

The background of the cover is a photograph of a building facade, likely a residential or institutional structure, featuring large windows and solar panels. The image is heavily filtered with a warm, orange-red color and overlaid with a semi-transparent grid pattern, giving it a technical or architectural feel.

MSc thesis Building Technology

Towards Solar Irradiation Prediction on 3D Urban Geometry using Deep Neural Networks

Job de Vogel
2024

MSc thesis in Building Technology

Towards Solar Irradiation Prediction on 3D Urban Geometry using Deep Neural Networks

Job de Vogel

September 2024

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

Job de Vogel: *Towards Solar Irradiation Prediction on 3D Urban Geometry using Deep Neural Networks* (2024)

© ⓘ This work is licensed under a Creative Commons Attribution 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by/4.0/>.

The work in this thesis was carried out in:

Building Technology
Delft University of Technology

Supervisors:

Dr.ir. Michela Turrin

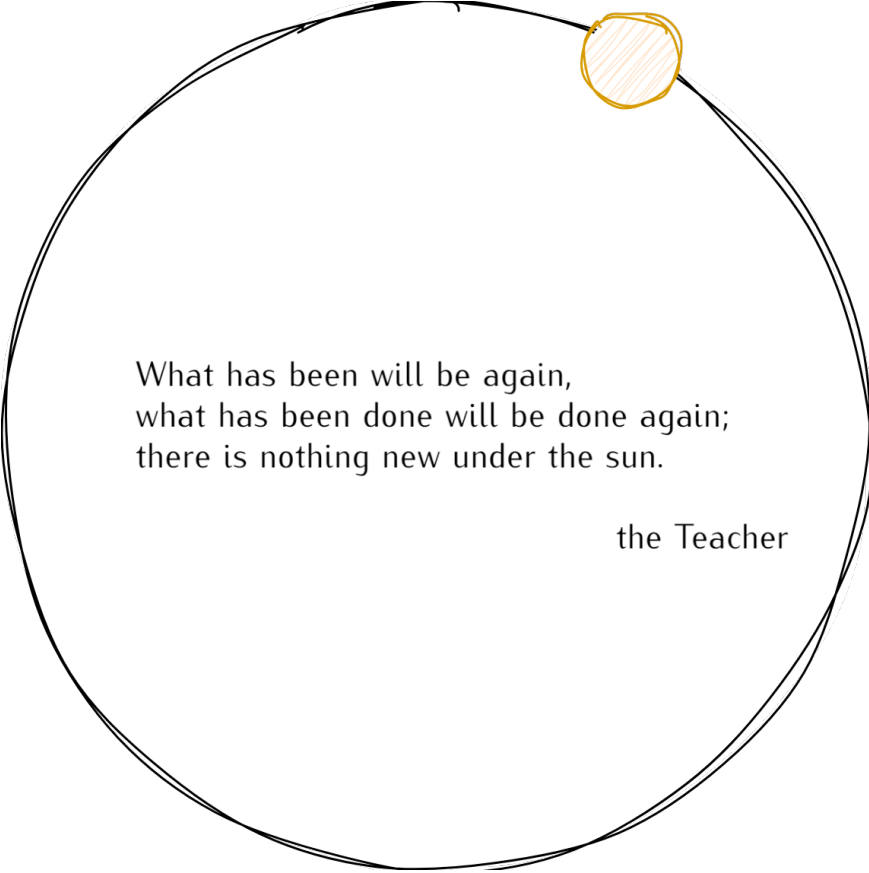
Dr.ir Seyran Khademi

External Advisor:

Dr.ir. Eleonora Brembilla

Representative Board of Examiners:

Prof.dr.ir. M.G. Elsinga



What has been will be again,
what has been done will be done again;
there is nothing new under the sun.

the Teacher

Abstract

This research is an early exploration into the potential of deep neural networks predicting physically-based building performance metrics on 3D urban geometry. Specifically, this work aims to predict annual solar irradiation using networks trained on point clouds. It is expected that the proposed method will allow designers to optimize their designs based on solar performance, due to the significantly lower inference time, in comparison to traditional simulation models. Furthermore, this research paves the way for generative models, which include the evaluation of performance such as solar irradiation, wind and acoustics.

Prior research has suggested several methods to predict solar irradiation on 2D building data and low resolution 3D buildings. These researches have in common that there is a lack of real observed irradiation data on buildings. Therefore, this research proposes several methods to efficiently synthesize an irradiation dataset based on 3D building geometry, which samples are larger in scale and resolution in comparison to earlier work.

Based on the synthesized datasets, several models have been trained to predict the irradiation values. Experiments have shown that a finetuned version of the model is able to predict irradiation with an average [RMSE](#) of 21 kWh/m² with an average inference time of 0.7 seconds, on a patch of 100x100 meters. Furthermore, this thesis provides an analysis on how patch size and point sampling technique affect the prediction error of the model.

Finally, this research provides an implementation of the network in a user-friendly client-server ecosystem, that can assist architects and engineers to fine-tune their building designs in urban environments.

Keywords: irradiation, synthesized dataset, deep neural networks, urban geometry

Acknowledgements

First and foremost, I would like to express my gratitude to my main mentors, Michela Turrin and Seyran Khademi, for their guidance throughout this research. Even after several extensions, they continued to provide the support I needed to complete this thesis. I am also sincerely appreciative of all the assistance Eleonora Brembilla offered, which greatly enhanced my understanding of daylight and simulations. Additionally, I would like to thank the representative from the Board of Examiners, Marja Elsinga, for her involvement.

This thesis would not have been possible without the many people who took the time to have a coffee with me, delve into my code, and share their expertise to assist me during this research. In particular, I would like to thank Shenglan Du, Berk Ekici, Nima Frouzandeh, Mark Bekooy, Peter Nelemans, and Lisa van Barneveld for their insights and suggestions. I am especially grateful to Aytaç Balci for his generous help, both during and outside working hours, in using and managing the BK Renderfarm.

Lastly, I am deeply thankful for all the support from my family, friends, and housemates throughout the highs and lows of this thesis. Without their encouragement, I could not have completed this work.

Contents

1. Introduction	1
1.1. Problem Statement	2
1.2. Research Question	2
1.3. Disciplinary Approach	3
1.4. Scope	3
1.5. Software and Hardware	4
1.6. Thesis Structure	4
2. Related Literature	7
2.1. Solar Irradiation Simulations	7
2.1.1. Raytracing	7
2.1.2. Daylight Simulations	8
2.1.3. Mathematical Background	10
2.1.4. 2-Phase Method and Neural Networks	12
2.1.5. Software and Simulation Techniques	13
2.1.6. Limitations of Daylight Simulations	14
2.2. Deep Neural Networks	15
2.2.1. Artificial Neural Networks	15
2.2.2. Convolutional Neural Networks	15
2.2.3. Generalization and Normalization	16
2.2.4. UNET	16
2.2.5. Generative Adversarial Networks	17
2.2.6. Diffusion Models	17
2.2.7. Transformers	18
2.2.8. Conclusion	18
2.3. Related Research Solar Irradiation Prediction	19
2.3.1. Related Research Limitations	22
2.4. Neural Network Architectures with 3D inputs	23
2.4.1. VoxNet	23
2.4.2. OCNN and OctNet	23
2.4.3. MeshCNN	25
2.4.4. Multiview CNN	26
2.4.5. PointNet	26
2.4.6. Model Discussion	27
2.4.7. Conclusion	27
3. Methods	29
3.1. Framework	30
3.2. Generation	30
3.2.1. Geometry source	31
3.2.2. Partitioning	32
3.2.3. Augmentation	32

Contents

3.2.4.	Point Sampling	32
3.2.5.	Final Format	33
3.3.	Simulation	34
3.3.1.	Parameter Convergence Test	35
3.3.2.	Final Format	39
3.4.	Parallelization	40
3.5.	Prediction	42
3.5.1.	PointNet	42
3.5.2.	PointNet++	43
3.5.3.	PointNeXt	44
3.5.4.	Model Sizes	48
3.5.5.	Training, Validation and Testing	48
3.5.6.	PointNext	50
3.6.	Interaction	53
3.6.1.	Preprocessing	53
3.6.2.	Live Prediction	53
3.6.3.	Visualization	55
3.6.4.	Optimization	55
3.6.5.	Conclusion	55
4.	Analysis	57
4.1.	Dataset Generation, Simulation, and Parallelization	57
4.1.1.	Dataset Sizes and Types	57
4.1.2.	Generation	58
4.1.3.	Simulation	61
4.1.4.	Parallelization	62
4.2.	Prediction	63
4.2.1.	Baseline Evaluation	63
4.2.2.	Hyperparameter Tuning	65
4.2.3.	Average Performance Improvements	73
4.2.4.	Visual Evaluation	74
4.2.5.	Imbalanced Dataset Correction	77
4.2.6.	Network Inference Optimization	81
4.2.7.	Experiment 1: Random Dataset	82
4.2.8.	Experiment 2: Sample Size	87
4.2.9.	Conclusion	90
4.3.	Interaction	91
4.3.1.	Optimization	97
4.3.2.	Overall Grasshopper Script	97
4.3.3.	Future Design Framework	99
5.	Discussion	101
5.1.	Research Questions	101
5.2.	Generation	102
5.2.1.	Regular Mesh Preprocessing Limitations	102
5.2.2.	Poisson Disk Sampling Optimization	103
5.2.3.	Augmentation	103
5.2.4.	Framework Limitations	103
5.2.5.	Further Research	104

5.3. Simulation	104
5.3.1. Accelerad vs Radiance	104
5.3.2. Direct vs Indirect Irradiation	104
5.3.3. Materials	105
5.3.4. Further Research	105
5.4. Parallelization	105
5.4.1. Further Research	105
5.5. Prediction	105
5.5.1. Practical Implications of "21 kWh/m ² RMSE"	106
5.5.2. Time and Location Invariance	106
5.5.3. Improving Mode Coverage	107
5.5.4. Geometric Level of Detail	107
5.5.5. Further Research	107
5.6. Interaction	108
5.6.1. External Hardware	108
5.6.2. CPU Inference	108
5.6.3. Implementation Other Design Software	108
5.6.4. Optimization	109
5.7. Hardware	109
5.8. Future Research Questions	109
6. Conclusion	111
7. Reflection	113
7.1. Academic Relevance	113
7.2. Societal Impact	113
7.3. Ethics	113
7.4. Personal Reflection	114
A. Regular Point Sampling Method	115
A.1. 3D BAG Mesh Format	115
A.2. Mesh Discretization	115
A.3. Sample Outline	116
A.4. Building Component Extraction	117
A.5. Ground Levelling	117
A.6. Mesh Face Quadrangulation	118
A.7. Roof Levelling	119
A.8. Facade Mesh Generation	119
A.9. Combining Mesh Elements	120
A.10. Regular Point Sampling	120
A.11. Dividing Wall Sensor Point Removal	120
B. Point Sampling Techniques	123
Bibliography	125

List of Figures

2.1. Raytracing principle	8
2.2. Typical Honeybee irradiation simulation	9
2.3. Skydome discretization	9
2.4. Skydome and daylight coefficients	12
2.5. Ladybug daylight simulation recipes	13
2.6. Convolutional Neural Network	16
2.7. Image generator architectures	17
2.8. ANN by Alamar et al. (2021)	19
2.9. cGAN by Huang et al. (2022)	20
2.10. Model by Han et al. (2022)	21
2.11. Model by Nakhaee and Paydar (2023)	21
2.12. VoxNet by Maturana and Scherer (2015)	24
2.13. Quadtree explained	24
2.14. OCNN by Wang et al. (2017)	25
3.1. Workflow methodology	29
3.2. Used software for methodology	30
3.3. 3D BAG LoD's	31
3.4. Tiling system 3D BAG	31
3.5. Dataset array format without irradiance	33
3.6. Accelerad simulator	34
3.7. Parameter Convergence Test	36
3.8. AcceleRad parameters visualization 1 (low settings)	37
3.9. AcceleRad parameters visualization 2 (medium settings)	37
3.10. AcceleRad parameters visualization 3 (high settings)	38
3.11. Radiance parameters visualization 4 (low settings)	38
3.12. Dataset array format with irradiation	39
3.13. Parallelization code for dataset generation and simulation.	40
3.14. Parallelization workflow	41
3.15. PointNet	42
3.16. PointNet++	43
3.17. PointNeXt	44
3.18. PointNeXt: subsampling	45
3.19. PointNeXt: grouping	45
3.20. PointNeXt: MLP and reduction	46
3.21. PointNeXt: InvResMLP	47
3.22. PointNeXt: interpolation	48
3.23. Dataset imbalance	51
3.24. Client-server interaction system	54
4.1. Dataset geometry samples (regular)	59
4.2. Dataset geometry errors (regular)	60

List of Figures

4.3. Dataset pointclouds (regular)	60
4.4. Irradiation distribution	61
4.5. Log timings simulation	62
4.6. Basemark training loss	63
4.7. Basemark validation loss	64
4.8. Basemark test loss	64
4.9. Basemark test loss domain	65
4.10. Basemark micro/macro avg. F1-scores	65
4.11. Validation loss for normalization	68
4.12. Validation loss for lower radius and higher nsample	69
4.13. Validation loss for global residual connections	69
4.14. Validation loss for lower strides	70
4.15. Validation loss for more epochs	71
4.16. Validation loss for more epochs	71
4.17. Validation loss for model scaling	72
4.18. Validation loss for other hyperparameters	73
4.19. Highest avg. errors regular samples	74
4.20. Median avg. errors regular samples	75
4.21. Lowest avg. errors regular samples	76
4.22. Validation loss for different loss functions	77
4.23. Accuracy over irradiation bins	78
4.24. RMSE frequency samples	79
4.25. RMSE frequency points	79
4.26. Optimized models' confusion matrix	79
4.27. Validation loss for adjusted Weighted Mean Squared Error (WMSE)	80
4.28. Macro/micro avg. F1-score for adjusted WMSE	80
4.29. Recall voor (adjusted Weighted)MSE	81
4.30. Visualization highest errors Poisson Disk dataset	83
4.31. Visualization median errors Poisson Disk dataset	84
4.32. Visualization lowest errors Poisson Disk dataset	85
4.33. Accuracy of bins in Poisson Disk dataset	86
4.34. Errors in Poisson Disk dataset samples	86
4.35. Errors in Poisson Disk dataset points	86
4.36. Analysis of Poisson Disk prediction performance	87
4.37. Visualization highest irradiation 300x300m dataset	88
4.38. Visualization highest irradiation 300x300m Poisson Disk dataset	89
4.39. Interaction environment irradiation prediction	91
4.40. Grasshopper 3D BAG downloader node	92
4.41. Preprocessing context and design seperately	92
4.42. Server waiting in the interaction environment	93
4.43. Server executing client call	94
4.44. Irradiation visualization on mesh	95
4.45. Vertical irradiation extraction	96
4.46. Horizontal irradiation extraction	96
4.47. Optimizing urban context for irradiation	97
4.48. Grasshopper script combined	98
4.49. Future design framework	99
4.50. Future design framework multiple domains	100
A.1. 3D BAG mesh	115

A.2. 3D BAG sensor grid	116
A.3. Sample outline	116
A.4. Extracted roofs and facades	117
A.5. Extracted building/courtyard outlines	117
A.6. Ground mesh	118
A.7. Triangle mesh faces on ground	118
A.8. Building roofs	119
A.9. Building walls	119
A.10. Merged non-manifold mesh	120
A.11. Vertical raytracing	121
A.12. Horizontal raytracing	121

List of Tables

- 2.1. Related research papers solar irradiation prediction. 22
- 3.1. Selected and benchmark parameters for Radiance and AcceleRad settings. 35
- 4.1. Dataset sizes 57
- 4.2. Efficiency of generation and simulation methods. 62
- 4.3. Hyperparameter tuning part A 66
- 4.4. Hyperparameter tuning part B 67
- 4.5. Performance improvements after tuning 73
- 4.6. Optimized inference timings 81

Acronyms

AHN	Algemeen Hoogtebestand Nederland	31
AI	Artificial Intelligence	1
ANN	Artificial Neural Network	15
aWMSE	Adjusted Weighted Mean Squared Error	80
BAG	Register of Buildings and Addresses	
cGAN	Conditional Generative Adversarial Network	17
CNN	Convolutional Neural Network	15
CPU	Central Processing Unit	
CUDA	Compute Unified Device Architecture	
FPS	Farthest Point Sampling	43
GAN	Generative Adversarial Network	17
GPT	Generative Pre-trained Transformer	
GPU	Graphics Processing Unit	
GIS	Geographic Information System	20
GSI	Ground Space Index	62
GUI	Graphical User Interface	108
IO	Input Output	
InvResMLP	Inverted Residual Multi-Layer Perceptron	44
LBT	Ladybug Tools	8
LiDAR	Light Detection and Ranging	
LoD	Level of Detail	31
LSTM	Long Short-Term Memory	18
MLP	Multi-Layer Perceptron	15
MSE	Mean Squared Error	19
MVCNN	Multiview Convolutional Neural Network	
OCNN	Octree Convolutional Neural Network	23
RAM	Random Access Memory	
ReLU	Rectified Linear Unit	15
RMSE	Root Mean Squared Error	35
RNN	Recurrent Neural Network	18
VAE	Variational Autoencoder	21
VRAM	Video Random Access Memory	
WMSE	Weighted Mean Squared Error	xvi

1. Introduction

The recent rise of Artificial Intelligence (AI) models has demonstrated that computers are capable of replacing complex human tasks. Large language models such as GPT (OpenAI et al., 2023) and image generators based on Stable Diffusion (Rombach et al., 2021), significantly influenced tasks such as writing, designing and the creation of art. Similarly to other fields, architecture in the built environment has also been influenced by the uprising of large AI models. Yet, no holistic model exists that is able to design buildings in the integral manner as an architect.

Over the past few years however, there has been growing interest in the development of AI models for architecture in the conceptual design phase (Castro Pena et al., 2021). These new technologies typically focus on the evaluation of certain specific objective and subjective subtasks within the design process (M.Matter & G.Gado, 2024). It is now possible to generate alternative visualizations of a building within seconds using AI (Ploennigs & Berger, 2023), predict pedestrian wind factors in urban environments (Mokhtar et al., 2021) and generate floorplans using neural networks (Wu et al., 2019). Given these developments, it is plausible that models will be combined in the future, to assist architects overcome entire phases of the design cycle.

Architecture is a profession mainly based on experience and creativity. For some design decisions, architects use computer-aided simulations to understand how a building will perform in the future. Examples are the prediction of building physics such as irradiation, wind, acoustics and energy consumption, human behavior or construction evaluations. Although these simulations can be accurate estimations of real building behavior, they are usually time consuming, and require expert knowledge to interpret the results.

