 - SS_res / (SS_tot + Ks.epsilon()))

n_batchsize = 128             # Size of each batch
lr          = 0.0005          # Learning rate for optimizer
epochs      = 50000           # No. of each epochs
verbose     = 2               # Type of display of the accuracy and loss of the
    network

def NN_baseline(epochs, lr):
    adam    = optimizers.adam(lr=lr)
    adamax = optimizers.adamax(lr=0.002)
    model = Sequential([

        Dense(236, input_shape=(11,)),
        LeakyReLU(),
        Dropout(0.29780690374246566),
        Dense(250),
        LeakyReLU(),
        Dropout(0.1),
        Dense(300),
        LeakyReLU(),
        Dropout(0.1),
        Dense(15, activation='linear')
    ])

    model.compile(loss='mean_absolute_error', optimizer=adam, metrics=[r_square])

    # Checkpoint
    filepath       = "Cabinet-W.best.hdf5"

    checkpoint     =  ModelCheckpoint(filepath, monitor='val_r_square', verbose=1,
     save_best_only=True, mode='max')
    callbacks_list = [checkpoint]
    history        = model.fit(X_train1, Y_train1, epochs=epochs, batch_size=
    n_batchsize,
                               validation_data=(X_test1, Y_test1), shuffle=True,
    verbose=verbose,
                               callbacks=callbacks_list)

    return model, history

model, history = NN_baseline(epochs, lr)
print(model.summary())

plt.figure(1)
plt.plot(history.history['r_square'])
plt.plot(history.history['val_r_square'])
plt.title('Accuracy (Row W)- Cabinet Temperature $[^o C]$')
plt.ylabel('Accuracy $[R^2]$')
plt.xlabel('Epochs')
plt.ylim(0, 1)
```

```python
plt.legend(['Train', 'Test'], loc='lower right')
plt.grid()

plt.figure(2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Loss (Row W)- Cabinet Temperature $[^o C]$')
plt.ylabel('Loss $[MSE]$')
plt.xlabel('Epochs')
plt.legend(['Train', 'Test'], loc='upper right')
plt.grid()
plt.show()
```

Listing F.1: (Row W) - Cabinet Temperature Prediction

G

Appendix-G

Python Script for the Neural Network Predicting the Cabinet Temperature of Row Z

```python

# Import Libraries
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

import pandas as pd
import matplotlib.pyplot as plt

from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras import optimizers
from keras.layers import LeakyReLU

from keras.callbacks import ModelCheckpoint

print("Passed")

#Load Dataset
dataset = pd.read_csv('LessFeatureAnalysis.csv', sep=',')

# Input Varaibles
X                        = dataset.iloc[: ,0: 11]
dataset.iloc[: ,11:41]   = dataset.iloc[: ,11:41 ] *0.000472

# Output Variable
Y                        = dataset.iloc[: ,56:71]

X_data_columns          = X.columns
Y_data_columns          = Y.columns

Scaler                  = MinMaxScaler(feature_range=(0, 1))
X1                      = Scaler.fit_transform(X)

test_split                        = 0.2
X_train, X_test, Y_train, Y_test = train_test_split(X1, Y , shuffle=True,
                                                   random_state=9 ,test_size=
    test_split)

X_train1        = pd.DataFrame(data =X_train, columns = X_data_columns)
```

```python
X_test1          = pd.DataFrame(data =X_test, columns  = X_data_columns)

Y_train1         = pd.DataFrame(data=Y_train, columns=Y_data_columns)
Y_test1          = pd.DataFrame(data=Y_test, columns=Y_data_columns)

# coefficient of determination (R^2) for regression
def r_square(y_true, y_pred):
    from keras import backend as Ks
    SS_res = Ks.sum(Ks.square(y_true - y_pred))
    SS_tot = Ks.sum(Ks.square(y_true - Ks.mean(y_true)))
    return (1 - SS_res / (SS_tot + Ks.epsilon()))

n_batchsize = 128             # Size of each batch
lr          = 0.0005          # Learning rate for optimizer
epochs      = 50000           # No. of each epochs
verbose     = 2               # Type of display of the accuracy and loss of the network

def NN_baseline(epochs, lr):
    adam    = optimizers.adam(lr=lr)
    adamax = optimizers.adamax(lr=0.002)
    model = Sequential([

        Dense(236, input_shape=(11,)),
        LeakyReLU(),
        Dropout(0.29780690374246566),
        Dense(250),
        LeakyReLU(),
        Dropout(0.1),
        Dense(300),
        LeakyReLU(),
        Dropout(0.1),
        Dense(15, activation='linear')
    ])

    model.compile(loss='mean_absolute_error', optimizer=adam, metrics=[r_square])

    # Checkpoint
    filepath       = "Cabinet-Z.best.hdf5"

    checkpoint     =  ModelCheckpoint(filepath, monitor='val_r_square', verbose=1, save_best_only=True, mode='max')
    callbacks_list = [checkpoint]
    history        = model.fit(X_train1, Y_train1, epochs=epochs, batch_size=n_batchsize,
                               validation_data=(X_test1, Y_test1), shuffle=True, verbose=verbose,
                               callbacks=callbacks_list)

    return model, history

model, history = NN_baseline(epochs, lr)
print(model.summary())

plt.figure(1)
plt.plot(history.history['r_square'])
plt.plot(history.history['val_r_square'])
plt.title('Accuracy (Row Z)- Cabinet Temperature $[^o C]$')
plt.ylabel('Accuracy $[R^2]$')
plt.xlabel('Epochs')
plt.ylim(0, 1)
plt.legend(['Train', 'Test'], loc='lower right')
plt.grid()

plt.figure(2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Loss (Row Z)- Cabinet Temperature $[^o C]$')
plt.ylabel('Loss $[MSE]$')
plt.xlabel('Epochs')
plt.legend(['Train', 'Test'], loc='upper right')
plt.grid()
plt.show()
```

Listing G.1: (Row Z) - Cabinet Temperature Prediction

H

Appendix-H

Python Script for Non-Dominated Sorting Genetic Algorithm-II to find the Optimum Input Design Parameters