Future AI models will only be able to help designers with complex decisions, if they are able to get a deeper understanding in the relationship between design and building performance outcomes. They need to be able to interpret how a certain design change will affect building physics, construction, financial costs and human experience. In contradiction to simulators, AI models are able to gain experience by learning from buildings that architects have developed in the past, together with their building performance. The intrinsic knowledge of AI models is based on significantly larger datasets than an architect or designer can have based on experience. It is therefore expected that AI models could aid architects in the future, by providing suggestions based on millions of buildings.

This thesis is an early exploration into the capability of AI models to predict building performance metrics by using simulated training examples. Specifically, the focus lies on the prediction of solar irradiation on 3D urban geometry using surrogate deep learning models. In this context, 'surrogate models' refer to statistical approximations of the relationship between data inputs and outputs. They are useful to mimic simulation models while being less time consuming. In the short term, this research can result in faster evaluation and optimization of buildings based on solar performance. In the long term, it is expected that the suggested methods can contribute to the development of more holistic generative building design models.

1.1. Problem Statement

Simulations of annual solar irradiation are computationally intensive. Using neural networks to predict solar irradiation, rather than simulating it, is expected to significantly reduce computation time. This reduction can assist architects in evaluating their designs more effectively and facilitates the use of traditional brute-force optimization approaches due to the decreased evaluation time.

Meanwhile, one might argue that advances in computer hardware will eventually mitigate the issue of slow simulations. However, it is important to recognize that predicting solar irradiation using surrogate AI models is fundamentally distinct from simulations, owing to the intrinsic understanding of the relationship between design and solar performance. Thus, this approach is expected to contribute significantly to the development of generative design models.

Additionally, the practical implementation of these models must be considered. It is anticipated that future users, such as architects, may lack a computer science background. Given that most AI models are implemented with advanced coding, it is essential to simplify the process of utilizing these algorithms.

1.2. Research Question

For this research, the following research question has been identified: 'How can one predict annual solar irradiation on 3D urban geometry using deep neural networks?'. Furthermore, the following sub questions will be discussed, to answer the main research question:

Literature research

- How do simulation models compute solar irradiation on 3D geometry?
- What type of input dataset is required to predict solar irradiation using deep neural networks?

Implementation

- How can one compute a dataset with 3D urban geometry with solar irradiation values efficiently?
- Can deep neural networks predict solar irradiation values accurately, faster than a simulation model?
- How can architects interact with the deep learning model in a user-friendly ecosystem?

1.3. Disciplinary Approach

This thesis consists of two types of research: literature review and project development through (visual) programming.

For the literature review on the theoretical background of solar irradiation simulations and neural networks, the information is mainly derived from publicly available research papers and academic books. In some specific cases, technical documentation from software was used to explain the inner workings of certain tools.

This research is an early exploration, but not the first to strive for the prediction of solar irradiation using AI. Therefore, an overview of earlier work is provided. The limitations of the tools developed in these studies are described based on the authors' personal experience and discussions with other academics in the field. It is important to note that not all results from earlier research can be verified, as not all code was available open-source. Consequently, the outcomes of these studies are verified using published conference papers, books, and the websites of the corresponding academic groups.

This research heavily relies on the development of neural network architectures by other researchers. Specifically, most of the work is based on the research by Qian et al. (2022). The modifications made to the architecture are explicitly described in this thesis. To facilitate the reuse of the code developed for this thesis, a GitHub repository is provided.

The author acknowledges that architects and designers typically do not use the coding languages employed for the implementation of the proposed model. Therefore, significant effort was invested in developing a workflow that is accessible and easy to use in practice.

1.4. Scope

Within the scope of this thesis, several challenges have been identified in solving the problem of solar irradiation prediction using deep neural networks. However, due to the complexity of the task and limited time, several simplifications have been implemented to make the research more feasible.

This research focuses exclusively on the prediction of annual solar irradiation. While there are many other intriguing solar performance metrics, such as glare, illuminance, or hourly irradiation, annual irradiation simulations are typically the most time-consuming (excluding cumulative hourly irradiation simulations). Moreover, most other research in the field of AI-based solar metric prediction also concentrates on this metric.

Another significant limitation is that the deep neural network will be trained on simulated data, not on real observed data. This is necessary due to the limited availability of observed data. In future research, it will be possible to fine-tune the proposed model with real observed data to overcome potential errors.

The purpose of this project is to predict solar irradiation, not to provide users with design advice, as generative models might do. Based on earlier developments in the AI field, it is expected that the proposed model can serve as a foundation for the further development of generative models related to solar performance. Finally, there are several simplifications concerning the dataset, particularly regarding the detail of the buildings and the generalization capacity of the model:

1. Introduction

- The AI model will not be location invariant. It is trained only on sun positions from the city of Amsterdam. The discussion will provide suggestions on how to make the model location invariant.
- The dataset consists of heavily simplified building geometry and leveled surrounding ground due to the complexity of accurately preprocessing the data. Within the scope of this thesis, one method is analyzed to address more complex geometry.
- No material types were included in the dataset, as this data was not available. Average values for absorption and reflection coefficients were selected. Suggestions on how to include materials in the prediction of irradiation will be provided in the discussion of this thesis.

1.5. Software and Hardware

The primary codebase for this project was developed in Python 3, chosen for its compatibility with CPython libraries. McNeel Rhino and Grasshopper were utilized for visualization and interaction purposes. To facilitate access to Python 3 AI models, a server-client system was established, with Python 3 functioning as the server and IronPython 2.7 as the client. Pytorch was selected as the main framework for developing the AI model.

For dataset development, a high-performance desktop equipped with two Intel Xeon E5-2640 v4 CPUs and two NVIDIA Quadro M6000 24GB GPUs was employed. Generating the dataset on a renderfarm or high-performance cloud computer was not feasible due to the dependency on Rhino.Inside, which is incompatible with the Linux operating system and requires pay-per-core-hour on the Windows Server operating system.

The AI model was trained using a renderfarm with a dual 28-core AMD EPYC 7453 CPU and two NVIDIA A40 48 GB GPU's on a Windows Server operating system. While most of the training was conducted on a single GPU, two GPUs were utilized for computational intensive hyperparameter settings.

The images included in this report were primarily created using draw.io and Matplotlib. This thesis was written using Overleaf \LaTeX .

1.6. Thesis Structure

The structure of this thesis is outlined as follows:

Chapter two, titled "Related Literature," addresses the literature research sub-questions posed in this thesis. Initially, it provides a mathematical overview of solar irradiation simulations, followed by a discussion of the software tools that implement these mathematical principles (2.1). The second section of chapter two delves into the fundamentals, history, and mechanisms of neural networks (2.2). Subsequently, it reviews previous research on the prediction of solar irradiation using neural networks (2.3). The fourth section explores the potential of using neural networks with 3D data, thereby addressing the second sub-question (2.4). Finally, an overview is provided on efficient computation methods for the generation of 3D urban irradiation datasets, focusing on various programming and software techniques (2.5). The chapter concludes with recommendations regarding the optimal models and methods to employ (2.6).

Chapter three, "Methods", details the implementation of this research through five key steps: the generation of building geometry (3.2), the simulation of irradiation (3.3), the parallelization of the aforementioned processes (3.4), the prediction and training using deep neural networks (3.5), and the interaction with the model from a user perspective (3.6).

Chapter four, "Analysis", presents an analysis of the implementation results. It begins by discussing the limitations encountered during dataset generation, simulation and parallelization (4.1). The subsequent section illustrates the results, accuracy, and performance of the neural network during training, evaluation, and testing phases (4.2). The final subsection of this chapter analyses the implementation of the model in an interaction framework (4.3).

Chapter five, "Discussion", offers a discussion that critically reviews the methodology and results. Chapter six, "Reflection", reflects on the research process, considering both the academic and societal impacts of this work.

2. Related Literature

2.1. Solar Irradiation Simulations

Solar studies on conceptual designs have become increasingly vital in the fields of architecture and engineering. By simulating sunlight on 3D building geometry, it is possible to address critical issues such as energy efficiency, visual comfort, and energy production through solar panels. Since the 1980s, advancements in computer processing power have enabled the accurate prediction of irradiation and illumination values on buildings (Tregenza & Waters, 1983). This chapter will explore the most significant advancements and methodologies in daylight simulations to better understand their potential and limitations. Building on this foundation, the following sections will examine the potential of AI models to replace or enhance these traditional methods.

2.1.1. Raytracing

The prediction of daylight values on 3D geometry has always been closely linked to the concept of ray tracing. Ray tracing refers to the process of calculating the path a light ray travels from its source to an object and ultimately to the camera or human eye. In the natural world, light travels from the light source to the object; however, in computer simulations, light is typically traced from the object back to the light source, a method known as "backward ray tracing". This approach is generally less computationally expensive than "forward ray tracing", as it allows for more efficient computation of reflections and refractions by tracing the light path in reverse (Arvo, 1986).

Daylight simulations employ two types of backward ray tracing: deterministic and stochastic. Deterministic algorithms produce consistent results each time they are run, regardless of the number of computations. In contrast, stochastic algorithms incorporate a random element, more closely mirroring the natural behavior of light, where photons travel in random directions. Both types of algorithms have their respective drawbacks in the context of ray tracing. Deterministic algorithms, while accurate, may fail to capture all the intricate details of light interactions. Stochastic algorithms, on the other hand, often introduce noise into the final render. Ideally, the strengths of both approaches are combined to achieve a balance between accuracy and computational efficiency, providing a fast yet precise estimation of light behavior (G. Ward & Shakespeare, 2011).

Another important concept in computer graphics is radiosity, also referred to as diffuse inter-reflection (Heckbert, 1993). Unlike ray tracing, which relies on the Monte Carlo principle and incorporates a statistical component, the radiosity algorithm simplifies calculations by assuming that scenes consist only of diffuse surfaces. This simplification makes radiosity more computationally efficient than Monte Carlo ray tracing. Additionally, radiosity offers the significant advantage of being view-independent, meaning the simulation is not tied to a specific viewpoint. In architectural applications, both Monte Carlo ray tracing and radiosity-based simulations are commonly employed (Tsangrassoulis & Bourdakos, 2003).

2. Related Literature

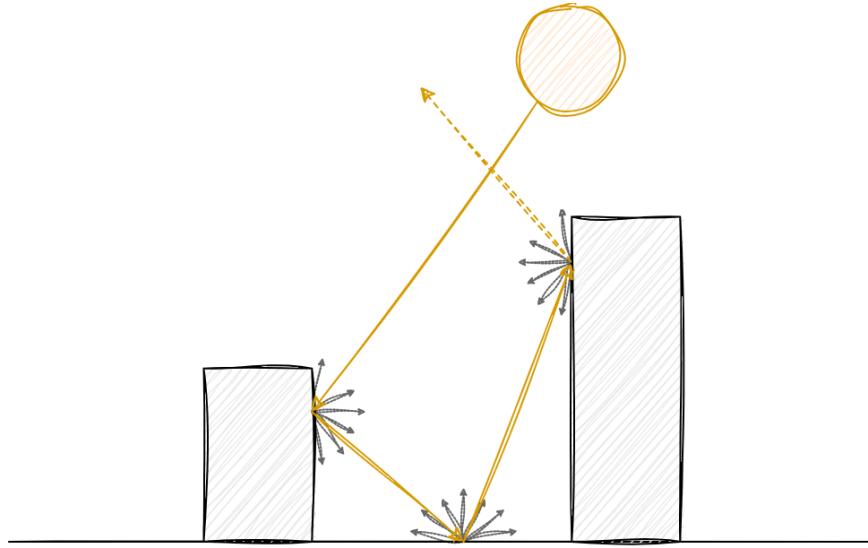


Figure 2.1.: Raytracing from the sun origin to an urban building block. (Image by author)

However, both Monte Carlo ray tracing and radiosity have inherent limitations, as neither can capture all possible lighting effects. To address these limitations, researchers have developed a technique known as photon mapping, which is based on a specific type of forward ray tracing. Photon mapping also benefits from being view-independent. While more advanced simulation software packages have integrated photon mapping techniques, these are not available in the simulation tools (Ladybug Tools ([LBT](#))) used in this research (G. J. Ward et al., [2022](#)).

2.1.2. Daylight Simulations

A daylight simulation involves predicting illuminance or irradiance values on 2D or 3D geometry (Figure: [2.2](#)). Among the various daylight metrics relevant to design, two are particularly significant for this research: illuminance, which is the luminous flux per unit area expressed in lux, and irradiance, which measures the radiant flux per unit area expressed in watts per square meter (W/m^2). Illuminance is the preferred metric when information about the quantity of light is needed, while irradiance is more appropriate when assessing the solar energy received by an object, such as in the context of solar panels. If irradiance is measured over a specified time period, the term irradiation is used, expressed in Watt-hours per square meter (Wh/m^2)¹. One Watt-hour indicates one Watt expended over one hour. When speaking about daylight coming from a source, given a direction, it is referred to as luminance, expressed in candela per square meter (cd/m^2), and radiance, expressed in watt per steradian per square meter ($\text{W}\cdot\text{sr}^{-1}\cdot\text{m}^{-2}$).

Daylight simulations are generally conducted over a specified time period. For instance, an annual direct sun study involves simulating light rays from the sun's position at each hour of the year. When these rays intersect with a model's geometry, it indicates that direct sunlight reaches the surface at that particular moment. While physical models were traditionally used for this purpose, contemporary architects typically create these models using computer software.

¹In this thesis, solar irradiation is usually expressed in kilowatt-hours per square meter (kWh/m^2)

2.1. Solar Irradiation Simulations

Daylight is composed of multiple components, each of which requires individual consideration. Primarily, daylight can be divided into regular sunlight and skylight, the latter being sunlight refracted within the atmosphere. Regular sunlight can be further classified into direct sunlight and indirect sunlight, the latter being light reflected between buildings. The impact of indirect sunlight can be significant, depending on the reflecting material. Similarly, skylight can be broken down into direct skylight and indirect skylight. The term "diffuse light" is often used in literature to refer to all components of daylight that are not direct sunlight. In contrast, when light is measured by a weather station, the terms "direct normal radiation" and "diffuse horizontal radiation" are typically used to describe these components (Mardaljevic et al., 2009).

As described, sunlight is not only reflected by physical objects but also by the sky. Even under cloudy conditions, objects still receive light, as indirect light from the sky is included in most daylight simulations. Daylight values can be mathematically represented by discretizing a skydome into multiple patches. This method, originally developed by Tregenza and Waters in 1983, standardizes the discretization into 145 patches, each corresponding to an average luminance or radiance value. Later advancements by Reinhart led to further refinement, dividing the skydome into 577 patches (Bourgeois et al., 2008) (figure: 2.3). This skydome approach enables the computation of illuminance and irradiance values for various models and sky conditions.

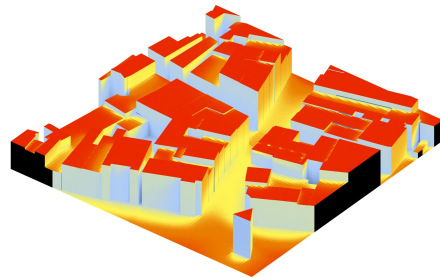


Figure 2.2.: An example of a typical daylight simulation using the Honeybee Cumulative Irradiance recipe. The colored heatmap indicates the irradiation received by the urban patch, distributed over a year. (Image by author)

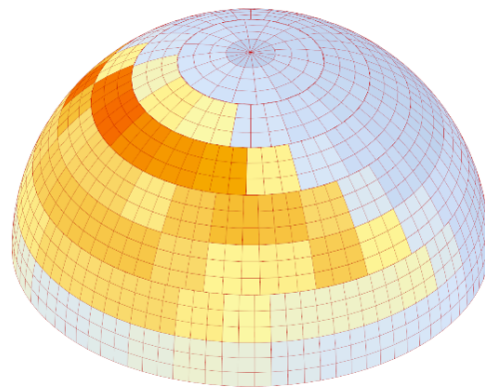


Figure 2.3.: A skydome discretized in 577 patches, as suggested by Reinhart (Image by author)

2. Related Literature

2.1.3. Mathematical Background

Daylight can be expressed as luminance or radiance using the following equation (G. Ward & Shakespeare, 2011):

$$L_r(\theta_r, \phi_r) = L_e + \iint L_i(\theta_i, \phi_i) f_r(\theta_i, \phi_i, \theta_r, \phi_r) |\cos \theta_i| \sin \theta_i d\theta_i d\phi_i \quad (2.1)$$

Note that the equation is recursive and is formulated as function in terms of itself. L_r describes the reflected light from a point in the direction (θ_r, ϕ_r) . L_e describes the emitted light from the point at which illuminance or irradiance are computed. L_i is the complex part of this equation, which describes the summation of incoming light from other direct and indirect directions. This value is multiplied by the reflectance-transmittance function f_r , which describes the ratio between reflected and transmitted light at this point.

Direct irradiance E_{dir} can be computed through specifically optimized algorithms such as selective shadow testing, adaptive source subdivision and virtual light calculation. Direct irradiance from a specific part of the sky can be described using:

$$E_{dir} = \int L_{dir}(\theta_i, \phi_i) \cos \theta_i \sin \theta_i d\omega_i \quad (2.2)$$

Where E_{dir} describes the incoming irradiance, L_{dir} the direct radiance from the direction (θ_r, ϕ_r) . ω_i describes the area of the given sky segment i which is visible from the origin. Thus, the total direct light from n direct directions (e.g. sun positions over time) in the sky can be computed using:

$$E_{dir;tot} = \sum_{i=0}^n E_{dir;i} \quad (2.3)$$

For the indirect component of incoming light, it is almost impossible to compute the quantity exactly. Overall, indirect irradiance can be described with (G. Ward & Shakespeare, 2011):

$$E_{ind} = \iint L_{ind}(\theta_i, \phi_i) \cos \theta_i \sin \theta_i d\theta_i d\phi_i \quad (2.4)$$

In which E_{ind} describes the incoming irradiance, L_{ind} the indirect radiance from a source from direction (θ_i, ϕ_i) on a projected hemisphere.

This equation is not computable due to the practically infinite number of directions light comes from. Therefore, it is approximated using the Monte Carlo inversion technique, with the following equation (G. Ward & Shakespeare, 2011):

$$E = \left(\frac{\pi}{M \cdot N} \right) \sum_{j=0}^{M-1} \sum_{k=0}^{N-1} L_{j,k} \quad (2.5)$$

E is the incoming irradiance, X_j and Y_k are uniformly distributed random variables in the range $[0, 1]$ and MN are the number of rays in which $N \approx \pi M$.

L_j, k is the indirect radiance in the direction $(\theta_j, \phi_k) = \left(\arcsin \sqrt{\frac{j+X_j}{M}}, 2\pi \frac{k+Y_k}{N} \right)$ on a projected hemisphere (G. Ward & Shakespeare, 2011).

2-Phase Method

When dealing with continuous light directions, such as those from the sky, a large number of potential light sources must be considered. To address this, the Daylight Coefficient method was developed (Tregenza & Waters, 1983). In academic literature, this method is sometimes interchangeably used with the "2-phase" method (Subramaniam, 2017).

As the name suggests, the 2-phase method computes illuminance or irradiation values in two distinct steps. The first step involves calculating the flux-transfer relationships between the segments of a skydome and a sensor point. In simple terms, the method does not directly predict specific irradiation values; rather, **it computes coefficients that describe the percentage of light from each sky patch that contributes to the incoming light at a sensor point**. Thus, if a skydome is divided into 145 patches, each sensor point will have 145 corresponding daylight coefficients.

The second step of the 2-phase method involves the creation of a sky matrix, which represents a skydome with sunlight values for one or more hours of the year. When assessing the cumulative amount of sunlight that a sensor point receives over a year, the values from all sky patches are summed.

Finally, the daylight coefficient matrix is multiplied by the sky matrix to compute the illumination or irradiation values for a given sensor point. Mathematically, the 2-phase method can be described by the following equations (Subramaniam, 2017).

$$\Delta E_{\theta\phi} = D_{\theta\phi} \cdot L_{\theta\phi} \cdot \Delta S_{\theta\phi} \quad (2.6)$$

Where $E_{\theta\phi}$ is the illuminance at a sensor point, $D_{\theta\phi}$ is the Daylight Coefficient that depends on the reflectance and transmittance of surrounding surfaces, $L_{\theta\phi}$ is the luminance of a sky patch, and $S_{\theta\phi}$ is the angular size of the sky patch, considering the altitude θ and azimuth ϕ . It should be noted, however, that illumination can be replaced by irradiation if $L_{\theta\phi}$ is replaced by radiance.

When converted to matrices, the equation can also describe the results for a grid of sensor points (Subramaniam, 2017):

$$E = C_{dc} \cdot S \quad (2.7)$$

Where C_{dc} describes a matrix of Daylight coefficients and S describes the sky vector. When luminance or radiance values are used for each hour of the year, sky vector S can be replaced by a sky matrix in which each row contains the luminance/radiance values for all patches at a certain hour.

The following example, derived from Subramaniam (2017), illustrates how the 2-phase method works. Assume an urban design with 1000 sensor points and a discretized sky with 145 patches. The dimensions of the Daylight Coefficient matrix C_{dc} would be $[1000 \times 145]$ and sky matrix S would be $[145 \times 8760]$ for 8760 hours of the year. The final result E with the irradiation values for all grid points for all hours of the year would be $[1000 \times 8760]$. To make the simulation run faster, all radiance values per sky patch can be summed or averaged, as described by (Robinson

2. Related Literature

& Stone, 2004). In that case, the resulting matrix E would only be of size $[1000 \times 1]^2$. Figure 2.4 shows a visual representation of this example.

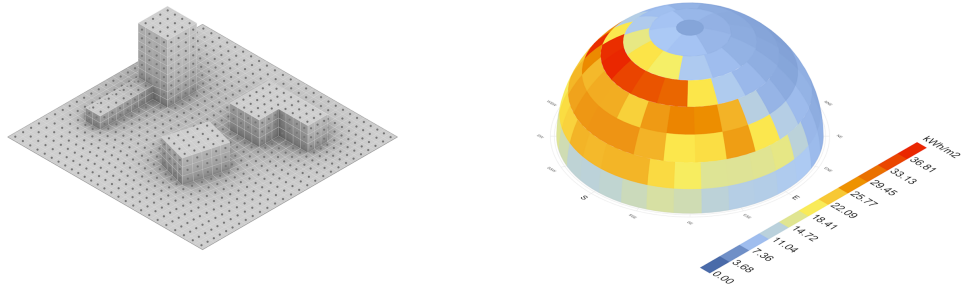


Figure 2.4.: On the left: urban geometry divided in 1000 sensor points. A sensor point has a daylight coefficient in relation to each sky patch. On the right: a skydome with 145 patches visualizing the sky matrix, generated with a Radiance function called gendaymtx. Each sky patch has a color indicating the radiance over a year. (Image by author)

In summary, daylight simulators use deterministic and stochastic raytracing equations to compute daylight coefficients which describe how much daylight is received from different sky patches. When multiplied with the skymatrix, the approximate irradiation or illumination values can be computed.

2.1.4. 2-Phase Method and Neural Networks

A thorough understanding of the equations for the 2-phase method is deemed essential for this thesis. When a neural network is used to replace a simulation, either one or both of the terms C_{dc} and S have to be fitted by the model, to be able to compute accurate results. When the location of a set of geometric training samples is not changed, only C_{dc} would be changing based on the setup of the design. On the other hand, if the geometry would be the same, but the location would be changing, only S would influence the results. Using neural networks to predict irradiation is a valid machine learning approach, due to the non-linear relation of the underlying equations (2.1) in the 2-phase method.

²Advanced versions of the 2-phase method sometimes make a distinction between the computation of direct and diffuse sunlight.

2.1.5. Software and Simulation Techniques

This section provides an overview of simulation tools available for estimating daylight on urban building blocks.

Ladybug Tools and Honeybee

Ladybug Tools is a plugin designed to simulate solar radiation, wind, and thermal performance in conceptual design. It is among the most widely utilized software packages for climate simulation in architecture. Within the Ladybug Tools suite, several methods (recipes) are available for simulating solar irradiation with varying levels of accuracy (Figure 2.5).

Firstly, a basic simulation of direct sun hours can be approximated using the Ladybug Direct Sun Hours component. This method casts rays from hourly sun positions to a grid of sensor points, calculating the total number of hours each geometry patch receives direct sunlight, without considering intensity or climate characteristics. It does not account for the diffuse components of sunlight (Ladybug Tools, 2024b).

For larger geometric models requiring more accurate results, the Annual Irradiance recipe from Honeybee is recommended. This model employs the 2-phase method to predict both direct and indirect irradiation on the proposed design. Additionally, it utilizes the Radiance engine, enabling the computation of ambient light bounces between buildings and the ground. Although this simulation offers high accuracy, it is also more time-consuming (Ladybug Tools, 2024a).

In situations where speed is critical for obtaining design feedback, alternative methods that compute both direct and indirect sunlight cumulatively over a year have been developed. These methods, as described by Robinson and Stone (2004), replace the traditional sky matrix with a summation of expected values from each sky patch. Ladybug Tools provides solutions for both scenarios—with (Honeybee) and without (Ladybug) Radiance—allowing for the computation of ambient light bounces as well (Ladybug Tools, 2024a).

Both Ladybug and Honeybee can be utilized through design software such as McNeel Rhino Grasshopper, as well as via the Python API.

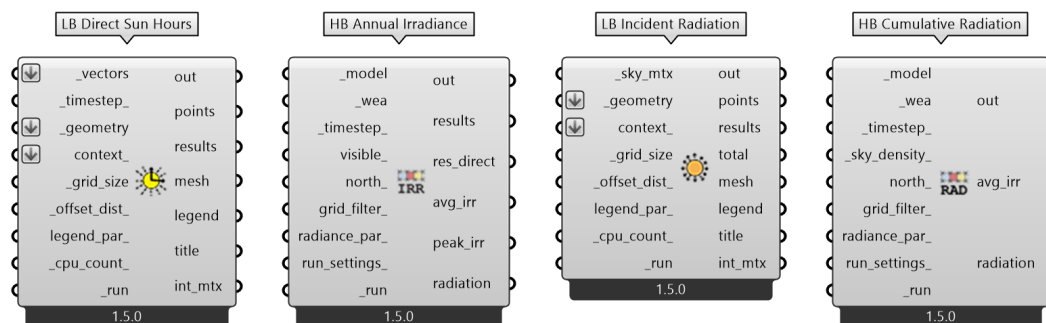


Figure 2.5.: Four available daylight simulation recipes in Ladybug and Honeybee. On the left: two recipes based on hourly simulations, of which the first indicates sun hours and the second simulation irradiance from sun and skylight. On the right: two recipes computing cumulative yearly results. Honeybee recipes are able to include ambient light bounces between buildings. (Image by author)

2. Related Literature

Radiance

Radiance is a comprehensive software package designed for daylight analysis and visualization. Developed by Greg Ward at the Lawrence Berkeley National Laboratory, Radiance provides a range of tools for simulating and evaluating daylight in architectural contexts. Although there is no singular approach to using Radiance, its tools are primarily accessed through a command line interface. To facilitate usage for non-programmers, various software packages, such as Honeybee, have developed wrappers around these commands (Subramaniam, 2017).

Radiance has been validated as an accurate simulation tool for predicting daylight values, as demonstrated by several studies by Brembilla and Mardaljevic (2019), Kharvari (2020), and Reinhart and Walkenhorst (2001). Consequently, Radiance will be employed in this thesis to generate the required dataset.

AcceleRad

AcceleRad, developed by Jones and Reinhart (2017), is a tool designed to optimize specific functions within Radiance for GPU acceleration. According to the documentation, AcceleRad can enhance processing speeds by a factor of up to 6. It integrates with Radiance by replacing particular system files, thereby improving computational efficiency (Jones & Reinhart, 2017).

Jones and Reinhart (2022) analyzed the error differences between Accelerad simulations, Radiance, and observed daylight values. They found that the largest errors occurred in illuminance-based simulations, which are similar to the simulation of irradiation on building surfaces (although irradiation simulations were not analyzed by the original authors). Therefore, it is crucial to examine these differences before determining whether it is appropriate to use Accelerad instead of Radiance.

2.1.6. Limitations of Daylight Simulations

Although daylight simulations are valuable tools for optimizing solar performance, they have notable limitations. This thesis focuses on the limitations specific to Radiance, as this research field predominantly emphasizes the validation of Radiance methods.

Firstly, many Radiance daylight simulations rely on sky descriptions based on the Perez weather model. However, actual sky conditions may deviate from the model's estimations, leading to discrepancies between simulated and real illuminance and irradiation values (Geisler-Moroder et al., 2017).

Secondly, variations in simulation results can arise due to inherent randomness in the method used to compute the indirect components of daylight. These differences are particularly noticeable in lower Radiance settings (Kharvari, 2020).

Lastly, the parameter settings in Radiance significantly influence the output results. Variations in parameter configurations across different studies can result in outcomes that are not always directly comparable (Kharvari, 2020).

2.2. Deep Neural Networks

Since the late 1980s, neural networks have gained prominence as powerful tools for addressing complex problems. Neural networks, a subset of machine learning in computer science, involve the development and use of models that simulate natural learning processes. They are inspired by the human brain's architecture, utilizing artificial neurons and connections to learn features from a dataset (Anderson & Mcneill, 1992).

A specific subfield of machine learning, computer vision, focuses on learning from image-based data. This includes tasks such as classification, segmentation, and detection of objects within images (Stevens et al., 2020).

2.2.1. Artificial Neural Networks

Before exploring Convolutional Neural Networks, it is essential to understand the fundamentals of Artificial Neural Networks (ANNs), also known as Multi-Layer Perceptrons (MLPs). ANNs are a type of supervised machine learning model inspired by the human brain. An ANN typically consists of three types of layers: the input layer, hidden layers, and the output layer. The input layer provides the data to the model, while the hidden layers contain neurons with associated weights and biases. These hidden layers are interconnected and usually decrease in size as they progress. The output layer produces the final predictions.

The difference between the predicted and actual values is computed using a loss function, which varies depending on the problem being solved. The loss is then backpropagated through the network and optimized using the Gradient Descent algorithm. ANNs are distinguished by their ability to learn features in non-linear problems, facilitated by non-linear activation functions applied to the weights, inputs, and biases of each neuron (Stevens et al., 2020).

2.2.2. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) extend the principles of ANNs, primarily for pattern recognition tasks. Initially designed for image processing, CNNs are now applicable to various types of data. A key advantage of CNNs is their ability to handle high-dimensional inputs more effectively than traditional ANNs.

A typical CNN architecture (figure 2.6) comprises three main components: convolutions, pooling, and fully connected (dense) layers. In image recognition, a convolution involves applying small 2D matrices of weights, known as filters or kernels, to regions of the input image to detect specific patterns such as lines or shapes. These convolutions are followed by an activation function, often the Rectified Linear Unit (ReLU). Filters are characterized by hyperparameters including kernel size, stride, zero-padding, and dilation. One significant benefit of CNNs is the reuse of filters across the entire image, which reduces the number of parameters that need to be learned.

Following the convolutional layers, pooling layers are used to reduce the dimensionality of the data. For example, max-pooling divides the input into smaller segments and retains the maximum value from each segment, effectively decreasing the image's size. For instance, an image with dimensions of 784x784 pixels can be reduced to 392x392 pixels using a pooling operation with a stride of 2.

2. Related Literature

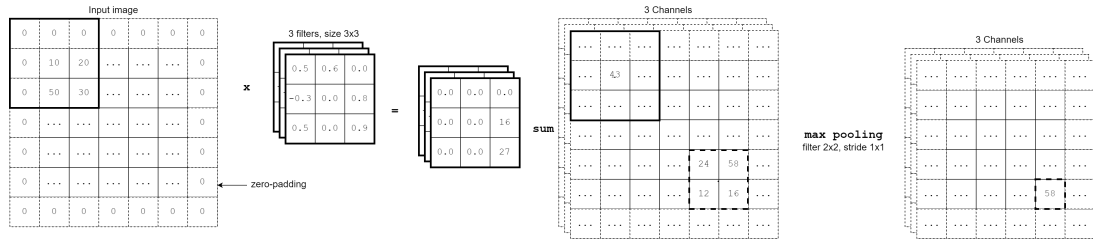


Figure 2.6.: Principle of the Convolutional Neural Network architecture (Image by author)

After several convolution and pooling operations, the resulting data is flattened into a one-dimensional array and fed into the fully connected layer. This layer functions similarly to the hidden layers in an ANN, performing the final classification or regression tasks (Stevens et al., 2020).

2.2.3. Generalization and Normalization

The primary goals in developing well-trained neural networks are to achieve generalization and reduce training time. Generalization refers to the model's ability to make accurate predictions on new, unseen data, avoiding both underfitting and overfitting. To facilitate this, various adjustments have been made to CNN architectures. Two significant developments are dropout and batch normalization.

Batch normalization normalizes the activation vectors within hidden layers. This process accelerates training by reducing internal covariate shifts and stabilizes learning, thus leading to faster convergence (Ioffe & Szegedy, 2015).

Dropout is a regularization technique that involves randomly deactivating a subset of neurons during each optimization step. This helps prevent overfitting by ensuring that the model does not become overly reliant on specific neurons (Garbin et al., 2020).

2.2.4. UNET

Introduced in 2015, the UNET architecture represents a significant advancement in CNN design, particularly for biomedical image segmentation. The UNET features an encoder-decoder structure that allows for precise segmentation of images. The encoder follows the conventional CNN approach with convolutions and pooling layers, while the decoder includes deconvolution and up-pooling layers to restore the input dimensions to their original resolution. This architecture enables UNET to generate high-resolution outputs suitable for regression or classification tasks (Ronneberger et al., 2015).

2.2.5. Generative Adversarial Networks

Generative Adversarial Networks (GANs) are designed for generating new data samples that resemble examples from a given dataset. GANs are distinguished by their generator-discriminator framework. The generator creates new data samples from random noise, while the discriminator evaluates whether these samples are real (from the dataset) or fake (generated). The discriminator's feedback is used to refine the generator, leading to progressively more realistic samples (Goodfellow et al., 2014).

Conditional Generative Adversarial Networks (cGANs) extend this approach by incorporating class labels into the generator and discriminator. For instance, given a black-and-white image, a cGAN can generate a corresponding color image, with the discriminator assessing the authenticity of the colored image. cGANs are considered semi-supervised due to their use of labeled data.

2.2.6. Diffusion Models

Diffusion models are a class of generative models that produce images by introducing noise to the input data and then reversing this process to generate new samples. This process involves gradually adding noise to the data and learning to denoise it step-by-step. The iterative nature of diffusion models enables them to capture a broader range of the data distribution, often resulting in more diverse and representative samples compared to GANs. Despite this advantage in diversity and mode coverage, GANs generally excel in producing highly detailed and visually coherent images (figure: 2.7).

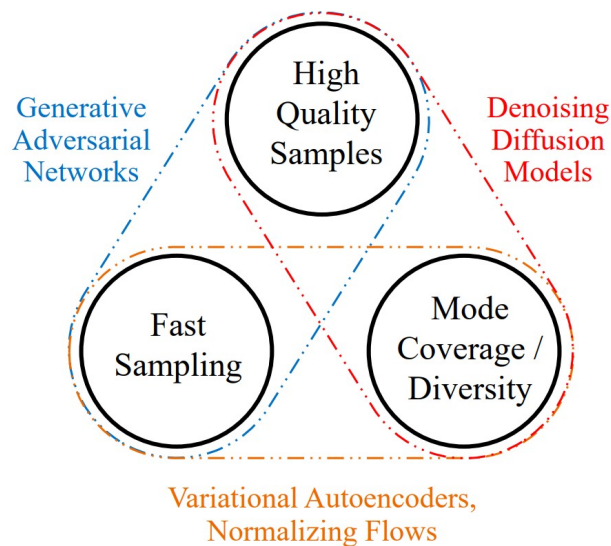


Figure 2.7.: Qualities of distinct image generator architectures (Vahdat & Kreis, 2022)

2. Related Literature

2.2.7. Transformers

Transformers utilize an attention mechanism to process input data, allowing the model to focus on different parts of the input based on their relevance rather than their positional distance. This mechanism effectively overcomes some of the limitations associated with Recurrent Neural Networks ([RNNs](#)) and Long Short-Term Memory ([LSTM](#)) networks, such as difficulty in handling long-range dependencies. Originally developed for natural language processing tasks, transformers have recently been adapted for 3D data applications. Notable examples include the analysis of point clouds (Zhao et al., [2021](#)) and the semantic segmentation of 3D meshes (Vecchio et al., [2023](#)).

2.2.8. Conclusion

This literature review describes how different network architectures can be combined for the purpose of generating images. Based on the previous descriptions, the reader should be able to understand the principles behind the networks for both 2D and 3D data, that will be described in the following chapters.

2.3. Related Research Solar Irradiation Prediction

In 2021, Alammam et al. introduced an ANN designed to assess solar irradiation on the facade of a specific building, utilizing a parametrically developed envelope of surrounding structures (see Figure 2.8). This ANN was evaluated against a Random Forest algorithm, a supervised machine learning technique that does not involve deep learning. The comparison revealed that the Random Forest model exhibited superior performance, primarily due to its use of categorical inputs. This outcome prompts a critical evaluation of the ANNs practical applicability, given its limitations in generalizability.