```python

!pip install deap

# Import Libraries
import random as rn
import numpy as np
import pandas as pd

from deap import algorithms
from deap import base
from deap import creator
from deap import tools
#import history
import copy

from keras.models import load_model

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler

import matplotlib.pyplot as plt
import seaborn

IND_SIZE = 11 # Individual size

creator.create("FitnessMin", base.Fitness, weights=(-1.0,-1.0))
creator.create("Individual", list, fitness=creator.FitnessMin)

toolbox = base.Toolbox()
toolbox.register("attr_float", rn.uniform, 0, 1)

toolbox.register("individual", tools.initRepeat, creator.Individual, toolbox.
    attr_float, n=IND_SIZE)

toolbox.register("population", tools.initRepeat, list, toolbox.individual)

indii = toolbox.individual()

```

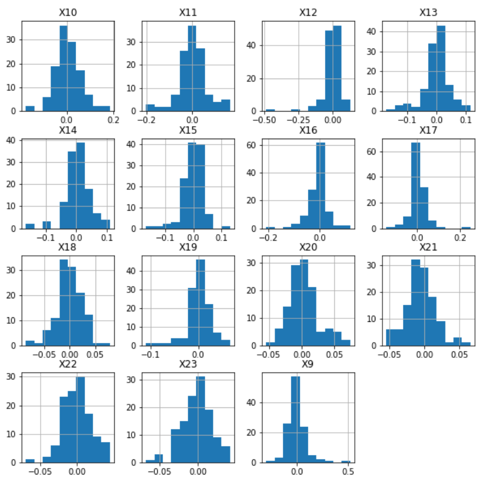
```

toolbox.register("mutate1", tools.mutGaussian,mu=0,sigma=0.1,  indpb=0.1)
toolbox.mutate1(indii)

i = toolbox.population(2)

def r_square(y_true, y_pred):
    from keras import backend as Ks
    SS_res = Ks.sum(Ks.square(y_true - y_pred))
    SS_tot = Ks.sum(Ks.square(y_true - Ks.mean(y_true)))
    return (1 - SS_res / (SS_tot + Ks.epsilon()))

modelX = load_model('Tile-X.best.hdf5', custom_objects={"r_square": r_square})
modelY = load_model('Tile-Y.best.hdf5', custom_objects={"r_square": r_square})
print("Passed")

def evaluate(individual):

    for i in range(0, len(individual)):
      if individual[i]<0 or individual[i]>1:
        print('True')
        individual[i]=rn.random()

    indi = np.transpose(individual)
    indi = pd.DataFrame(indi)
    indi = indi.T

    predictX = modelX.predict(indi)
    predictY = modelY.predict(indi)

    X_avg    = np.average(predictX)
    Y_avg    = np.average(predictY)

    X_std    = np.std(predictX)
    Y_std    = np.std(predictY)



    Average_diff = abs(((X_avg + Y_avg)/2) - 0.4011553325) # In m^3/s (850cfm)

    a = []
    a.append(X_std)
    a.append(Y_std)

    Standard_diff= abs(np.max(a) - 0.0)                     # In m^3/s 0.014766

    return Average_diff, Standard_diff

toolbox.register("evaluate", evaluate)
toolbox.register("mate", tools.cxOnePoint)
toolbox.register("mutate", tools.mutGaussian,mu=0.5,sigma=0.2,  indpb=0.01)
toolbox.register("select", tools.selNSGA2)
toolbox.parent_size = 2000
toolbox.child_size  = 1000
toolbox.max_gen     = 1
toolbox.mut_prob    = 0.01
toolbox.crx_prob    = 0.8

def main(toolbox, stats=None):
    rn.seed(64)

    pop = toolbox.population(n=toolbox.parent_size)
    pop = toolbox.select(pop, len(pop))

    hof = tools.ParetoFront()
    stats = tools.Statistics(lambda ind: ind.fitness.values)

    stats.register("avg", np.mean, axis=0)
    stats.register("std", np.std,  axis=0)
    stats.register("min", np.min,  axis=0)
    stats.register("max", np.max,  axis=0)



    POP, STATS = algorithms.eaMuPlusLambda(pop, toolbox, mu=toolbox.parent_size,
                                         lambda_=toolbox.child_size,
                                         cxpb=toolbox.crx_prob,
                                         mutpb=toolbox.mut_prob,
                                         ngen=toolbox.max_gen,
                                         stats=stats,
                                         verbose=True,
                                         halloffame=hof)

    return POP, STATS, hof

toolbox.max_gen     = 1

toolbox.max_gen     = 100

toolbox.max_gen     = 200

toolbox.max_gen     = 300

toolbox.max_gen     = 400

toolbox.max_gen     = 500

toolbox.max_gen     = 600


fronts1   = tools.emo.sortNondominated(HOF1, len(HOF1))

fronts100 = tools.emo.sortNondominated(HOF100, len(HOF100))

fronts200 = tools.emo.sortNondominated(HOF200, len(HOF200))

fronts300 = tools.emo.sortNondominated(HOF300, len(HOF300))

fronts400 = tools.emo.sortNondominated(HOF400, len(HOF400))

fronts500 = tools.emo.sortNondominated(HOF500, len(HOF500))

fronts600 = tools.emo.sortNondominated(HOF600, len(HOF600))

len(HOF600)

plot_colors = seaborn.color_palette("Set1", n_colors=15)
fig, ax = plt.subplots(1, figsize=(10,10))

for i, inds in enumerate(fronts1):
    par = [toolbox.evaluate(ind) for ind in inds]
    df = pd.DataFrame(par)
    df.plot(ax=ax, kind='scatter', label='Generation 1 ',
                 x=df.columns[0], y=df.columns[1],
                 color=plot_colors[0], marker='o')


plt.grid()
plt.xlabel("Difference of Average of TFR $[m^3/s]$")
plt.ylabel("Difference of Standard Deviation of TFR $[m^3/s]$")

plt.show()
plt.savefig('GenerationParetoFront-1.png')

hp1 = pd.DataFrame(HOF1)
hp100 = pd.DataFrame(HOF100)
hp200 = pd.DataFrame(HOF200)
hp300 = pd.DataFrame(HOF300)
hp400 = pd.DataFrame(HOF400)
hp500 = pd.DataFrame(HOF500)
hp600 = pd.DataFrame(HOF600)



hpv1= np.ones((hp1.shape[0],2))
hpv100= np.ones((hp100.shape[0],2))
hpv200= np.ones((hp200.shape[0],2))
hpv300= np.ones((hp300.shape[0],2))
hpv400= np.ones((hp400.shape[0],2))
hpv500= np.ones((hp500.shape[0],2))
hpv600= np.ones((hp600.shape[0],2))

hpv1[0] = np.array(HOF1[0].fitness.values)
hpv100[0] = np.array(HOF100[0].fitness.values)
hpv200[0] = np.array(HOF200[0].fitness.values)
hpv300[0] = np.array(HOF300[0].fitness.values)
hpv400[0] = np.array(HOF400[0].fitness.values)
hpv500[0] = np.array(HOF500[0].fitness.values)
hpv600[0] = np.array(HOF600[0].fitness.values)
# hpv600