Similarly, Lila et al. (2021) proposed a more sophisticated approach for predicting solar irradiation. This method featured a model with inputs that were significantly more complex and irregular. Although the input to this ANN remained categorical, it was of higher dimensionality compared to previous models. The findings indicate that this advanced model is capable of predicting solar irradiation on building rooftops with reasonable accuracy.

Input	Input Neuron Type	Data Range
Hour	Discrete	6 to 18 in steps of 1
Month	Discrete	0,1,2,3
B00	Discrete	0,1,2
B01	Discrete	0,1,2
B02	Discrete	0,1,2
B03	Discrete	0,1,2
Facade Floor Level	Discrete	0,1,2
Orientation	Discrete	0,1,2,3
Façade Height	Discrete	6 to 66
x-coordinate	Continuous	[-11.08, 12.28]
y-coordinate	Continuous	[-13.06, 10.26]
z-coordinate	Continuous	[6.40, 69.60]

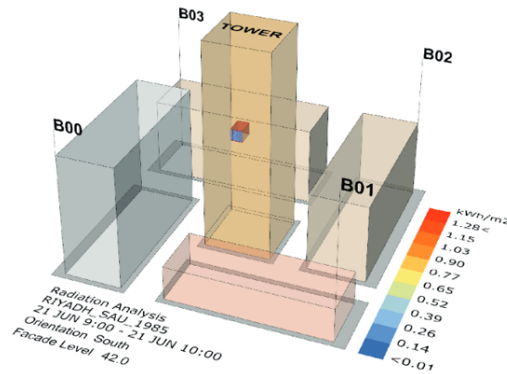


Figure 2.8.: Model developed by Alammam et al., 2021. On the left: categorical inputs used for training the Artificial Neural Network. On the right: an example of a facade patch, for which the irradiation is predicted. (Images by Alammam et al., 2021)

Tehrani et al. (2024) developed a model to predict average annual solar irradiation based on general urban properties such as coordinates, average building height, and azimuth angle. The dataset for this research was created using a Grasshopper parametric script and OpenStreetMap data (OpenStreetMap contributors, 2017). The geometric attributes were transformed into a one-dimensional array of properties related to average solar radiation values. The model employed was a manually crafted ANN, evaluated using the Mean Squared Error (MSE) metric. Although this approach demonstrates the use of location-variant datasets for solar irradiation prediction, it is limited in that it only predicts a single average value for a given urban topology.

Yue et al. (2024) developed similar work but added additional input features such as building type, building volume, and orientation of surfaces (roof, south, north, west, and east). Instead of predicting a single average irradiation value, their model distinguished between different surface orientations and accounted for obstructions from the local context. The performance of 17 common models was compared using a dataset of geometric data from the city of Zhengzhou, China.

2. Related Literature

Galanos et al. (2024) presented their approach for solar irradiation prediction on a larger scale in As and Basu (2021), using open-source GIS data for the city of Vienna. They performed solar radiation simulations using Ladybug tools and generated height maps from the imported 3D models. A GAN was then utilized to predict solar irradiation on these height maps. The final output showed irradiation values on the 2D ground between buildings. Similarly, Huang et al. (2022) used a GAN (see Figure 2.9) not only to predict solar irradiance on parametrically generated models but also to approximate wind levels and thermal comfort. This was achieved through a multi-objective genetic algorithm, requiring advanced GAN variants such as Pix2pix (Isola et al., 2016) and CycleGAN (Zhu et al., 2017). It is important to note that both studies focus on 2D surfaces around buildings, excluding facades and roofs from their predictions, which limits the feedback provided to designers.

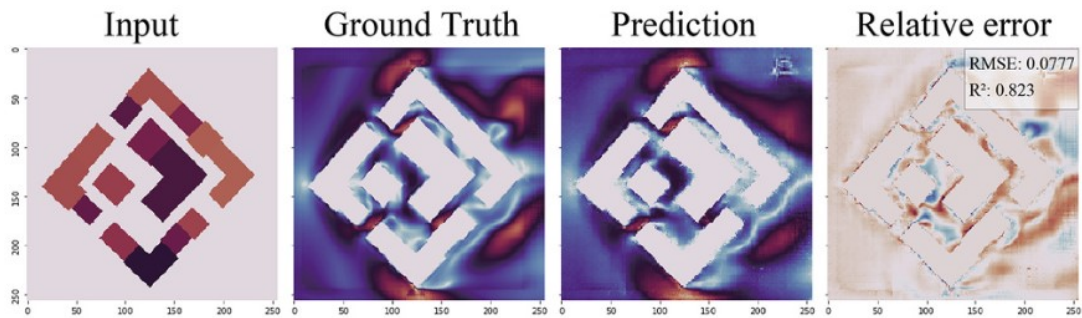


Figure 2.9.: Example result from the cGAN developed by Huang et al., 2022. (Images by Huang et al., 2022)

In contrast to the previously discussed studies, Han et al. (2022) introduced a novel approach for predicting solar irradiation on 3D geometry using advanced 3D CNNs (see Figure 2.10). Their method employs a synthetic database created through a parametric script that generates buildings with simple geometric shapes like cubes and cylinders on a flat envelope. This network, named Coolvox, is built on a UNET-like architecture, incorporating both deconvolutional and convolutional layers. Notably, Coolvox is capable of handling more complex building features, such as balconies, making it a more comprehensive tool for assessing solar irradiance on both the building envelope and facades.

While the 3D approach of Han et al. (2022) offers significant advantages for design applications—since it includes detailed facade geometry in addition to the surrounding envelope—it also presents substantial limitations. The primary challenge is the scalability of Coolvox. The use of 3D convolutional layers makes the model highly memory-intensive, which can be problematic during the training phase. As a solution, the authors suggest exploring Octree-based or Transformer-based networks in future research to alleviate memory constraints and enhance scalability.

Recent advancements in irradiation prediction research are focusing on addressing memory consumption issues associated with 3D models by utilizing fish-eye camera perspectives for training data.

Nakhaee and Paydar (2023) introduced a GAN-based network called DeepRadiation, built upon the Pix2pix (Isola et al., 2016) architecture (see Figure 2.11). Their approach begins with a dataset of New York buildings derived from Geographic Information System (GIS) data, which is used to simulate solar irradiation. This simulated irradiation data is then projected into a fish-eye

2.3. Related Research Solar Irradiation Prediction

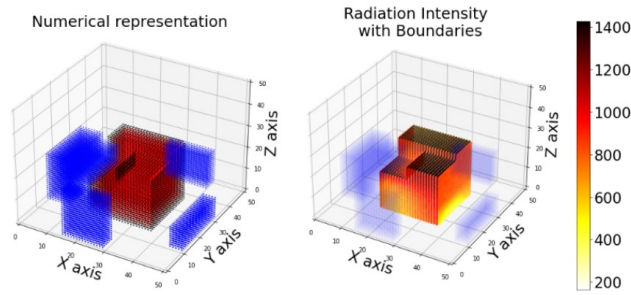


Figure 2.10.: Prediction of solar irradiance by Han et al., 2022. On the left: numerical representation of occupancy grid surrogate buildings. On the right: resulting irradiance values on building by 3DCNN model. (Images by Han et al., 2022)

perspective from eye height. The original fish-eye images, devoid of irradiation information, are semantically segmented and combined with a depth map. These processed images are used to train the GAN, enabling it to generate new images annotated with irradiation values. A notable limitation of this model is its restriction to predicting irradiation from an eye-level perspective, which may not always be practical for all design scenarios.

Meanwhile, Zhang et al. (2023) developed SolarGAN, an advanced GAN-based model that improves upon several limitations observed in previous models, including DeepRadiation. Unlike DeepRadiation, SolarGAN employs a Variational Autoencoder (VAE) architecture instead of a Pix2pix (Isola et al., 2016) architecture. This VAE-based approach allows the network to incorporate additional parameters into the latent space, facilitating training across diverse locations and making the model more versatile for designers. Instead of generating images with irradiation values, SolarGAN predicts an ensemble of annual hourly solar irradiation time series for building facades. Additionally, the model includes semantic segmentation of facades into ground, opaque surfaces, glazing, and sky, providing valuable insights for design decisions. The researchers also suggest that SolarGAN has the potential to be applied to real-world fish-eye images of streets, further enhancing its practical utility.

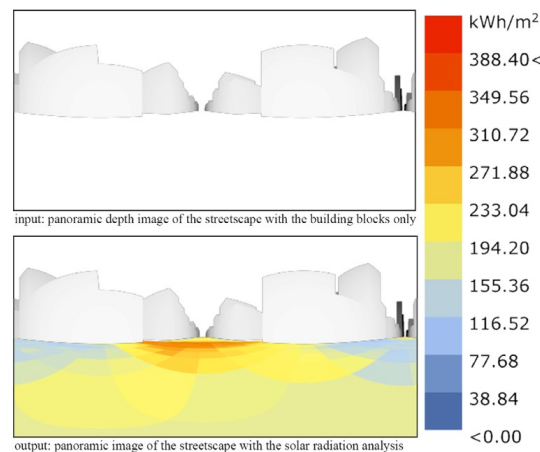


Figure 2.11.: Irradiation prediction on the ground using a fish-eye perspective. (Images by Nakhaee and Paydar, 2023)

2. Related Literature

2.3.1. Related Research Limitations

In summary, extensive research is ongoing to address the challenge of predicting solar irradiation using AI (Table: 2.1). However, as noted, each proposed solution presents its own set of limitations and challenges. Based on this literature review, the following key issues have been identified:

- To the best of the author's knowledge, no comprehensive model currently exists that can predict solar irradiation with results comparable to those of traditional solar irradiation simulations. Specifically, no model is capable of providing irradiation values across a 3D urban patch, including ground, facades, and roofs. The Coolvox model proposed by Han et al. (2022) is an exception, but it was trained only on rough, synthesized data;
- Most models are restricted to training on data from a single geographic location;
- With the exception of SolarGan, existing models typically produce only cumulative irradiation results for an entire year;
- Most research is based on parametrically synthesized building geometry, not on 3D models from real-world buildings, such as GIS data.

Author	Year	Name	Dataset	Input	Model	Prediction Type
Alammar A. et al.	2021	N/A	Surrogate envelopes	Categorical	ANN & Random Forest	Single façade patch
Lila A. et al.	2021	N/A	Surrogate envelopes	Categorical	ANN	Single roof
Tehrani et al.	2024	N/A	GIS data	Categorical	ANN	Average Cumulative Radiation
Yue et al.	2024	N/A	GIS data Zhengzhou	Categorical	17 common models	Cumulative Radiation by surface orientation
Galanos T. & Chronis A.	2022	N/A	GIS data Vienna	Heightmap	GAN	Envelope ground
Huang C. et al.	2022	N/A	Surrogate envelopes	Categorical, heightmap	CycleGAN, pix2pix, ANN	Envelope ground
Han J.M. et al.	2022	CoolVox	Surrogate envelopes	3D occupancy grid	3DCNN	Facade and roof of single building
Nakhaee A. et al.	2023	Deep-Radiation	GIS data New York	Sem. segm. And depth-map fish-eye perspective	GAN with pix2px generator	Envelope ground
Zhang Y. et al.	2023	SolarGAN	GIS data Zurich and Singapore	Fish-eye perspective and location	GAN with VAE generator and more	Solar Irradiation time series

Table 2.1.: Related research papers solar irradiation prediction.

2.4. Neural Network Architectures with 3D inputs

In the following paragraphs, several network architectures that are capable of processing 3D models will be discussed, focusing on different approaches to address input dimension irregularities.

2.4.1. VoxNet

One intuitive method for processing 3D data is through a preprocessing step called voxelization. Voxelization involves discretizing a 3D object into a 3D array of voxels. These voxels form a regular grid, making them suitable as training data for neural networks. The features stored in each voxel can include geometry occupancy, average normal direction, or color.

VoxNet, developed in 2015, was designed to classify 3D objects using voxel clouds as input (Figure: 2.12). In VoxNet, the inputs are referred to as Volumetric Occupancy Grids, where each voxel indicates whether geometry exists within the voxel's boundaries. The architecture of VoxNet resembles that of a traditional CNN; however, instead of 2D convolutions with a third dimension holding feature maps, VoxNet employs three-dimensional convolutions, adding a fourth dimension to represent features. Pooling is achieved by replacing 2D pooling layers with 3D non-overlapping blocks (Maturana & Scherer, 2015).

The scalability and generalization of VoxNet are constrained by the model's substantial memory consumption. As the input resolution of the voxel grid increases, memory requirements grow cubically. According to experiments conducted by Wang et al. (2017), VoxNet was only capable of handling models with a maximum resolution of 64^3 voxels on an 8GB GPU. A potential solution to this limitation, as suggested by Han et al. (2022), is to use an Octree Convolutional Neural Network or Transformer network to reduce memory consumption.

Publication

VoxNet is available in TensorFlow and Pytorch.

2.4.2. OCNN and OctNet

To reduce the memory and computational costs associated with VoxNet, the Octree Convolutional Neural Network (OCNN) was developed. There are several implementations of octree-based CNNs, with the most notable being OCNN by Wang et al. (2017), OctNet by Riegler et al. (2016), and the octree network by Wang et al. (2020). The key differences between these models lie in the methods used to convert regular voxel clouds into octrees and the specific implementation of convolution and pooling layers.

For simplicity, this discussion will focus solely on the implementation of OctNet, as it is considered the easiest to comprehend and has been thoroughly compared to the performance of VoxNet. As mentioned, an octree-based CNN requires an octree data structure as input. The octree data structure can be most easily understood by considering its two-dimensional counterpart: the quadtree.

To illustrate, imagine a sparse two-dimensional grid containing data points with corresponding features. A quadtree can be constructed by recursively dividing the grid into four subsections,

2. Related Literature

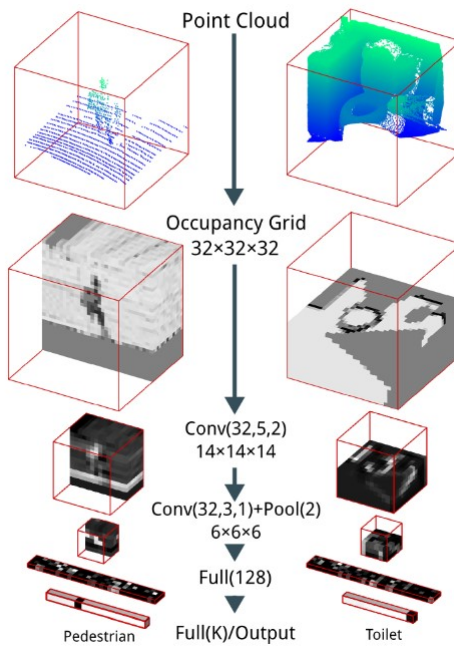


Figure 2.12.: VoxNet model encoder which downscales features through voxelization. (Image by Maturana and Scherer (2015))

with pointers indicating the parent-child relationships between nodes. The method of storing this data may vary depending on the implementation, but visually, it can be represented as a hierarchical graph (see figure 2.13).

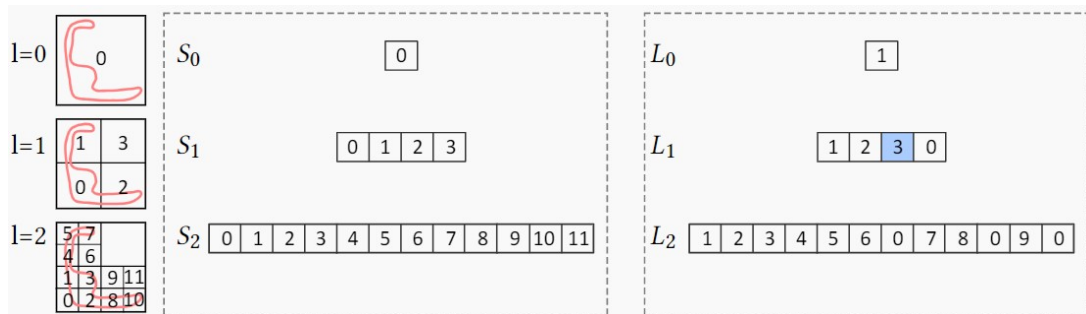


Figure 2.13.: Quadtree explained. Left: image subdivided in quadrants. Center: corresponding leaf node indices. Right: leaf nodes replaced by 0 when no occupancy. (Image by Wang et al. (2017))

Similarly, a voxel grid can be divided into eight equal subparts to convert it into an octree. The primary advantage of octrees is their ability to significantly reduce the memory size required for storing sparse voxel clouds. However, it is important to note that not all octree-based CNNs use the same type of input. In the case of OctNet, a hybrid octree with a specified maximum depth was developed to address the challenge of efficiently accessing underlying data.

OctNet and [OCNN](#) are not limited to using occupancy data as input; they can also utilize additional features such as normal directions and colors. This flexibility allows for the use of irregularly sized point clouds or meshes as input, provided they are first converted into an octree structure. In experiments, OctNet was tested on both classification and segmentation tasks, demonstrating performance that was either comparable to or better than VoxNet. Moreover, OctNet was capable of processing grids with sizes up to 256^3 , a feat that was not achievable with VoxNet.

Publication

[OCNN](#) is publicly available to use in Pytorch. OctNet is currently only available in C++.

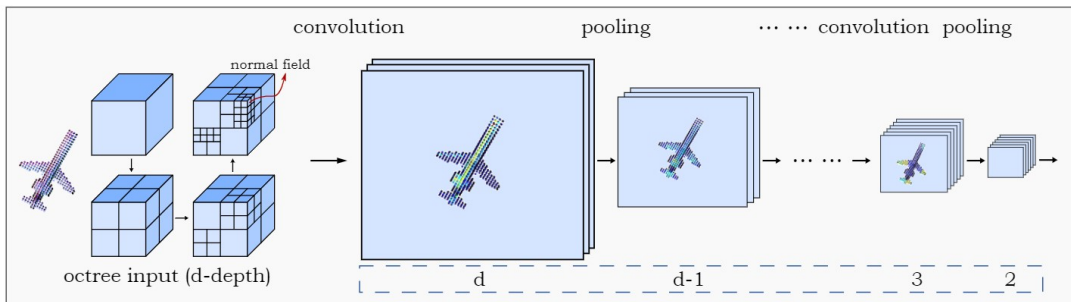


Figure 2.14.: Visual representation of OCNN model. (Image by Wang et al. (2017))

2.4.3. MeshCNN

Hanocka et al. (2019) propose that 3D shape predictions can be made by directly feeding a neural network with a mesh—an approximate representation of a shape using vertices and faces. The primary advantage of using meshes as network input lies in their ability to better preserve fine details and sharp edges of shapes. Moreover, using meshes requires minimal preprocessing, making it a more efficient approach for certain tasks. MeshCNN addresses the challenge of irregular input data by employing specially designed convolutional and pooling layers that operate directly on the edges of the mesh.

In practice, however, the use of MeshCNN is expected to be limited by several factors. Firstly, the network can only process manifold meshes, which are meshes where every edge belongs to exactly two faces. Although manifold mesh datasets are available for both classification and segmentation tasks, it is common in practice to encounter geometry that is not fully manifold. Secondly, MeshCNN is restricted to shape features as input, meaning that the mesh is represented as a collection of edges along with some of their associated features.

The study by Hanocka et al. (2019) demonstrates that MeshCNN is suitable for both classification and segmentation problems.

Publication

The implementation of MeshCNN is publicly available in PyTorch, making it accessible for further research and application.

2. Related Literature

2.4.4. Multiview CNN

An intriguing approach to address the irregularity of 3D geometry involves using 2D rendered images of an object from multiple perspectives. Conceptually, this approach is straightforward: 2D images are rendered from various views of a model, and a traditional CNN is employed to analyze each pixel.

MVCNN (Ma et al., 2017) utilizes this multiview representation as the model's input. The camera perspectives can be rendered either from consistent positions relative to the model or from more random positions. When these images are fed into the model, they first pass through a convolutional layer. The views are then aggregated and processed through a second convolutional layer for classification purposes.

Publication

As of the time of writing, there is no implementation available for segmentation tasks.

2.4.5. PointNet

Another approach for handling 3D geometry involves using point clouds, which are sets of points in three-dimensional space that outline the boundaries of an object. Research by Qi et al. (2016) demonstrated that it is feasible to train neural networks directly on such 3D point clouds. In the initial version of the PointNet model, the network is trained on batches of point clouds with a consistent size. The input can also include additional dimensions, such as normal vectors or color information. Once trained, the network can classify or segment 3D point clouds with the same number of input points as used during training.

PointNet++ (Qi et al., 2017), an advanced version of the original PointNet model, is a hierarchical neural network that subsamples points from the original point cloud using a neighborhood ball. These subsamples, which are overlapping regions of points, collectively describe the original geometry. The smaller version of the PointNet model is then used to make predictions on these subsets of points, with weights that can be shared across the subsamples. PointNet++ employs a recursive subsampling approach, where each step processes fewer points, thereby focusing on a smaller region. Finally, PointNet++ includes a grouping layer that concatenates information from all detail levels.

Qian et al. (2022) propose an advanced version of PointNet++ called PointNeXt, which introduces optimized training strategies and several new layers, making the model more scalable. Their research indicates that PointNeXt outperforms Transformer-based point cloud models in several experiments.

Other researchers, such as Zhao et al. (2021), suggest that Transformer-based neural networks can also be utilized to classify and segment point clouds. Due to their permutation invariance, Transformer models are naturally suited for handling point clouds. However, Transformer-based networks are generally known for their complexity in fine-tuning and their computational expense.

Publication

All versions of PointNet are available in both Pytorch and Tensorflow.

2.4.6. Model Discussion

All the models discussed have potential for implementation in predicting solar irradiation on 3D urban geometry, albeit with varying degrees of required modification. However, several factors could limit their performance and practical usability.

Limitations

- **3DCNN networks (e.g., VoxNet):** While 3DCNNs like VoxNet could theoretically be adapted for irradiation prediction, they are hindered by poor scalability. The primary limitation is memory consumption; the number of values needing prediction far exceeds what was proposed in the original VoxNet paper, making these models less viable for large-scale irradiation prediction tasks.
- **Octree-based networks:** These networks present a promising alternative to traditional 3D CNN due to their more efficient handling of sparse data. The main challenges include converting a mesh to the octree data structure and accurately indicating where irradiation predictions are required. Addressing these challenges is crucial for successful implementation.
- **Mesh-based Convolutional Neural Networks:** Mesh-based CNNs pose several challenges for irradiation prediction. Firstly, the input mesh must be manifold, which is not always guaranteed, especially in datasets derived from scanned building data. Additionally, the mesh would need to be restructured so that the positions of the faces correspond to the expected irradiation values. Due to these preprocessing demands, MeshCNN may not be the most practical choice for this application.
- **Multiview-based networks:** These networks face significant challenges in both preprocessing and postprocessing stages. Converting 3D meshes and irradiation values into 2D perspective views could lead to a loss of important information, given the limited perspectives. Moreover, the subsequent step of reconstructing a 3D model from the 2D images could introduce substantial overhead in processing time, further complicating their use in practical applications.
- **PointNet-based networks:** PointNet models require the input dataset to be converted into point clouds. This conversion could be rapid if points are sampled randomly, but it may become slow and cumbersome if points need to be regularly spaced in a grid. However, a significant advantage is that the point cloud not only represents the geometry but also indicates the positions where irradiation values are expected, making PointNet a more tailored solution.

2.4.7. Conclusion

From the literature review, both Octree-based CNNs and PointNet-based networks appear to be viable solutions for predicting solar irradiation. However, given the superior performance of PointNeXt in point cloud segmentation tasks, this research will focus on implementing PointNet-based models for irradiation prediction.

3. Methods

The methods of this project are divided into five distinct steps: generation, simulation, parallelization, prediction, and interaction. This section outlines each step, detailing the geometric data used, the simulation processes, the deep learning model development, and the framework for user interaction. Additionally, it discusses the parallelization techniques implemented and the software employed throughout the project.

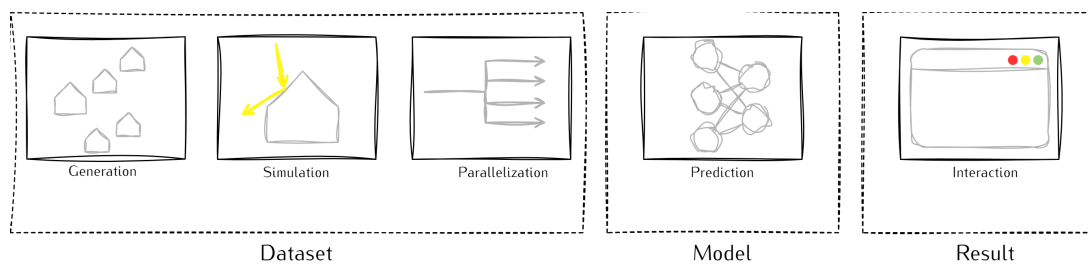


Figure 3.1.: Workflow steps for this project: generation, simulation, parallelization, prediction and interaction (Image by author)

The generation step focuses on the geometric data utilized, its retrieval methods, and necessary preprocessing. This phase involves defining the types of geometric data required and outlining how this data is sourced and prepared for subsequent stages.

In the simulation step, an overview of the simulation techniques employed is provided. This includes a description of the simulation type, the parameters chosen, and the rationale behind these selections. This phase ensures that the simulation outputs are accurate and relevant to the prediction models.

Parallelization techniques are discussed to address the computational demands of the project. This includes strategies for optimizing performance and managing resources efficiently, ensuring that the dataset generation and simulation processes are completed in a reasonable time frame.

Prediction encompasses the development and training of deep learning models. This section details the models used, the training process, and the evaluation metrics employed. The goal is to create robust prediction models capable of delivering accurate results based on the geometric data and simulation outputs.

The interaction step involves designing a framework for user engagement with the prediction model. This framework facilitates the user's ability to interact with and utilize the prediction results effectively. The design and functionality of this framework are crucial for making the prediction models accessible and user-friendly.

3. Methods

3.1. Framework

Initially, the framework was initially sketched in McNeel Rhino Grasshopper, and a single dataset sample was generated. However, Grasshopper presented several challenges, such as limited integration with Python 3-based packages and difficulty in parallelizing code due to memory overhead.

To overcome these limitations, the project was transitioned to external Python code development. This approach allowed for better integration with state-of-the-art AI development packages and more efficient parallelization. The main software and packages selected for the workflow include:

- Grasshopper for workflow experimentation and visualization;
- GHPython for data IO need for visualization;
- Python 3.10.14 for LBT-tools Honeybee Radiance and Rhino.Inside for dataset generation;
- AcceleRad, a GPU version of Radiance for simulation;
- Python 3.10.14 with Pytorch for irradiation prediction;
- Weights and Biases for hyperparameter tuning;
- Matplotlib for visualization intermediate training steps;
- The code of this thesis will be available on Github.

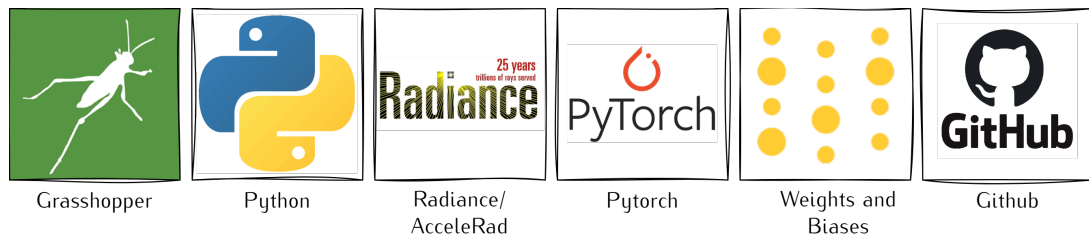


Figure 3.2.: Software used for this project (logo's by respective developer)

3.2. Generation

Generally speaking, researchers strive to use real physically observed data to train their neural models. Although there are no dataset available which contain irradiation data on buildings, the number of accurate urban 3D datasets have increased over the last few years.

3.2.1. Geometry source

The project utilizes the 3D BAG dataset (Peters et al., 2022), which provides 3D representations of cities across the Netherlands in various file formats. This dataset is primarily based on the Algemeen Hoogtebestand Nederland (AHN), which was obtained through LiDAR technology.

The 3D BAG dataset includes three levels of detail (LoD), each representing different accuracies and details in urban geometry. For this project, the lowest Level of Detail (LoD) was chosen. This level features building representations with irregular vertical facades and simplified roofs with flat surfaces.

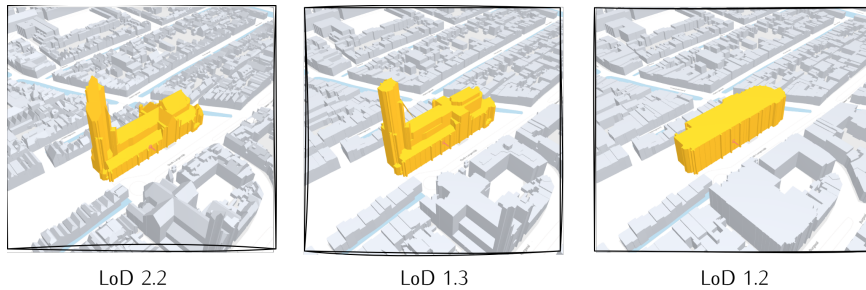


Figure 3.3.: Different 'level of details' (LoD) of the city center in Delft (Image by author)

The 3D BAG dataset is available in two formats: CityJSON and .obj. The .obj format was selected for this project. A custom Python script was used to convert .obj files into triangular meshes, which were then visualized using McNeel Rhino and Grasshopper. Manual downloading of data from the 3D BAG can be time-consuming due to numerous download links corresponding to specific city coordinates. To streamline this process, a web scraper was developed. The download links for the 3D BAG are named according to the size of the city block, horizontal coordinate, and vertical coordinate. By iterating over the possible names, the scraper efficiently downloaded all urban geometry from coordinates 8-208-480 to 8-392-568 (based on BAG version v2024.04.20 beta).

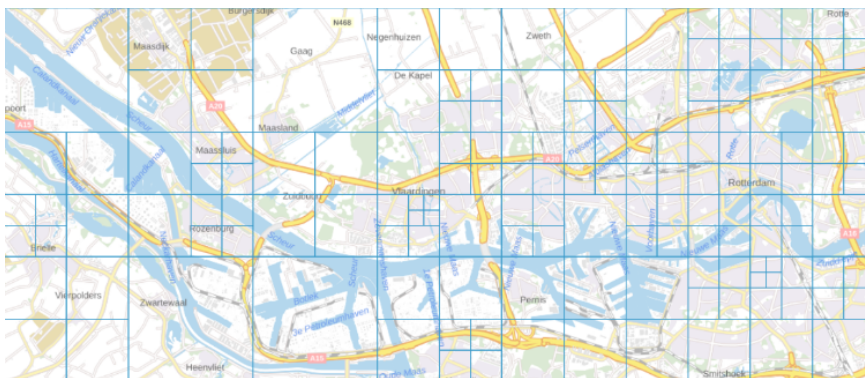


Figure 3.4.: Overview of the tiling system for downloading 3D BAG regions (Image by author)

3. Methods

3.2.2. Partitioning

For training the neural network on building geometry, the dataset was partitioned into smaller patches, similar to previous research on solar irradiation prediction. Two patch sizes were used: 100 x 100 meters and 300 x 300 meters. Prior studies have indicated that using 100 x 100 meter samples is necessary due to the limited processing capacity of the model. Additionally, the pre-processing time for the samples does not scale linearly with sample size. Therefore, maintaining smaller patch sizes is crucial to manage preprocessing time and computational efficiency.

3.2.3. Augmentation

Data augmentation is a potentially useful step in the generation process of this methodology, as it can reduce preprocessing time and enhance the diversity of the training data. For the training of neural networks, various augmentation techniques can be applied. In this project, three primary augmentation approaches were considered:

- Translation: This involves shifting the outlines of the samples along the x and y axes. By doing so, more samples can be extracted from a single 3D [BAG](#) file, effectively increasing the dataset size without additional data acquisition.
- Rotation: Rotating the sample changes the orientation of the geometry. This approach ensures that the neural network learns to recognize buildings from different directional perspectives, mimicking real-world variations in orientation.
- Scaling: Scaling involves enlarging or reducing the size of the buildings. This method helps the model generalize to buildings of various sizes.

In the implementation of this project, the first two augmentation methods—translation and rotation—were employed. The scaling method was excluded because altering the building sizes would result in geometries that do not accurately represent real-world dimensions, potentially leading to confusion for the model. Moreover, the other two augmentation techniques were not used in the final dataset development. It was determined that allowing additional computation time was acceptable in exchange for obtaining more unique and realistically oriented building samples, as will be described in the discussion of this thesis.

3.2.4. Point Sampling

As described in the previous chapter, it is required to convert the mesh geometry to point clouds, to be able to feed them to PointNet based networks. For this research, two approaches have been implemented:

- Regular Point Sampling
In regular point sampling, the points are arranged in a regular grid to the greatest extent possible. This process necessitates extensive preprocessing, during which the meshes are discretized into quad mesh faces with dimensions approximately 1x1 meter. It is anticipated that neural networks will more accurately predict irradiance values on a regular grid due to the consistency in point positions between samples. However, a notable drawback is the complexity involved in generating the grid. A detailed overview of this procedure is provided in [Appendix A](#).

- Random Point Sampling

Given that PointNet-based networks are invariant to input permutations, it is feasible to randomize point positions in an arbitrary order. Utilizing the Poisson Disk Point sampling method (Yuksel, 2015), as implemented with the Open3D Python library, allows for the efficient generation of random points with approximately equal spacing between them.

Other sampling techniques which could potentially be used for the prediction of irradiation are described in appendix B.

3.2.5. Final Format

The samples in the dataset are stored in numpy arrays using the .npy file format. Each row in the array describes a point, with the corresponding normal vector of the geometry. For visualization, the mesh corresponding to the points is stored using a JSON string, which can be read by McNeel Rhino.

$$\begin{bmatrix} x_0 & y_0 & z_0 & u_0 & v_0 & w_0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_n & y_n & z_n & u_n & v_n & w_n \end{bmatrix}$$

Figure 3.5.: $n \times 6$ matrix with x, y, z, and normal u, v, w point cloud data.

3.3. Simulation

After generating the geometric samples and corresponding point clouds, the annual irradiation values can be computed using simulation software. Specifically, the GPU version of Radiance, AcceleRad, was employed for these calculations. To facilitate parallelization and enhance processing speed, the simulation was conducted directly in a Python 3 wrapper, rather than through Grasshopper.

The simulation requires four types of data inputs: a geometric mesh, material properties, sensor point locations, and weather data (figure: 3.6). To optimize performance, the complexity of the original mesh was minimized to the lowest number of faces feasible, which simplifies the construction of an octree graph inside AcceleRad. Sensor points were defined using the previously generated point clouds, and the weather data was provided in the form of an .epw file for Amsterdam¹.

To simplify the project, only one material with the following properties was used: reflectance of 0.2, specular of 0.0, and roughness of 0.0. As of the time of writing, there is no available data on the materials used in the 3D BAG data. Additionally, incorporating multiple materials would complicate the prediction process using neural networks. Future research may address the integration of material data into the simulation.

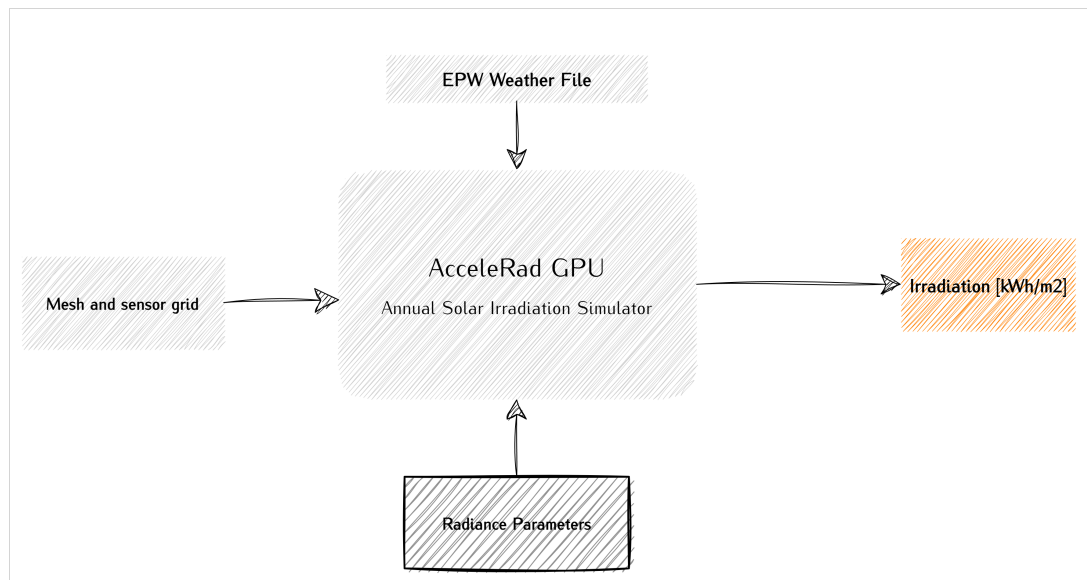


Figure 3.6.: Simulation principle using Accelerad. Materials have not been included due to lack of data. (Image by author)

¹Epw files can be downloaded from: <https://www.ladybug.tools/epwmap/>

3.3.1. Parameter Convergence Test

Before conducting the simulations, parameters for the annual solar irradiation simulation have to be selected. To facilitate this process, an automated convergence test script was developed. Initially, a set of high parameter values was chosen, under the assumption that these values would best approximate real light behavior.