for i in range(1,hp1.shape[0]):
  hpv1[i,:] = HOF1[i].fitness.values

for i in range(1,hp100.shape[0]):
  hpv100[i,:] = HOF100[i].fitness.values

for i in range(1,hp200.shape[0]):
  hpv200[i,:] = HOF200[i].fitness.values

for i in range(1,hp300.shape[0]):
  hpv300[i,:] = HOF300[i].fitness.values

for i in range(1,hp400.shape[0]):
  hpv400[i,:] = HOF400[i].fitness.values

for i in range(1,hp500.shape[0]):
  hpv500[i,:] = HOF500[i].fitness.values

for i in range(1,hp600.shape[0]):
  hpv600[i,:] = HOF600[i].fitness.values


```

```python
hpv1 = pd.DataFrame(hpv1)
hpv100 = pd.DataFrame(hpv100)
hpv200 = pd.DataFrame(hpv200)
hpv300 = pd.DataFrame(hpv300)
hpv400 = pd.DataFrame(hpv400)
hpv500 = pd.DataFrame(hpv500)
hpv600 = pd.DataFrame(hpv600)
# hpv600

hpv1.to_csv('Fitness1.csv')
hp1.to_csv('population1.csv')

hpv100.to_csv('Fitness100.csv')
hp100.to_csv('population100.csv')

hpv200.to_csv('Fitness200.csv')
hp200.to_csv('population200.csv')

hpv300.to_csv('Fitness300.csv')
hp300.to_csv('population300.csv')

hpv400.to_csv('Fitness400.csv')
hp400.to_csv('population400.csv')

hpv500.to_csv('Fitness500.csv')
hp500.to_csv('population500.csv')

hpv600.to_csv('Fitness600.csv')
hp600.to_csv('population600.csv')
```

Listing H.1: NSGA-II Algorithm to find the Optimum Input Design Parameters

Appendix-I

Absolute Error Statistics of Tile Flow Rate of each Floor Tiles present in Row X and Row Y


(a) Absolute Prediction Error in Row X

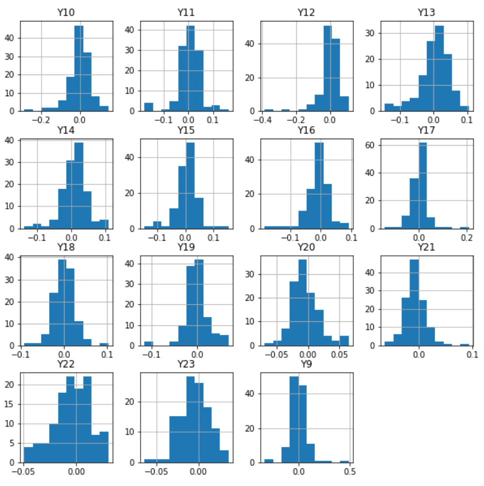
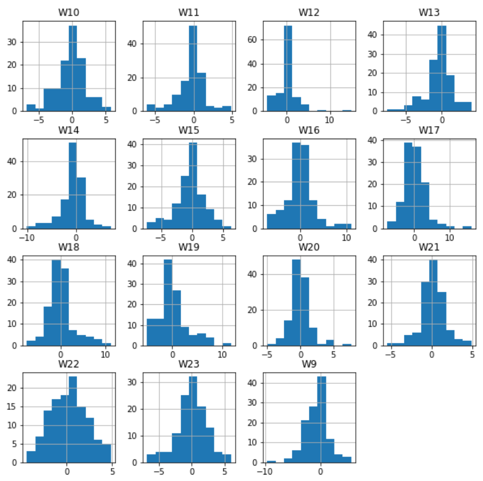
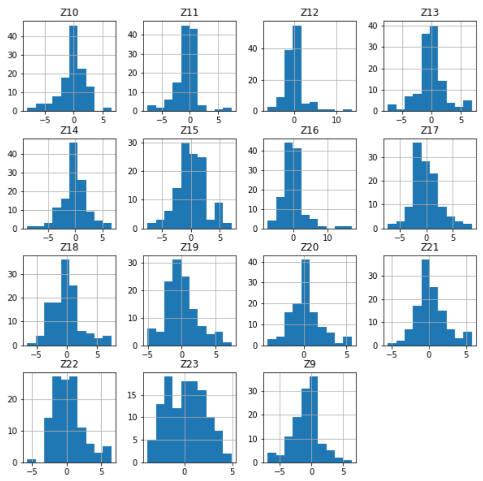

(b) Absolute Prediction Error in Row Y

Figure I.1: Absolute Tile Flow Rate Prediction Error $[m^3/s]$ of each Floor Tiles present in Row X and Row Y. Each Floor Tile Prediction contribute by a Different Amount to the Total Prediction Error. The Frequency of the Error is shown on the Vertical Axis vs the Absolute Error on the Horizontal Axis.

Absolute Error Statistics of Max. Cabinet Inlet Temperature of each Cabinets present in Row W and Row Z



(a) Absolute Prediction Error in Row W



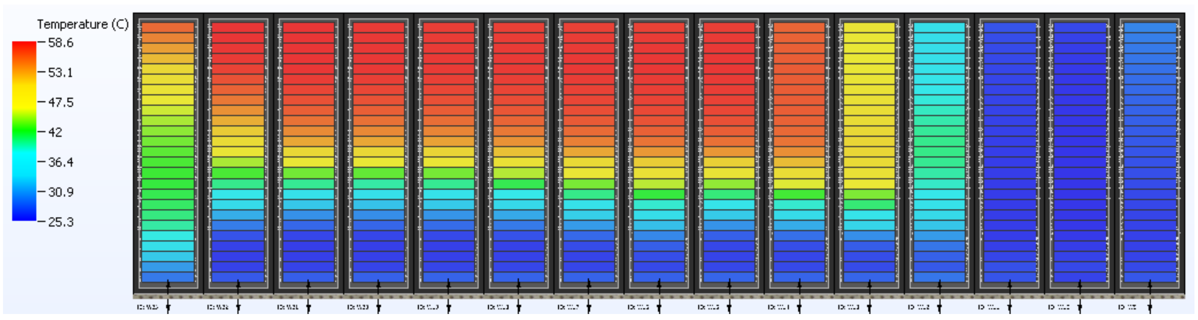
(b) Absolute Prediction Error in Row Z

Figure I.2: Absolute Max. Cabinet Temperature Prediction Error [$^{\circ}C$] of each Cabinets present in Row W and Row Z. Each Floor Tile Prediction contribute by a Different Amount to the Total Prediction Error. The Frequency of the Error is shown on the Vertical Axis vs the Absolute Error on the Horizontal Axis.

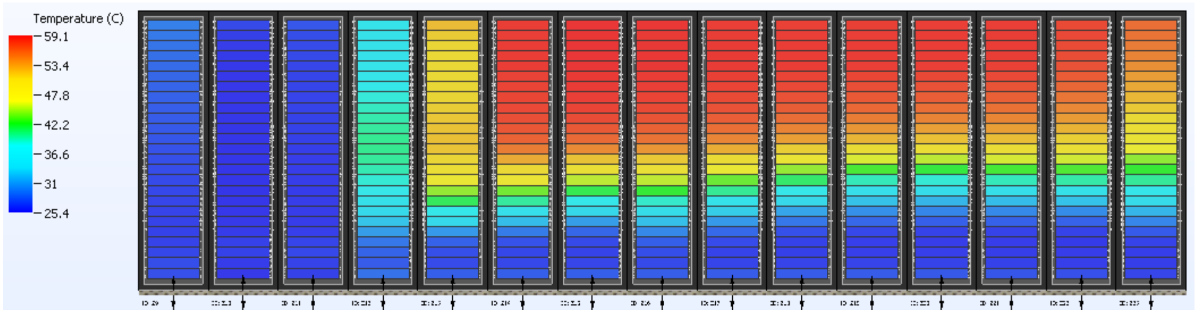
J

Appendix-J

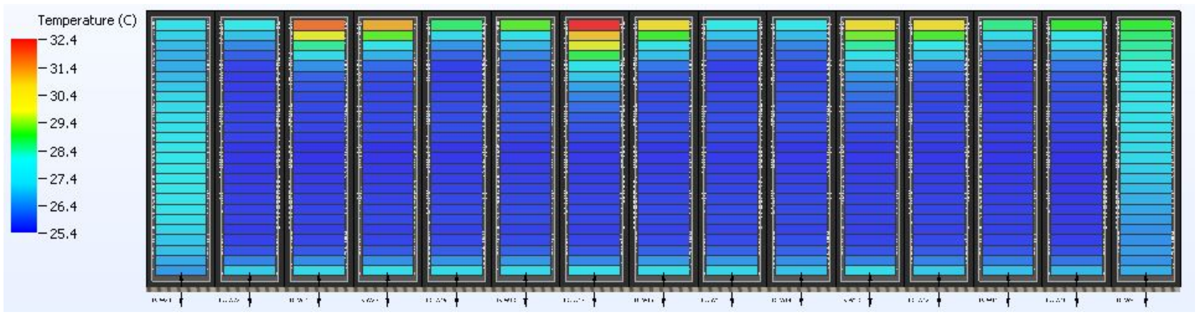
Comparison of Unbounded Server Mean Inlet Temperature from Pareto Front



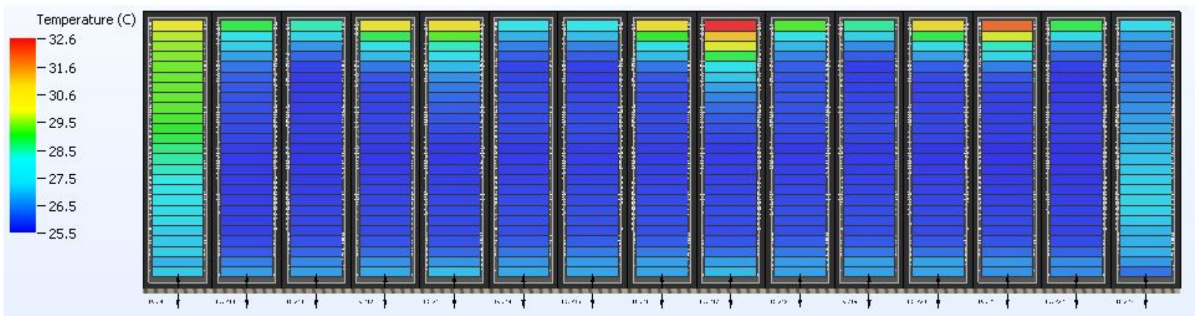
(a) Mean Server Inlet Temperature of row W (Black Zone)



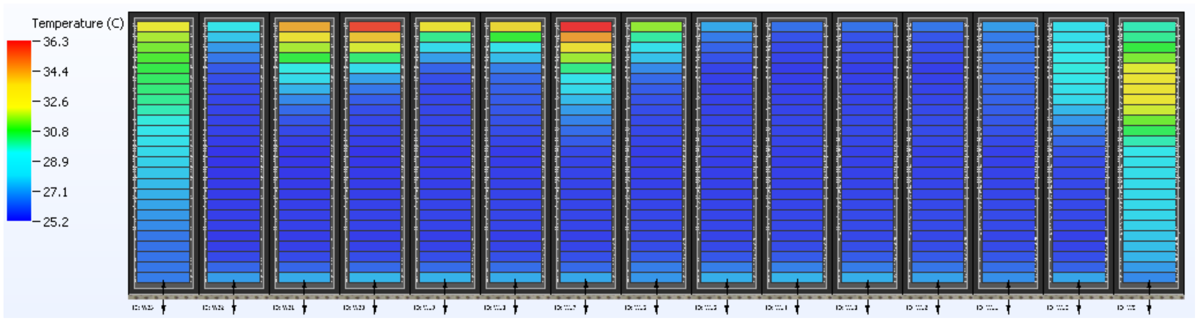
(b) Mean Server Inlet Temperature of row Z (Black Zone)



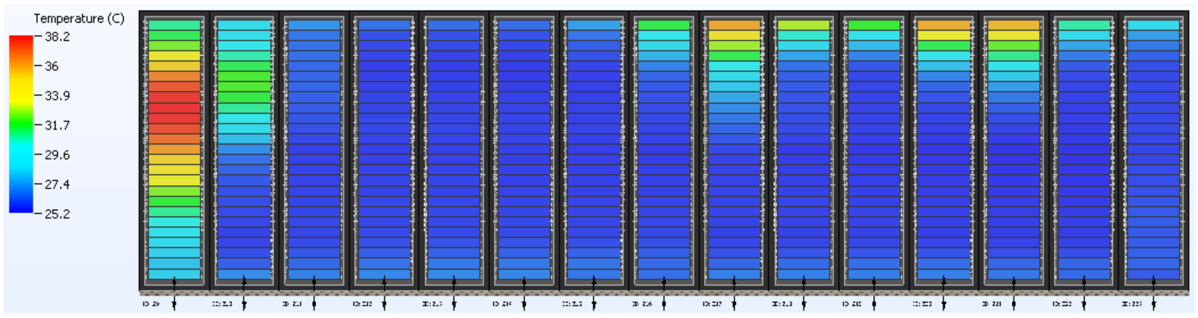
(c) Mean Server Inlet Temperature of row W (Orange Zone)



(d) Mean Server Inlet Temperature of row Z (Orange Zone)



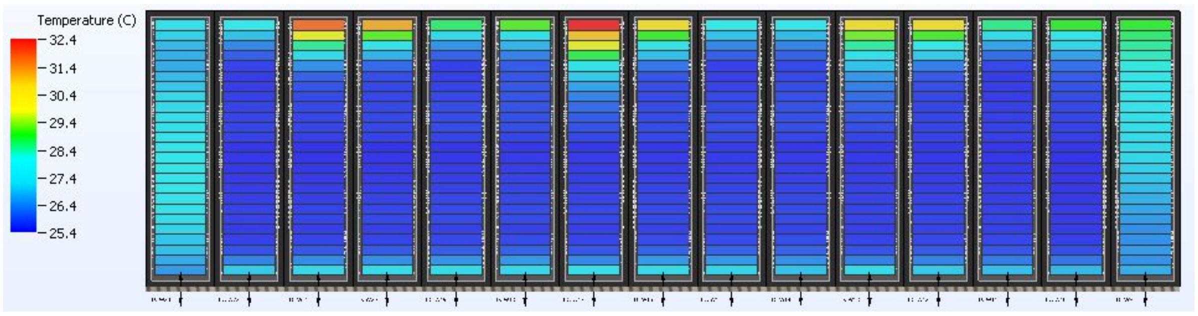
(e) Mean Server Inlet Temperature of row W (Green Zone)



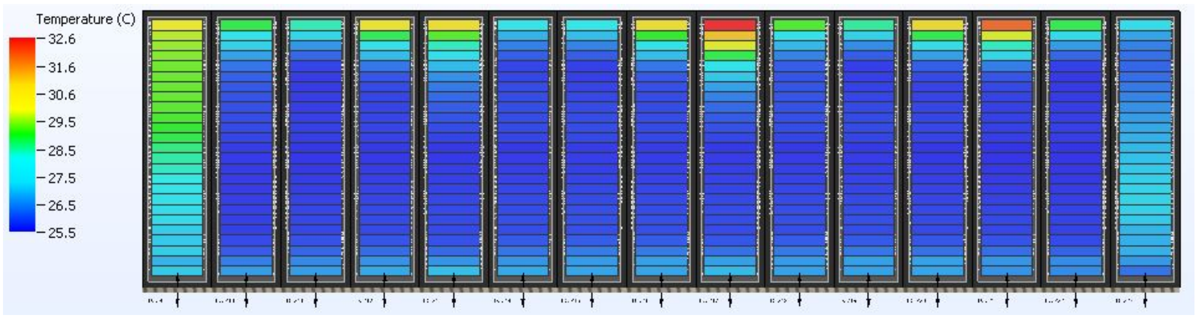
(f) Mean Server Inlet Temperature of row Z (Green Zone)

Figure J.1: The plot of mean server inlet temperature present in row W and row Z respectively. Solution points from the Pareto Front are taken from the black, orange and green zones and the best solution out of these Zones is taken for comparison. The actual mean inlet temperature of the servers is shown here. It is seen that the solution from the orange zone gives the better optimal solution as most of the server has a mean inlet temperature below $27^{\circ}C$ and very few beyond $32^{\circ}C$ as compared to the other Cases. Thus the solution shown in (c) and (d) can be considered as an optimal solution.

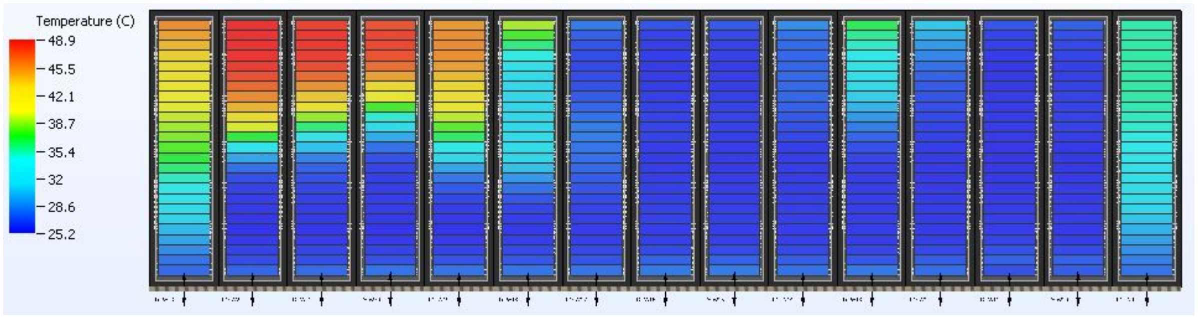
Comparison of Actual Mean Inlet Temperature of Servers in Optimized and Non-Optimized Cases



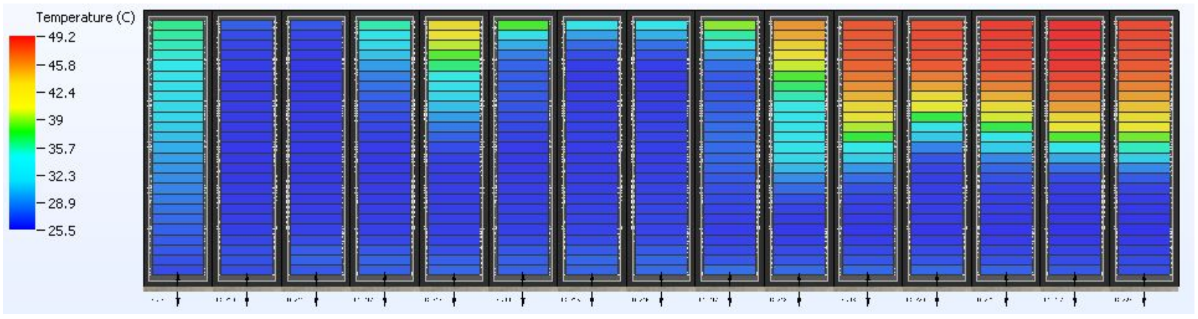
(a) Mean Server Inlet Temperature of row W (Optimized)



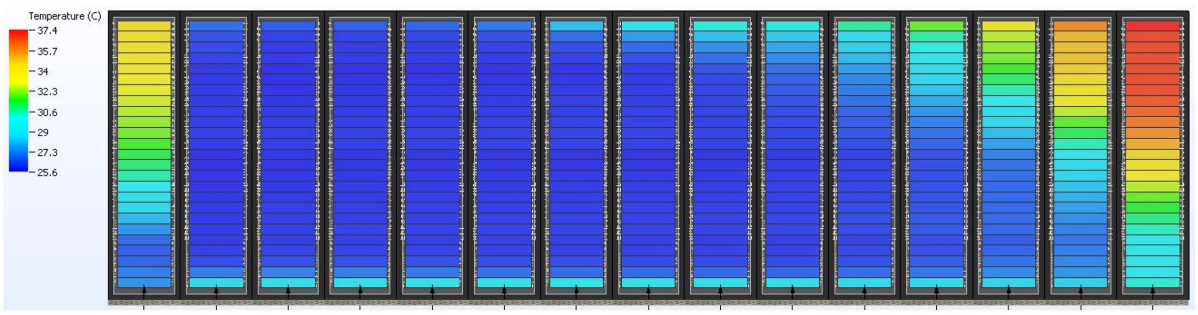
(b) Mean Server Inlet Temperature of row Z (Optimized)



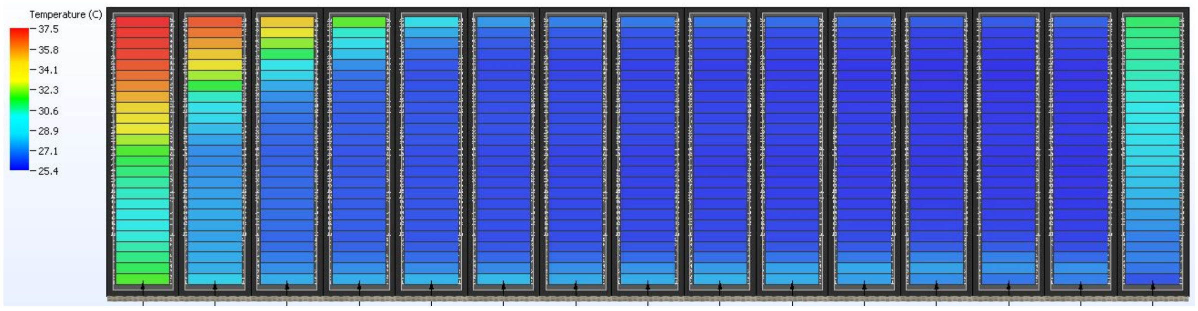
(c) Mean Server Inlet Temperature of row W (Non-Optimized)



(d) Mean Server Inlet Temperature of row Z (Non-Optimized)



(e) Mean Server Inlet Temperature of row W (Without Plates)



(f) Mean Server Inlet Temperature of row Z (Without Plates)

Figure J.2: The plot of mean server inlet temperature present in row W and row Z respectively and shown for the optimized case (a) and (b), non-optimized case (c) and (d), and without perforated plates (e) and (f). For the servers present in the optimized case, most of them are within the temperature of $27°C$ and very Few above $32°C$. In the non-optimized case, most of the server temperatures are above $32°C$ and in the case without the perforated plates, most of the server temperature exceeds $32°C$. Thus servers in case (a) and (b) have a better server inlet temperature.