Subsequently, the parameters were substantially scaled down. In total, 128 simulations were executed, with parameters being incrementally adjusted. Both GPU (AcceleRad) and CPU (Radiance) simulations were performed to compare their performance.

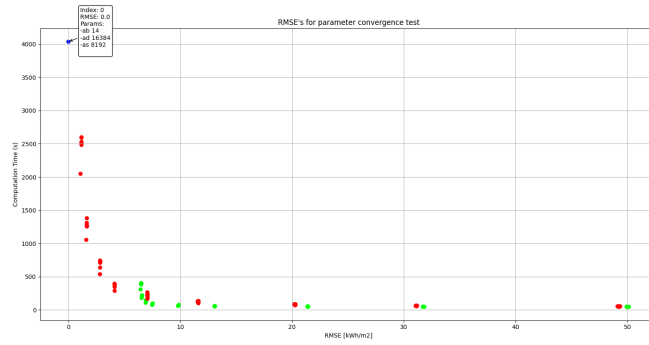
The results of all simulations are presented in the following plot (figure 3.7). The y-axis represents the runtime of the simulations, measured for both CPU and GPU configurations. The x-axis displays the errors, which are calculated as the difference between a given set of parameters and the baseline simulation benchmark. The Root Mean Squared Error (RMSE) was used as the metric to quantify this difference. The plot also features interactive capabilities, allowing users to view the specific parameters associated with each sample by hovering the mouse over the plot.

As can be seen in figure 3.7c, maximum errors for GPU based simulations are significantly higher than CPU based simulations. However, the standard deviations suggest only small difference between CPU and GPU. Visual representations of the errors as can be seen in figure 3.7, suggest that the maximum errors in GPU based simulations are only outliers. Therefore, the GPU parameters at the lower end of the elbow (figure 3.7b) were selected as best ratio between time and error.

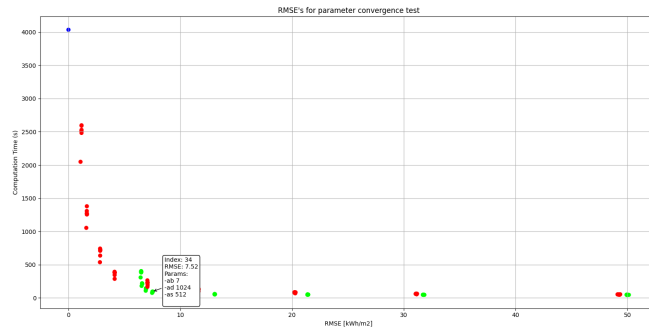
Parameter	Synthesized ground truth	Selected
Ambient Bounces (-ab)	14	7
Ambient Divisions (-ad) ²	16384	1024
Ambient Super Samples (-as)	8192	512
Sampling (-c)	1	1
Direct Certainty (-dc)	0.75	0.75
Direct Pretest Density (-dp)	512	512
Direct Relays (-dr)	3	3
Source Substructuring (-ds)	0.05	0.05
Direct Thresholding (-dt)	0.15	0.15
Limit Reflection (-lr)	8	8
Limit Weight (-lw)	4e-07	4e-07
Specular Sampling (-ss)	1.0	1.0
Specular Threshold (-st)	0.15	0.15
Sky Density	145	145

Table 3.1.: Selected and benchmark parameters for Radiance and AcceleRad settings.

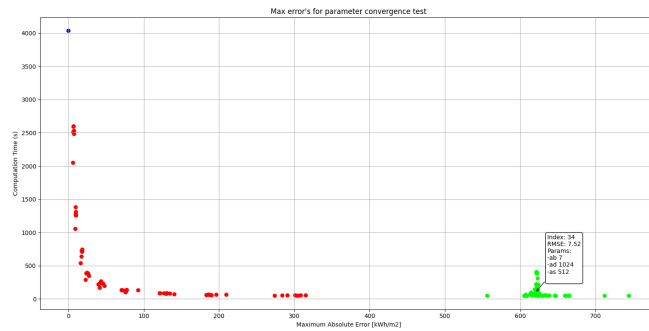
3. Methods



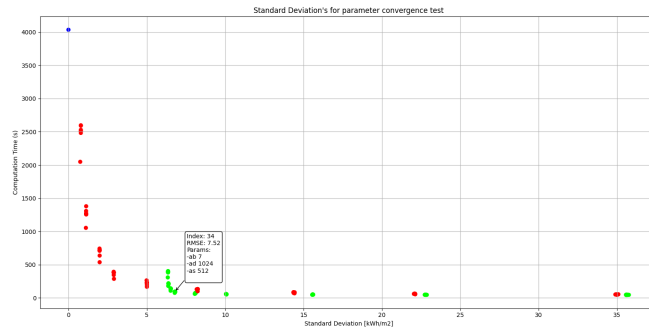
(a) AcceleRad parameters synthesized ground truth



(b) Selected AcceleRad parameters



(c) Maximum error in comparison to synthesized ground truth



(d) Standard deviation in comparison to synthesized ground truth

Figure 3.7.: Parameter Convergence Test for annual solar irradiation simulations. The blue dot indicates the synthesized ground truth, red the CPU Radiance-based tests and green the GPU AcceleRad-based simulations. In consecutive simulations, only the ambient bounces, ambient divisions and ambient super samples have been changed.

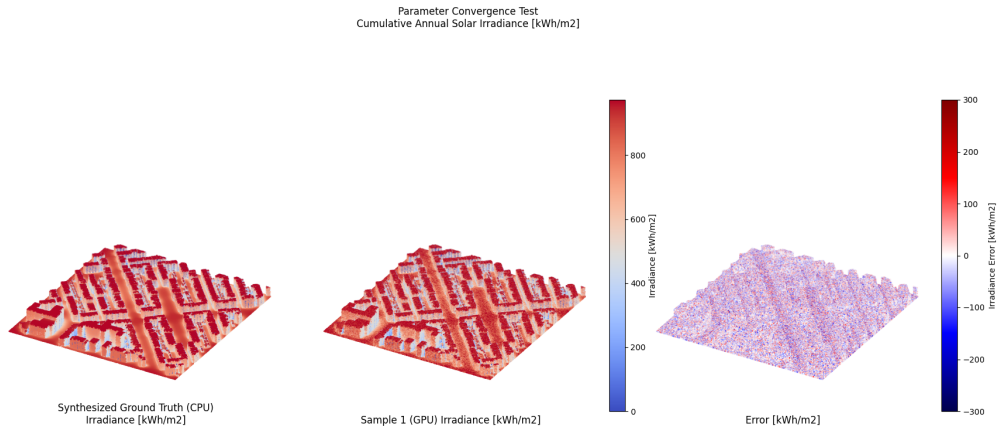


Figure 3.8.: Parameter Convergence Test for sample 1 with parameters:
`-ab 3 -ad 32 -as 16 -c 1 -dc 0.75 -dp 512 -dr 3 -ds 0.05 -dt 0.15 -lr 8 -lw 4e-07 -ss 1.0 -st 0.15 -ag -1 -w`

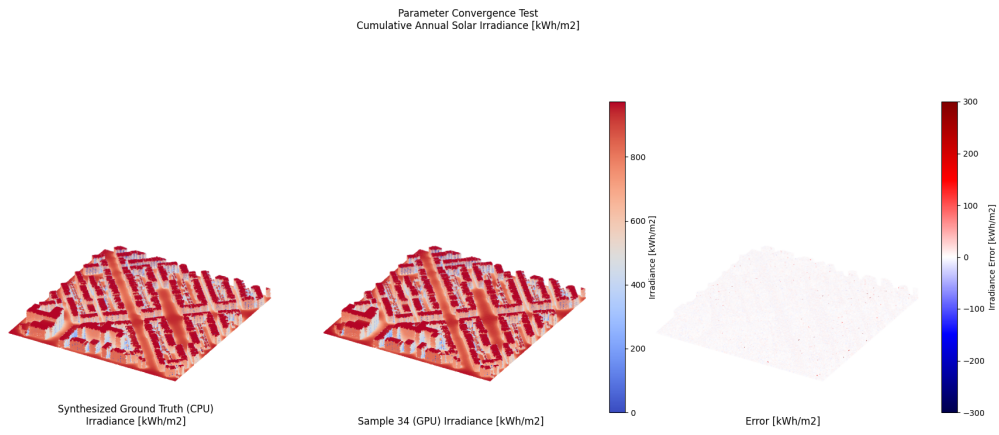


Figure 3.9.: Parameter Convergence Test for sample 34 with parameters:
`-ab 7 -ad 1024 -as 512 -c 1 -dc 0.75 -dp 512 -dr 3 -ds 0.05 -dt 0.15 -lr 8 -lw 4e-07 -ss 1.0 -st 0.15 -ag -1 -w'`

3. Methods

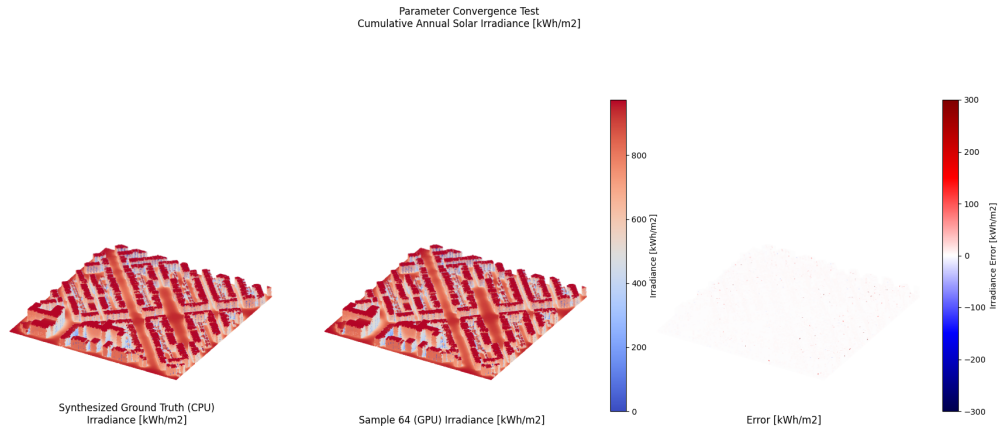


Figure 3.10.: Parameter Convergence Test for 64 with parameters:
`-ab 12 -ad 8192 -as 4096 -c 1 -dc 0.75 -dp 512 -dr 3 -ds 0.05 -dt 0.15 -lr 8 -lw 4e-07 -ss 1.0`
`-st 0.15 -ag -1 -w`

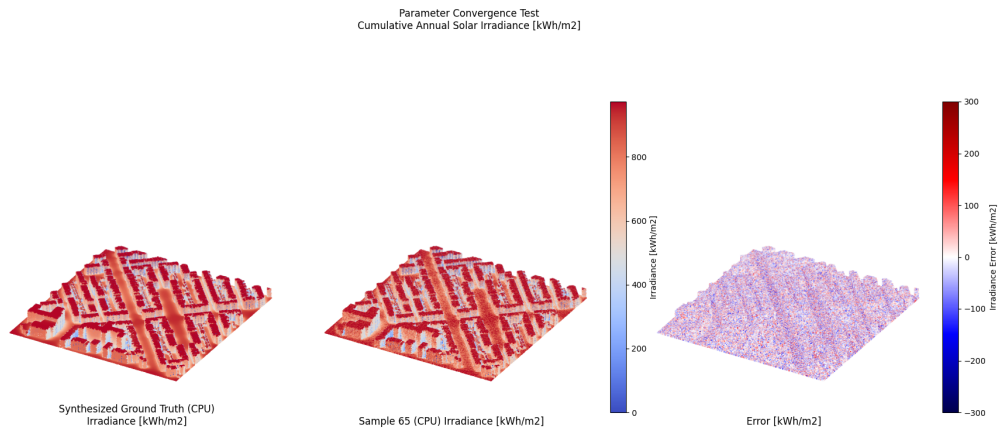


Figure 3.11.: Parameter Convergence Test for sample 65 with parameters:
`-ab 3 -ad 32 -as 16 -c 1 -dc 0.75 -dp 512 -dr 3 -ds 0.05 -dt 0.15 -lr 8 -lw 4e-07 -ss 1.0 -st`
`0.15 -w -g`

3.3.2. Final Format

After simulation, the irradiation values are added as column to the numpy data array. Together with the x,y,z coordinates and the u,v,w normal directions of the mesh faces, it results in a 'number of points' x 7 (32-bit float) shaped matrix (figure 3.12).

$$\begin{bmatrix} x_0 & y_0 & z_0 & u_0 & v_0 & w_0 & E_{i;0} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_n & y_n & z_n & u_n & v_n & w_n & E_{i;n} \end{bmatrix}$$

Figure 3.12.: $n \times 7$ matrix with x, y, z, and normal u, v, w point cloud data. The last column indicates the irradiation values.

3.4. Parallelization

Given the CPU and GPU-intensive nature of the workflow described, parallelization of both the dataset generation and simulation processes is crucial. Consequently, the project code was implemented not only in Grasshopper but also in Python 3, utilizing the Rhino.inside library along with the Honeybee Radiance package.

Dataset synthesis was parallelized using Python's built-in 'multiprocessing' library³. During development, the optimal multiprocessing approach was not immediately evident. Ultimately, it was determined that the *pool.imap_unordered* method was the most effective. This method processes an iterable of samples across available CPU cores in an unordered fashion. It was also found that limiting the number of tasks per CPU process (*maxtasksperchild*) to a low value, such as 1, is essential. Higher values lead to memory overflow issues after approximately 10,000 iterations due to inefficient memory garbage collection.

```
# Generation and simulation process function
process = generate_and_simulate_samples()

# Number of physical CPU cores
cpus = 18

# Iterable of 3D BAG .obj files
args = list_of_obj_files

# Initialize the pool
with multiprocessing.Pool(processes=cpus, maxtasksperchild=1) as pool:
    pool.imap_unordered(process, enumerate(args))
    pool.close()
    pool.join()
```

Figure 3.13.: Python code to run generation and simulation in parallel using the multiprocessing library.

The desktop PC was able to generate about 10.000 100x100 regular samples in 24 hours of computing, including the simulation of the irradiation values. An overview of the entire multiprocessing system is shown in figure 3.14.

As described earlier, the execution of this code was only possible on desktop based on a Windows non-server operating system. In this thesis, a high performance desktop was used with two Intel Xeon E5-2640 v4 CPUs and two NVIDIA Quadro M6000 24GB GPUs.

³<https://docs.python.org/3/library/multiprocessing.html>

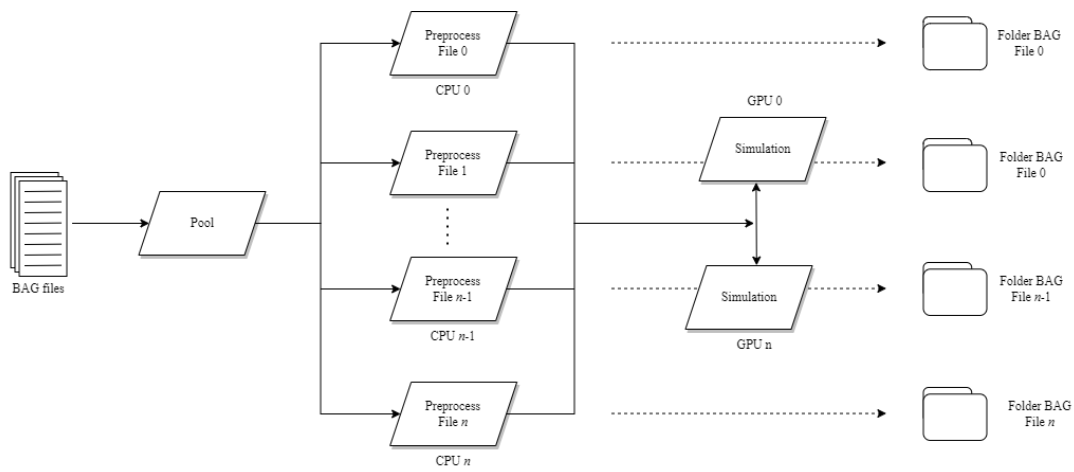


Figure 3.14.: Parallelization workflow for dataset generation and simulation (Image by author)

3.5. Prediction

As previously outlined, the synthesized datasets consist of samples structured as matrices with dimensions $num_points \times 7$, encompassing coordinates, normal directions, and irradiation values. The primary objective of this research is to predict the final column of the sample data, which contains the irradiation values, by utilizing the first six columns as input features for a deep neural network model. To achieve this, several PointNet-based networks and configurations were trained and evaluated to determine the extent to which a model can accurately predict irradiation.

In the following section, an overview of the methods and strategies employed for solar irradiation prediction will be provided. This section begins with an exploration of the backbone models used for prediction. Subsequently, challenges related to the dataset are addressed. The chapter concludes with a detailed overview of the model's training, validation, and testing processes, including hyperparameter optimization and metric selection.