References

[1] keras/optimizers.py at master · keras-team/keras · GitHub. URL https://github.com/keras-team/keras/blob/master/keras/optimizers.py#L467.

[2] ASHRAE TC9.9 Data Center Networking Equipment-Issues and Best Practices. Technical report, .

[3] Proposed Clarification to ASHRAE Thermal Guidelines SI Units Table 2011 Thermal Guidelines-SI Version Classes (a) Equipment Environmental Specifications for Air Cooling Product Operations. Technical report, .

[4] Air-Based Cooling vs. Liquid-Based Cooling | Green Revolution Cooling %. URL https://bit.ly/2JkgLTw.

[5] Neural Network Bias: Bias Neuron, Overfitting and Underfitting - MissingLink.ai. URL https://bit.ly/2QIIphq.

[6] GitHub - harvitronix/neural-network-genetic-algorithm: Evolving a neural network with a genetic algorithm. URL https://bit.ly/2y16yJl.

[7] Let's evolve a neural network with a genetic algorithmcode included. URL https://bit.ly/3bmvR6Y.

[8] Introduction to Genetic Algorithms Including Example Code. URL https://bit.ly/2vYoyU3.

[9] The Debate Between Hard Flooring and Raised Flooring in Data Centers. URL https://bit.ly/2V8LTv5.

[10] Latin Hypercube Sampling | RiskAMP. URL https://www.riskamp.com/latin-hypercube-sampling.

[11] [PDF] Alternating Cold and Hot Aisles Provides More Reliable Cooling for Server Farms | Semantic Scholar. URL https://bit.ly/2y4kwtX.

[12] Activations - Keras Documentation. URL https://keras.io/activations/.

[13] Gentle Introduction to the Adam Optimization Algorithm for Deep Learning. URL https://bit.ly/2xHOXWE.

[14] Understanding the Bias-Variance Tradeoff - Towards Data Science. URL https://bit.ly/3bszSag.

[15] Data Center Containment Solutions, . URL https://bit.ly/2JpauWY.

[16] Benefits of Data Center Containment | Data Center Knowledge, . URL https://bit.ly/2QT0R6K.

[17] Amsterdam district heating goal gets data centre boost | Euroheat & Power. URL https://bit.ly/33QTJxg.

[18] GitHub - DEAP/deap: Distributed Evolutionary Algorithms in Python. URL https://github.com/DEAP/deap.

[19] Energy-Efficient Cooling Control Systems for Data Centers | Department of Energy. URL https://bit.ly/3dEAFqy.

[20] Home | Future Facilities. URL https://www.futurefacilities.com/.

[21] David E. Goldberg - Wikipedia. URL https://en.wikipedia.org/wiki/David_E._Goldberg.

[22] David E. Goldberg - Wikipedia. URL https://en.wikipedia.org/wiki/David_E._Goldberg.

[23] Practical Guide to Hyperparameters Optimization for Deep Learning Models, . URL https://bit.ly/2JuiplL.

[24] GitHub - hyperopt/hyperopt: Distributed Asynchronous Hyperparameter Optimization in Python, . URL https://github.com/hyperopt/hyperopt.

[25] Home - Keras Documentation. URL https://keras.io/.

[26] keras/advanced_activations.py at master · keras-team/keras · GitHub, . URL https://github.com/keras-team/keras/blob/master/keras/layers/advanced_activations.py#L19.

[27] What are Hyperparameters ? and How to tune the Hyperparameters in a Deep Neural Network?, . URL https://bit.ly/3aDUnAw.

[28] NSGA-II explained! - analytics lab @ OU. URL https://bit.ly/39gby9X.

[29] Optimizers - Keras Documentation. URL https://keras.io/optimizers/.

[30] Multi-Objective Optimization using Evolutionary Algorithms - Kalyanmoy Deb - Google Books.

[31] pyDOE: The experimental design package for python pyDOE 0.3.6 documentation. URL https://bit.ly/2WN5R0D.