3.5.1. PointNet

PointNet (Qi et al., 2016) is a deep neural network initially developed for the classification and (part)segmentation of point clouds. An overview of the network architecture is presented in figure 3.15. PointNet utilizes two transformation networks as symmetric functions to ensure that the input points are invariant to permutation and to extract global features. These transformation networks are followed by several fully connected layers that predict the most likely class to which an object belongs. For segmentation tasks, additional layers are included, where local point features and global features are concatenated for each point. These concatenated features are then passed through multiple multilayer perceptrons, which output the final result using a one-hot encoding. Each entry in the one-hot encoding corresponds to a potential class label. Since the segmentation version of PointNet predicts per-point values, this model was used as base

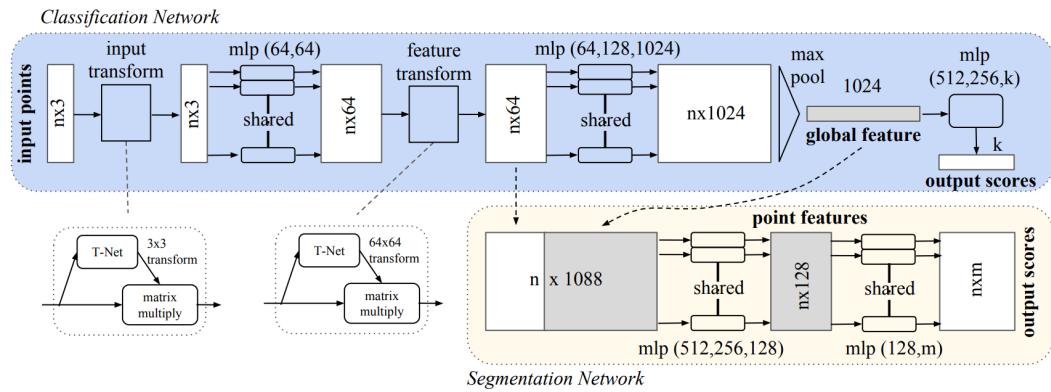


Figure 3.15.: The PointNet architecture as proposed by Qi et al. (2016)

for irradiation prediction. Furthermore, the following changes were made to the base architecture, to use it for this project:

- The softmax layer at the end of the model was removed. No activation function has been applied to the final layer;
- The one-hot encoding vector was reduced to only one output value;

As loss function, the MSE was used, which can be described by the following equation:

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2 \quad (3.1)$$

In which N describes the total number of samples, y_i the true value for the i th sample and \hat{y}_i the predicted value for the i th sample.

Training PointNet in batches requires inputs to be fixed-sized arrays. Therefore, 10,000 points were subsampled from the original points clouds, to make prediction. Inference after training also allows feeding differently sized inputs.

3.5.2. PointNet++

The PointNet++ architecture (Qi et al., 2017) extends the original PointNet model with the objective of capturing local context at varying scales. By employing a hierarchical feature learning framework, PointNet++ can effectively process point clouds of different sizes. The architecture is composed of three primary layers: sampling, grouping, and the PointNet layer.

In the sampling layer, points are not sampled in a random order. Instead, the Farthest Point Sampling (FPS) algorithm is utilized. In this approach, an initial random point is selected, and each subsequent point chosen is the one farthest from the previous point. This process continues until all points have been selected, ensuring a more uniform coverage of the point cloud.

The grouping layer then selects subsets of points using a ball query algorithm, which identifies all points within a specified radius from a selected point. The number of subsets generated may vary depending on the size of the input point cloud.

Finally, the PointNet architecture is applied to predict values for each point within the subsets. For segmentation tasks, the features of the subsampled points are propagated back to all original points in the point cloud. Figure 3.16 illustrates the PointNet++ procedure for both classification and segmentation.

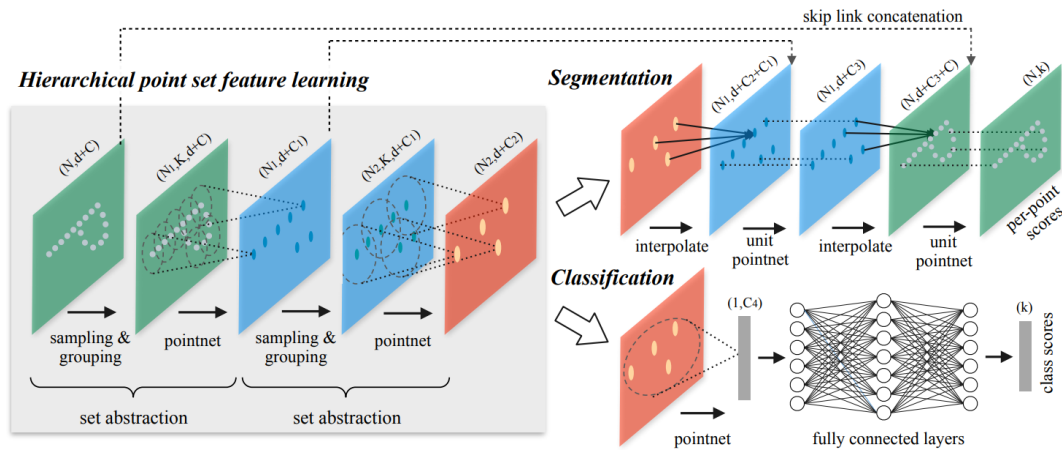


Figure 3.16.: The PointNet++ architecture as proposed by Qi et al. (2017)

3.5.3. PointNeXt

PointNeXt (Qian et al., 2022) builds upon and enhances the PointNet++ architecture, achieving optimized model performance through several key improvements. Firstly, state-of-the-art techniques for optimizers and learning rate schedulers were implemented, accompanied by further fine-tuning of hyperparameters. Secondly, it was discovered that reducing the radii used in the grouping layer led to a performance improvement of several percentage points. Lastly, various model scaling techniques were explored, resulting in even greater performance gains.

Given that PointNet++ is a relatively compact network, adjustments were necessary to scale the model effectively. To address this, an Inverted Residual Multi-Layer Perceptron (*InvResMLP*) was introduced after the Set Abstraction layers. Figure 3.17 illustrates these modifications in comparison to the original PointNet++ architecture.

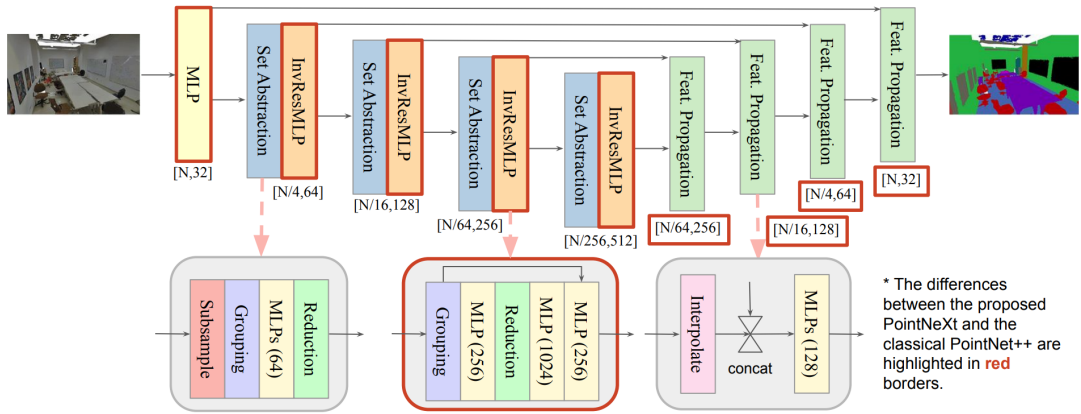


Figure 3.17.: The PointNeXt architecture as proposed by Qian et al. (2022)

The PointNeXt research offers models at four different scales: S, B, L, and XL, with the latter two being most relevant for this project. All four architectures were trained, validated, and tested to compare their performance.

For this project, the hyperparameters of the PointNeXt segmentation model served as a baseline for predicting solar irradiation. Subsequently, several hyperparameters were tuned specifically for the dataset, loss functions, and model.

The following section will describe the various layers of PointNeXt within the context of solar irradiation prediction based on point clouds, as required for this research. According to the original implementation, PointNeXt requires two inputs: batches of point clouds with x , y , z coordinates, and corresponding features derived from positional coordinates and normal values.

Encoder

The encoder's initial layer consists of a shared **MLP** (Conv1D block) that maps the input features to a higher-dimensional space without reducing the size of the point cloud. Following this, a SetAbstraction block, based on the PointNet++ architecture, is employed. The SetAbstraction block is composed of four key components:

Subsampling

Points are subsampled from the input based on the **stride**⁴. For instance, a stride of four reduces the number of points by a factor of four. As illustrated in figure 3.17, this subsampling process occurs four times across four SetAbstraction blocks, reducing an original point cloud of 10,000 points to 39 points by the end of the encoder block.

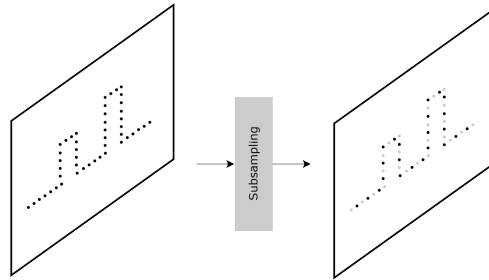


Figure 3.18.: Subsampling layer in the PointNeXt network (Image by author).

Grouping

Two hyperparameters, **radius** and **nsample**, define the grouping process. Points within the specified radius around the subsampled points are selected, with the number of points determined by nsample. The features of these grouped points are aggregated, optionally incorporating features from local skip connections.

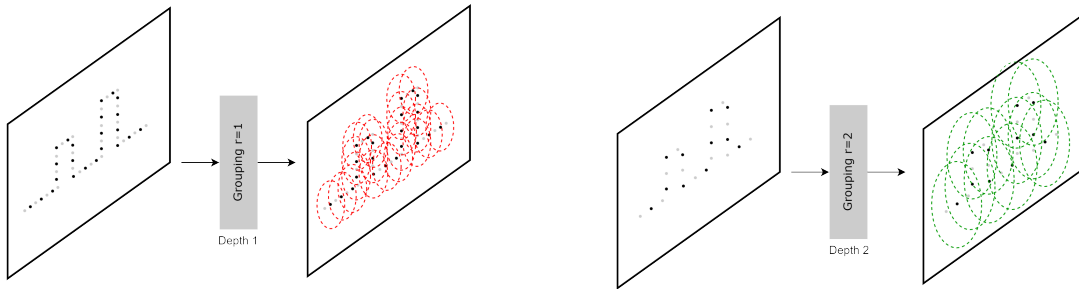


Figure 3.19.: Grouping layer in the PointNeXt network (Image by author).

With each successive SetAbstraction block, the radius used for grouping is *doubled*. For example, starting with an initial radius of 0.1 and progressing through four SetAbstraction blocks results in a maximum radius of 0.8. If the dataset coordinates are normalized to the $[0,1]$ domain, the maximum receptive field for the SetAbstraction blocks is 80 meters in real units (given a sample size of 100x100m). Conversely, if the coordinates are normalized within the $[-1, 1]$ domain, the

⁴For clarity, first-mentioned hyperparameters are highlighted in bold.

3. Methods

receptive field extends to 40 meters. This relationship highlights the interdependence between dataset normalization and radius scaling.

MLP

The aggregated features undergo convolution using a Conv2D block, followed by 2D batch normalization and a [ReLU](#) activation layer. The number of output channels for these convolutions is governed by the **width** hyperparameter. With each new SetAbstraction block, the output channel size is doubled relative to the initial width.

Reduction

The features are reduced using a max-pooling layer, which condenses the grouping process into a single value.

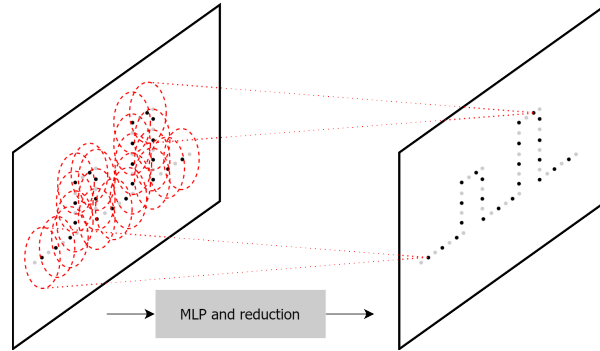


Figure 3.20.: MLP and reduction layer in the PointNeXt network (Image by author).

PointNeXt introduces the [InvResMLP](#) to enhance the scalability of the PointNet++ architecture. The [InvResMLP](#) operates similarly to the SetAbstraction block, with several notable differences:

1. The subsampling layer is omitted, ensuring that the point cloud retains the same size as the input.
2. A residual connection is introduced from the input to the output.
3. An additional [MLP](#) layer is appended before reduction, enabling pointwise feature extraction.
4. The block is repeated, with the output channel size multiplied each time, according to the **expansion** hyperparameter.
5. The grouping radius is doubled after the preceding SetAbstraction block.

As a result of the radius doubling, the final receptive field from the encoder expands to 160 meters in the $[0,1]$ point coordinate domain and 80 meters in the $[-1, 1]$ domain.

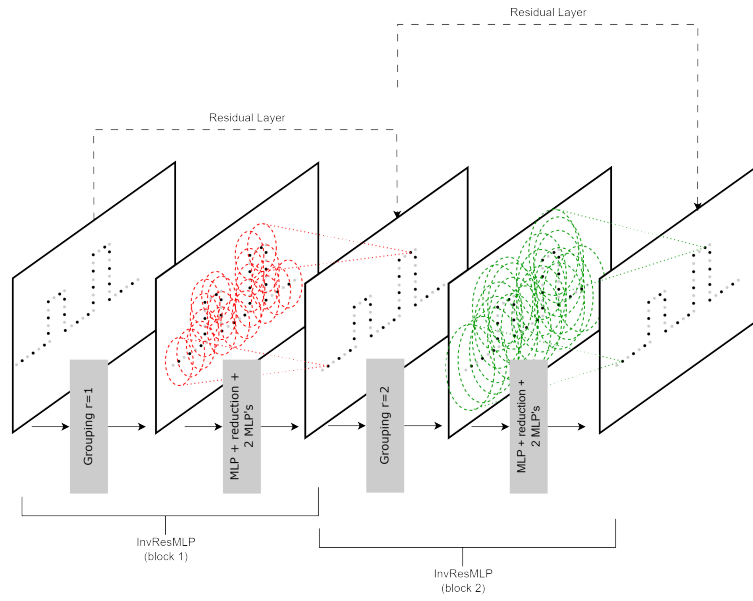


Figure 3.21.: InvResMLP layer in the PointNeXt network (Image by author).

3. Methods

Decoder

The decoder architecture in PointNeXt mirrors that of PointNet++. Subsampled points are interpolated to match the original sizes at each encoder depth, using an [MLP](#) composed of a Conv1D layer, one-dimensional batch normalization, and a [ReLU](#) activation layer.

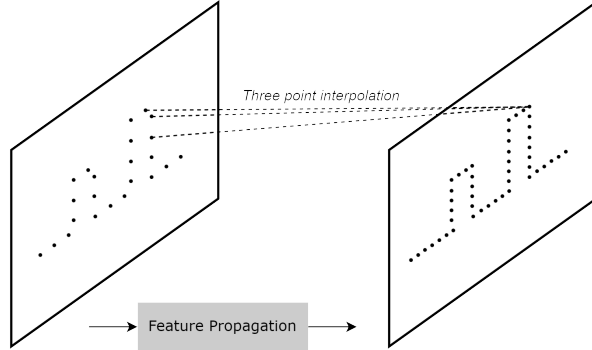


Figure 3.22.: Interpolation layer in the PointNeXt network (Image by author).

Head

The final head of the model begins with an [MLP](#) similar to those in the decoder, followed by a dropout layer. The feature channels are then reduced to an output size of 1, corresponding to a single irradiation value for each point.

3.5.4. Model Sizes

Without changing the architectures as suggested by Qian et al. (2022), the models exhibit the following number of hyperparameters: PointNet 3.6M, PointNet++ 1.0M, PointNeXt-S 0.8M, PointNeXt-B 3.8M, PointNeXt-L 7.1M, and PointNeXt-XL 41.6M.

3.5.5. Training, Validation and Testing

For this project, the implementation of the PointNeXt (Qian et al., 2022) source code was used as reference, using PointNet, PointNet++ and PointNeXt S, B, L and XL as base mark. The models were trained by feeding point clouds with x,y,z coordinates and normal directions for the geometry as feature inputs. For each point, the model regresses to a single value indicating the predicted annual solar irradiation.

Normalization and Centering

The dataset is randomly split into a training, validation and testing dataset, balanced to an 80%-10%-10% ratio respectively. Before feeding samples to the network, normalization of both inputs and ground truth output values is applied. Tests were performed with min-max normalization into the range $[0, 1]$ and $[-1, 1]$ for the output values. Similarly, point x,y,z coordinates were also normalized to the $[0, 1]$ and $[-1, 1]$ domain.

As described by Qian et al. (2022), subsets of points are renormalized after the Set Abstraction layers in PointNeXt, using a method called relative position normalization based on the neighborhood query radius.

Normal values u, v, w indicating the geometric orientation of the mesh faces were not normalized since they are already described in the $[-1, 1]$ domain.

Experiments were performed with and without the centering of point clouds. Avoiding centering results in geometric context (non-buildings) always having a z -value of zero.

Training Hyperparameters

For training, a batch size of 8 was used containing 10.000 points per sample, considering the fact that the dataset samples must have at least 10.000 points based on the 100m x 100m grid. In validation and testing, no point subsampling was applied. Similar to PointNext, Adamw was used as optimizer with a cosine learning rate scheduler.

The final model was trained for 100 epochs. Hyperparameter tuning runs were based on 25 epochs.

Training hyperparameters that have not been mentioned in this thesis, were derived from the PointNeXt source code.

Model Hyperparameters

Within the finetuning process of the model, several model hyperparameters were considered:

- width: the width of the output multi-layer perceptrons (doubling the width, quadruples the number of model parameters);
- nsample: the number of neighbors queried in each model block (since the point features of the neighbors are aggregated, the number of model parameters does not change);
- radius: the initial radius for the model point sampling;
- expansion: the expansion ratio of the InvResMLP block, as described by Qian et al. (2022);
- voxel max: maximum number of points sampled per point cloud in training;

Performance Metrics

Training, validation and testing performance were evaluated using both regression and classification metrics.

- In training, the loss is backpropagated using case specific loss functions: MSE, WMSE, Delta Loss and Reduction Loss;
- For performance comparison between different loss functions, the MSE is always computed for each training step;
- Finally the RMSE in kWh/m² is used as more indicative value for real-life usage.

Furthermore, the predicted and ground truth irradiation values have been classified by converting them to ten bins (0-100 kWh/m², 100-200 kWh/m², ..., 900-1000 kWh/m²). Based on these bins, classification scores can be used to measure performance.

3. Methods

- Using the classification bins, it was possible to compute accuracy, precision, recall and f1-scores;

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad (3.2)$$

$$\text{Precision} = \frac{TP}{TP + FP} \quad (3.3)$$

$$\text{Recall} = \frac{TP}{TP + FN} \quad (3.4)$$

$$\text{F1 Score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \quad (3.5)$$

in which TP (True Positives) is the number of correct positive predictions, TN (True Negatives) the number of correct negative predictions, FP (False Positives) the number of incorrect positive predictions and FN (False Negatives) the number of incorrect negative predictions.