[32] A Gentle Introduction to the Rectified Linear Unit (ReLU), . URL https://bit.ly/2U7u2oH.

[33] The vanishing gradient problem and ReLUs - a TensorFlow investigation - Adventures in Machine Learning, . URL https://bit.ly/2UzVlbv.

[34] scikit-learn: machine learning in Python scikit-learn 0.22.1 documentation. URL https://scikit-learn.org/stable/.

[35] medium-and-small-data-centers. URL https://bit.ly/3ai1GxW.

[36] Transfer learning - Wikipedia. URL https://en.wikipedia.org/wiki/Transfer_learning.

[37] The Vanishing Gradient Problem - Towards Data Science. URL https://bit.ly/2JxQwZX.

[38] General guidelines for data centers. URL https://www.ibm.com/support/knowledgecenter/8335-GTW/p9ebe/p9ebe_generalguidelines.htm.

[39] Waleed A. Abdelmaksoud, H. Ezzat Khalifa, Thong Q. Dang, Roger R. Schmidt, and Madhusudan Iyengar. Improved CFD modeling of a small data center test cell. In 2010 12th IEEE Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems, ITherm 2010, 2010. ISBN 9781424453429. doi: 10.1109/ITHERM.2010.5501425.

[40] H. A. Alissa, K. Nemati, B. Sammakia, K. Ghose, M. Seymour, and R. Schmidt. Innovative approaches of experimentally guided CFD modeling for data centers. In Annual IEEE Semiconductor Thermal Measurement and Management Symposium, volume 2015-April, pages 176–184. Institute of Electrical and Electronics Engineers Inc., apr 2015. ISBN 9781479986002. doi: 10.1109/SEMI-THERM.2015.7100157.

[41] Anders Andrae and Tomas Edler. On Global Electricity Usage of Communication Technology: Trends to 2030. Challenges, 6(1):117–157, apr 2015. ISSN 2078-1547. doi: 10.3390/challe6010117. URL https://bit.ly/3doRNAt.

[42] Jayati Athavale, Yogendra Joshi, and Minami Yoda. Artificial Neural Network Based Prediction of Temperature and Flow Profile in Data Centers. Proceedings of the 17th InterSociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems, ITherm 2018, pages 871–880, 2018. doi: 10.1109/ITHERM.2018.8419607.

[43] P. L. Betts and I. H. Bokhari. Experiments on turbulent natural convection in an enclosed tall cavity. International Journal of Heat and Fluid Flow, 21(6):675–683, dec 2000. ISSN 0142727X. doi: 10.1016/S0142-727X(00)00033-3.

[44] T. T. Chow, G. Q. Zhang, Z. Lin, and C. L. Song. Global optimization of absorption chiller system by genetic algorithm and neural network. Energy and Buildings, 34(1):103–109, jan 2002. ISSN 03787788. doi: 10.1016/S0378-7788(01)00085-8.

[45] Adil A. Dafa'Alla and Philip L. Betts. Experimental study of turbulent natural convection in a tall air cavity. Experimental Heat Transfer, 9(2):165–194, 1996. ISSN 15210480. doi: 10.1080/08916159608946520.

[46] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T Meyarivan. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. Technical Report 2, 2002.

[47] Zhimin Du, Xinqiao Jin, and Yunyu Yang. Fault diagnosis for temperature, flow rate and pressure sensors in VAV systems using wavelet neural network. Applied Energy, 86(9):1624–1631, sep 2009. ISSN 03062619. doi: 10.1016/j.apenergy.2009.01.015.

[48] Laurene V. Fausett. Fundamentals of neural networks : architectures, algorithms, and applications. Prentice-Hall, 1994. ISBN 0130422509.

[49] Rajat Ghosh and Yogendra Joshi. Dynamic reduced order thermal modeling of data center air temperatures. In ASME 2011 Pacific Rim Technical Conference and Exhibition on Packaging and Integration of Electronic and Photonic Systems, InterPACK 2011, volume 2, pages 423–432. American Society of Mechanical Engineers Digital Collection, feb 2011. ISBN 9780791844618. doi: 10.1115/IPACK2011-52029.

[50] Rajat Ghosh and Yogendra Joshi. Rapid temperature predictions in data centers using multi-parameter proper orthogonal decomposition. Numerical Heat Transfer; Part A: Applications, 66(1):41–63, jul 2014. ISSN 10407782. doi: 10.1080/10407782.2013.869090.

[51] Ji Hoon Han, Dong Jin Choi, Sang Uk Park, and Sun Ki Hong. Hyperparameter Optimization Using a Genetic Algorithm Considering Verification Time in a Convolutional Neural Network. Journal of Electrical Engineering and Technology, 15(2):721–726, mar 2020. ISSN 20937423. doi: 10.1007/s42835-020-00343-7.

[52] I. E. Idelchik and A. S. (Aron Semenovich) Ginevskii. Handbook of hydraulic resistance. Begell House, 2007. ISBN 9781567002515.

[53] Madhusudan Iyengar, Hendrik Hamann, Roger R. Schmidt, and Jim Vangilder. Comparison between numerical and experimental temperature distributions in a small data center test cell. In 2007 Proceedings of the ASME InterPack Conference, IPACK 2007, volume 1, pages 819–826, 2007. ISBN 0791842770. doi: 10.1115/IPACK2007-33508.

[54] Anil K. Jain, Jianchang Mao, and K. M. Mohiuddin. Artificial neural networks: A tutorial, mar 1996. ISSN 00189162. URL https://bit.ly/2WHWNdn.

[55] Nicola Jones. How to stop data centres from gobbling up the world's electricity, sep 2018. ISSN 14764687.

[56] Yogendra Joshi. Reduced order thermal models of multiscale microsystems. Journal of Heat Transfer, 134(3), mar 2012. ISSN 00221481. doi: 10.1115/1.4005150.

[57] Yogendra Joshi. Reduced order thermal models of multiscale microsystems. Journal of Heat Transfer, 134(3), mar 2012. ISSN 00221481. doi: 10.1115/1.4005150.

[58] Nachiket Kansara, Rohit Katti, Kourosh Nemati, Alan P Bowling, and Bahgat Sammakia. IPACK2015-48684. pages 1–6, 2017.

[59] B. E. (Brian Edward) Launder and D. B. (Dudley Brian) Spalding. Lectures in mathematical models of turbulence. Academic Press, 1972. ISBN 9780124380509. URL https://catalogue.nla.gov.au/Record/1970799.

[60] Marjolein Hester Irene ten Haaft. Datacenters as residual heat source for district heating in residential neighbourhoods of Amsterdam. (May), 2019.

[61] Donald S. Miller and BHRA (Association). Internal flow systems. BHRA Fluid Engineering, 1978. ISBN 0900983787.

[62] Anthony F. Mills. Heat transfer. Irwin, 1992. ISBN 0256076421.

[63] Ahmed F. Mohamed, Mahdi M. Elarini, and Ahmed M. Othman. A new technique based on Artificial Bee Colony Algorithm for optimal sizing of stand-alone photovoltaic system. Journal of Advanced Research, 5(3):397–408, 2014. ISSN 20901232. doi: 10.1016/j.jare.2013.06.010.

[64] Suhas V. Patankar. Airflow and Cooling in a Data Center. Journal of Heat Transfer, 132 (7):1–17, 2010. ISSN 00221481. doi: 10.1115/1.4000703.

[65] Jeffrey D Rambo. REDUCED-ORDER MODELING OF MULTISCALE TURBULENT CONVECTION: APPLICATION TO DATA CENTER THERMAL MANAGEMENT A Dissertation Presented to The Academic Faculty. Technical report, mar 2006.

[66] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. Nature, 323(6088):533–536, 1986. ISSN 00280836. doi: 10.1038/323533a0.

[67] Emad Samadiani and Yogendra Joshi. Proper orthogonal decomposition for reduced order thermal modeling of air cooled data centers. Journal of Heat Transfer, 132(7):1–14, jul 2010. ISSN 00221481. doi: 10.1115/1.4000978.

[68] Viacheslav Semin, Mark Bardsley, Oliver Rosten, and Chris Aldham. Application of a multilevel unstructured staggered solver to thermal electronic simulations. Annual IEEE Semiconductor Thermal Measurement and Management Symposium, 2015-April:287–292, 2015. ISSN 10652221. doi: 10.1109/SEMI-THERM.2015.7100174.

[69] Deep V. Sheth and Sandip K. Saha. Numerical study of thermal management of data centre using porous medium approach. Journal of Building Engineering, 22(July 2018): 200–215, 2019. ISSN 23527102. doi: 10.1016/j.jobe.2018.12.012. URL https://doi.org/10.1016/j.jobe.2018.12.012.

[70] Saurabh K. Shrivastava, James W. VanGilder, and Bahgat G. Sammakia. Data center cooling prediction using artificial neural network. In 2007 Proceedings of the ASME InterPack Conference, IPACK 2007, volume 1, pages 765–771. American Society of Mechanical Engineers Digital Collection, jan 2007. ISBN 0791842770. doi: 10.1115/IPACK2007-33432.

[71] Saurabh K. Shrivastava, James W. VanGilder, and Bahgat G. Sammakia. Optimization of cluster cooling performance for data centers. In 2008 11th IEEE Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems, I-THERM, pages 1161–1166, 2008. ISBN 9781424417018. doi: 10.1109/ITHERM.2008.4544392.

[72] Zhihang Song, Bruce T. Murray, and Bahgat Sammakia. Airflow and temperature distribution optimization in data centers using artificial neural networks. International Journal of Heat and Mass Transfer, 64:80–90, 2013. ISSN 00179310. doi: 10.1016/j.ijheatmasstransfer.2013.04.017. URL http://dx.doi.org/10.1016/j.ijheatmasstransfer.2013.04.017.

[73] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 15:1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

[74] Michael Stein. Large sample properties of simulations using latin hypercube sampling. Technometrics, 29(2):143–151, 1987. ISSN 15372723. doi: 10.1080/00401706.1987.10488205.

[75] Bertil Thomas and Mohsen Soleimani-Mohseni. Artificial neural network models for indoor temperature prediction: Investigations in two buildings. Neural Computing and Applications, 16:81–89, 01 2007. doi: 10.1007/s00521-006-0047-9.

[76] Ali Vafamehr. Energy-aware cloud computing. The Electricity Journal, 31, 03 2018. doi: 10.1016/j.tej.2018.01.009.

[77] H K Versteeg and W Malalasekera. An Introduction to Computational Fluid Dynamics Second Edition. Technical report. URL www.pearsoned.co.uk/versteeg.

[78] Charles Villemure, Louis Gosselin, and Guy Gendron. Minimizing hot spot temperature of porous stackings in natural convection. International Journal of Heat and Mass Transfer, 51(15-16):4025–4037, jul 2008. ISSN 00179310. doi: 10.1016/j.ijheatmasstransfer.2007.11.052.

[79] Lizhe Wang, Gregor Von Laszewski, Fang Huang, Jai Dayal, Tom Frulani, and Geoffrey Fox. Task scheduling with ANN-based temperature prediction in a data center: A simulation-based study. Engineering with Computers, 27(4):381–391, oct 2011. ISSN 01770667. doi: 10.1007/s00366-011-0211-4.

[80] Emelie Wibron, Anna Lena Ljung, and T. Staffan Lundström. Computational fluid dynamics modeling and validating experiments of airflow in a data center. Energies, 11(3), 2018. ISSN 19961073. doi: 10.3390/en11030644.

[81] Xuanhang Zhang, Madhusudan Iyengar, James W. VanGilder, and Roger R. Schmidt. Effect of rack modeling detail on the numerical results of a data center test cell. In 2008 11th IEEE Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems, I-THERM, pages 1183–1190, 2008. ISBN 9781424417018. doi: 10.1109/ITHERM.2008.4544395.

[82] Liang Zhou and Fariborz Haghighat. Optimization of ventilation system design and operation in office environment, Part I: Methodology. Building and Environment, 44(4): 651–656, apr 2009. ISSN 03601323. doi: 10.1016/j.buildenv.2008.05.009.

[83] Yu Zhou, Yelin Deng, Peng Wu, and Shi Jie Cao. The effects of ventilation and floor heating systems on the dispersion and deposition of fine particles in an enclosed environment. Building and Environment, 125:192–205, nov 2017. ISSN 03601323. doi: 10.1016/j.buildenv.2017.08.049.