- The performance per irradiation bin was evaluated using multi-class confusion matrices for each validation and test step.
- Micro/macro averaged F1-scores can be used for overall multi-class performance. In problems where each class (irradiation bin in this context) is equally important, the macro averaged F1-score should be used. Micro averaged F1-score can be seen as an overall accuracy considering all individual bins.

$$\text{Micro-averaged F1-Score} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN} \quad (3.6)$$

$$\text{Macro-averaged F1-Score} = \frac{1}{n} \sum_{i=1}^n \text{F1-score}_i \quad (3.7)$$

Finally, the overall performance was visually evaluated using colored point clouds, indicating the difference between predicted irradiation and ground truth.

3.5.6. Dataset Imbalance

As can be seen in figure 3.23, expected irradiation values are not equally distributed in the dataset. Most of the values tend to be relatively high, due to full exposure of the geometry to the sun. This dataset imbalance can result in the model seemingly performing well, but actually predicting higher values preferably.

Figure 3.23 shows a histogram with binned irradiation values. For the XL dataset on 100x100m samples with regular point clouds, 325 million points were sampled of which 46,7% (152 million)

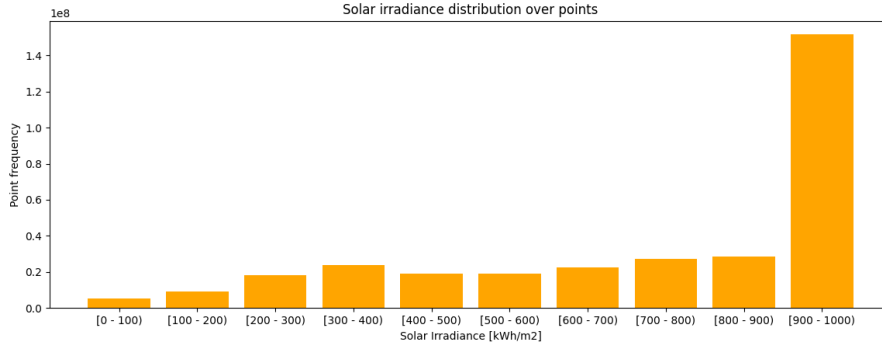


Figure 3.23.: Imbalance over the irradiation spectrum in the dataset

is part of the 900 kWh/m² to 1000 kWh/m² domain. This suggests a significant imbalance in the expected irradiation values of the dataset. To overcome this issue, several strategies have been explored. Firstly, one could consider to undersample the dataset. Simply said, samples are manually picked which have an equal distribution of irradiance values. However, in case of this research, this approach has two downsides:

- The limited size of the dataset is significantly reduced;
- The model is trained on atypical samples which do not necessarily represent common evaluation and test samples.

Alternatively, it is possible to use a weighted loss function, in which uncommon values get a higher penalty than common irradiance values. For this research, three loss functions have been designed which potentially overcome dataset imbalance:

1. Reduction loss is based on [MSE](#). However, the expected irradiation values are first converted to classes using bins. The bin with most points is then reduced to the size of the second-largest bin. Only a selection of points in the largest bin contribute to the overall loss of the prediction. This approach is deemed useful, since the irradiation dataset typically only has one bin which is significantly higher than the others.

A bin C_j can be mathematically be described as:

$$C_j = \{y_i | B_{j-1} \leq y < B_j\} \quad (3.8)$$

- C_j : A bin with irradiation values;
- y_i : True value for the i -th sample;
- B_j : Bin threshold j ;

where $B_1, B_2, \dots, B_n - 1$ denote the boundaries or thresholds of the bins. The bin with the maximum size C_{max} can be computed using:

$$C_{max} = \operatorname{argmax}_j n_j \quad (3.9)$$

The size of bin C_{max} is randomly reduced to the size of bin $C_{second-max}$.

The new point cloud with the reduced largest bin is used to compute the [MSE](#).

3. Methods

- Delta loss works similar to the well-known Huber loss. Just like Huber loss, some of the errors are penalized using the [MSE](#), other (mainly smaller) errors are penalized using l1 loss. However, in contradiction to Huber loss, delta loss makes a distinction between the [MSE](#) and l1 based on the expected outcome, not on the error itself. Therefore, it is possible to penalize irradiation values in larger bins with l1, and smaller bins with [MSE](#).

$$\text{Huber Loss} = \frac{1}{N} \sum_{i=1}^N \begin{cases} \frac{1}{2}(y_i - \hat{y}_i)^2 & \text{if } |y_i - \hat{y}_i| \leq \delta \\ \delta (|y_i - \hat{y}_i| - \frac{1}{2}\delta) & \text{if } |y_i - \hat{y}_i| > \delta \end{cases} \quad (3.10)$$

$$\text{Delta Loss} = \frac{1}{N} \sum_{i=1}^N \begin{cases} (y_i - \hat{y}_i)^2 & \text{if } |y_i| \leq \delta \\ |y_i - \hat{y}_i| & \text{if } |y_i| > \delta \end{cases} \quad (3.11)$$

with:

- N : Total number of samples;
 - y_i : True value for the i -th sample;
 - \hat{y}_i : Predicted value for the i -th sample;
 - δ : Threshold that determines the cutoff between MSE and L1 loss.
- In [WMSE](#), each bin class gets a weight, which is used to multiply the [MSE](#) of a given prediction. Thus, the weight is based on the expected value, not on the individual error of a given value prediction. In this project, the largest bin of irradiation values gets a weight of 0.25, and all others 1.0. This corresponds with the ratio of the size differences.

$$\text{Weighted MSE} = \frac{1}{N} \sum_{i=1}^N w_i (y_i - \hat{y}_i)^2 \quad (3.12)$$

- N : Total number of samples;
- y_i : True value for the i -th sample;
- \hat{y}_i : Predicted value for the i -th sample;
- w_i : Weight of value i based on the corresponding bin.

3.6. Interaction

Architects and designers typically do not possess a programming background, making it crucial to develop an intuitive approach for interacting with the AI model. Given that the dataset is derived from Honeybee simulations, the decision was made to implement the project in a manner that allows the code to be visually accessible through Grasshopper and McNeel Rhino. Additionally, this chapter will propose potential future implementations, considering broader contexts as outlined in the introduction of this thesis.

3.6.1. Preprocessing

Similar to the dataset development process, samples intended for evaluation by the prediction model must undergo preprocessing to be formatted correctly. Since many of the functions used for this preprocessing are based on Rhino Python packages, it was decided to implement these within Grasshopper itself. The Python code can be transformed into a Grasshopper plugin, thereby consolidating all the necessary preprocessing steps into a single tool.

To enhance the efficiency of preprocessing, several steps were delineated. First, the user must be prompted to specify the location where the building will be designed. By providing the coordinates of the specific location, the code should automatically download the corresponding city patch from the 3D BAG website and convert it into the correct format.

Secondly, a distinction should be made between context and design geometry. Context geometry refers to all objects in the scene that will not be modified by the designer. This distinction is particularly beneficial in an optimization process, as it allows the preprocessing phase for most of the geometry to be conducted only once. Conversely, the design geometry, which constitutes the building being designed or developed by the designer or computer, requires the preprocessing phase to be repeated for each design iteration.

Finally, both context and design geometry are combined into a preprocessing node that generates a point cloud, which will be used for prediction.

3.6.2. Live Prediction

In contrast to the preprocessing phase, the evaluation using the neural network must be executed through a specific Python 3 interpreter. Grasshopper, however, is limited to IronPython, which makes direct execution within it impossible⁵. Moreover, it is crucial that the designer software utilizing the code does not impose limitations on the network's usage.

Several techniques were explored for executing the neural network. Initially, it was found possible to call a command prompt through the `os.system()` and `subprocess.Popen()` functions to run the code from a Miniconda environment. This approach is similar to how Honeybee invokes Radiance/Accelerate programs. However, it resulted in significant overhead. First, booting the command window was relatively slow. Additionally, executing the code required loading all the necessary Python packages, which introduced several seconds of delay. While this delay might be acceptable for a single evaluation, it becomes a substantial bottleneck in large-scale optimization, reducing overall efficiency.

⁵In this project, Rhino 7 was used. Rhino 8 has new tools available in Grasshopper to use CPython packages.

3. Methods

A more efficient solution can be developed in the form of a server-client system (Figure: 3.24). In this setup, the server is initiated using the `subprocess.Popen()` method. The server preloads all the required Python packages in a Python 3 environment for running the neural network. An open connection is then established on the local machine using a socket system. Once the server is running, the client, which resides within the Grasshopper environment, gathers the point cloud, encodes it to bytes, and sends it to the server. After the server processes the point cloud using the neural network, the irradiation values are sent back to the client. This approach offers several advantages:

- Loading the Python packages only takes place once;
- The client can be easily implemented in different types of software;
- The client can send data to other machines, such as a Renderfarm or Supercomputer through a given port. This allows large numbers of prediction to be computed in a short timeframe.

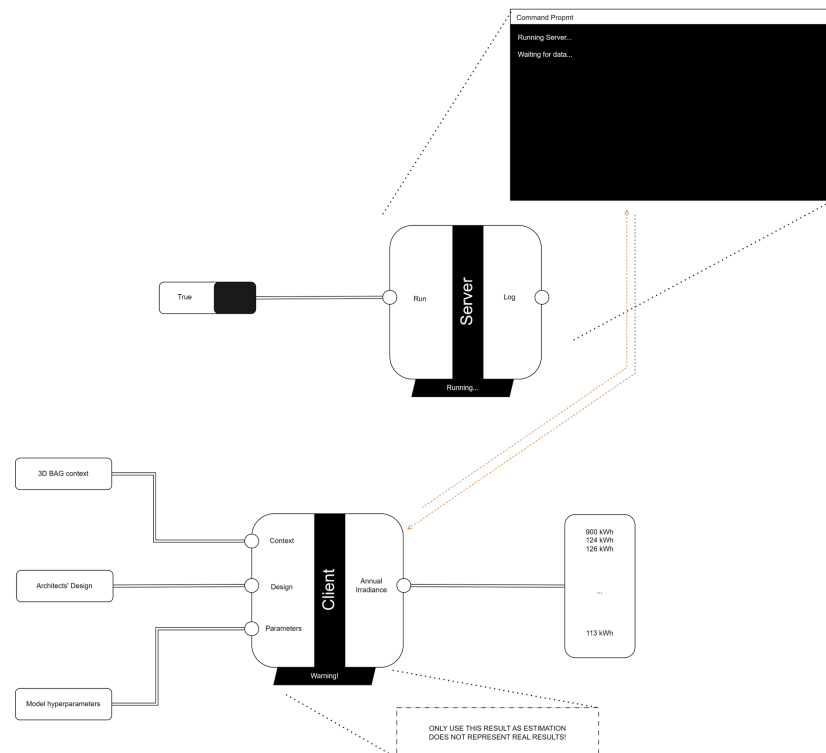


Figure 3.24.: The client-server system in Grasshopper, in which the server preloads all the packages. New data is sent from the client to the server. (Image by author)

3.6.3. Visualization

Based on the type of point cloud—either regular or random—the irradiation values can be visualized in Rhino using a mesh or colored point cloud, respectively. While generating a colored mesh was previously explored using the Ladybug package, it was found to be relatively slow. To address this, a new approach was implemented in Python, significantly enhancing the performance of visualization.

3.6.4. Optimization

By connecting the irradiation values from the client to a brute-force optimizer, such as Galapagos⁶, it is possible to optimize a design efficiently. To show the potential of this procedure, the following methodology was defined:

Design assignment: development of a tower in which the sun exposure on horizontal surfaces (ground & roofs) is maximized and sun exposure on vertical elements (facades) is minimized.

Fitness function⁷:

$$\text{Fitness} = \frac{\sum E_{i;hor}}{n_{hor} \cdot 1000} + \left(1 - \frac{\sum E_{i;ver}}{n_{ver} \cdot 1000}\right) \quad (3.13)$$

with:

- $\sum E_{i;hor}$ Annual irradiation on horizontal surfaces;
- n_{hor} : Number of sensors points on horizontal surfaces;
- $\sum E_{i;ver}$ Annual irradiation on vertical surfaces;
- n_{ver} : Number of sensors points on vertical surfaces.

Execution: The optimization process can be conducted in two ways: sequential design evaluation or batch design evaluation. In batch evaluation, point cloud sizes are padded to match the sample with the highest number of sensor points within the batch. This approach is particularly beneficial for devices with substantial VRAM, as it enables the simultaneous evaluation of multiple samples during the optimization process.

3.6.5. Conclusion

This chapter has detailed how architects and designers can interact with the AI model in a user-friendly environment, combined with an optimization procedure.

⁶<https://ieatbugsforbreakfast.wordpress.com/2011/03/04/epatps01/>

⁷This is a simplified fitness function. In reality, the area of mesh faces corresponding to the sensor points, should also be taken into account.

4. Analysis

In this chapter, conclusions will be drawn from the outcomes of the methods described in previous sections. The chapter is structured as follows: first, it presents the results related to dataset generation, simulation, and parallelization. Next, it provides an overview of the performance of PointNet, PointNet++, and various PointNeXt configurations. Finally, it addresses the findings from the interaction experiment previously discussed.

4.1. Dataset Generation, Simulation, and Parallelization

This section discusses the results obtained from the dataset generation, simulation, and parallelization methods. It includes a visual evaluation of the generated samples and an analysis of any errors that occurred during the process. Additionally, the computation time associated with the parallelization approach is summarized.

4.1.1. Dataset Sizes and Types

As part of this research, four distinct datasets were developed, each containing urban geometry, a corresponding point cloud description, and irradiation values. The properties of these datasets are detailed in Table 4.1. The first dataset comprises 18,380 samples with point clouds organized

Name	Patch Size	Point Sampling Strategy	Number of Samples
dset100_xl-regular	100m x 100m	Regular	18.380
dset100_xl-random	100m x 100m	Poisson Disk	21.601
dset300_s-regular	300m x 300m	Regular	2.206
dset300_s-random	300m x 300m	Poisson Disk	2.602

Table 4.1.: Four datasets developed with distinct properties.

in a regular grid. The second dataset contains slightly more samples, totaling 21,601, owing to the use of the Poisson Disk point sampling technique. Specifically, the preprocessing phase for regular point grids is more prone to errors, which necessitates the exclusion of some samples due to incorrect geometry.

While the neural network proposed in this research is trained on relatively large patches of 100m x 100m, it is also valuable to assess its performance on larger-scale samples of 300m x 300m. These larger samples are derived from the same volume of 3D [BAG](#) data as the previously mentioned dataset. Due to the increased area (a factor of 9 times larger), the number of samples in this dataset is approximately one-ninth of the smaller dataset. The prediction approach proposed in this thesis is less sensitive to memory consumption compared to previous work on 3D irradiation predictions by Han et al. (2022), allowing it to handle significantly larger patches.

4. Analysis

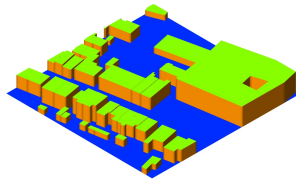
For both the 100m x 100m and 300m x 300m samples, point coordinates have been normalized to the $[0,1]$ or $[-1,1]$ domain before being input into the neural network.

4.1.2. Generation

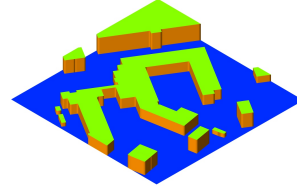
Despite extensive efforts to optimize the geometric preprocessing and simulation code, residual errors in the dataset remain evident. A comprehensive visual evaluation of 1,000 regular point cloud samples (see Figures 4.1 and 4.3) has identified several specific issues:

- **Leveling Buildings:** To simplify the prediction process, buildings were leveled by adjusting all vertex coordinates to ensure the lowest vertices were at $z = 0$. This adjustment unintentionally distorted roof height differences (Figure: 4.2c). A more precise approach would involve only translating the lowest vertices while preserving relative roof heights.
- **Boolean Splitting Accuracy:** The boolean splitting procedure employed during preprocessing has proven insufficiently accurate. The regular ground mesh is split, and the resulting elements are repositioned to form roofs. Figure 4.2a highlights missing triangular faces in the roofs. Future improvements could involve using more advanced splitting algorithms or implementing stricter area difference checks between expected and actual geometry.
- **Intersecting Walls:** Figure 4.2b indicates issues with the 3D BAG data, particularly concerning intersecting walls between buildings. This results in roofs being placed at incorrect heights. This issue could be mitigated by detecting and resolving building outline intersections before performing the ground mesh split.
- **Cut Off Corners:** Figure 4.2d shows that certain mesh split operations fail to accurately follow building outlines, leading to incorrectly cut-off corners. This issue may be resolved by employing more advanced splitting algorithms or using lower tolerance thresholds.
- **Point Cloud Reduction:** In some cases, the point cloud reduction algorithm, designed to remove dividing walls, failed to eliminate points at the ground level. This problem is likely due to rays not intersecting with nearby wall mesh faces because they intersect only with face edges. A potential solution could involve slightly moving these points upward before applying the reduction algorithm.

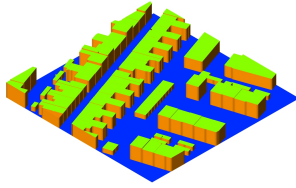
Given that only 1,000 samples were visually evaluated, the exact proportion of erroneous samples within the entire dataset cannot be precisely determined. However, empirical evidence suggests that fewer than 5% of the regular dataset samples contain errors, most of which are relatively minor.



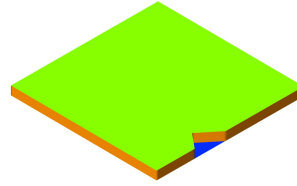
(a) Sample 57



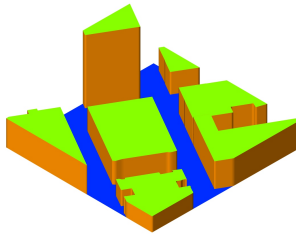
(b) Sample 113



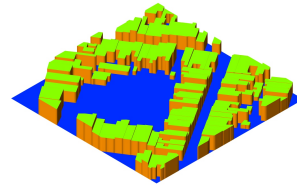
(c) Sample 139



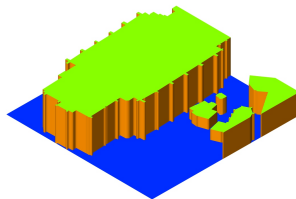
(d) Sample 190



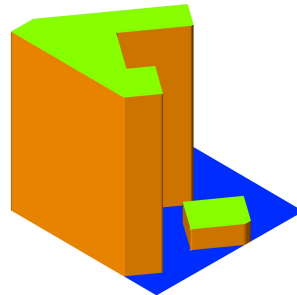
(e) Sample 294



(f) Sample 364



(g) Sample 779



(h) Sample 909

Figure 4.1.: A set of manually picked preprocessed dataset samples with distinct urban typologies. Blue colors indicate the ground context mesh, orange colors facades and green colors roofs.

4. Analysis

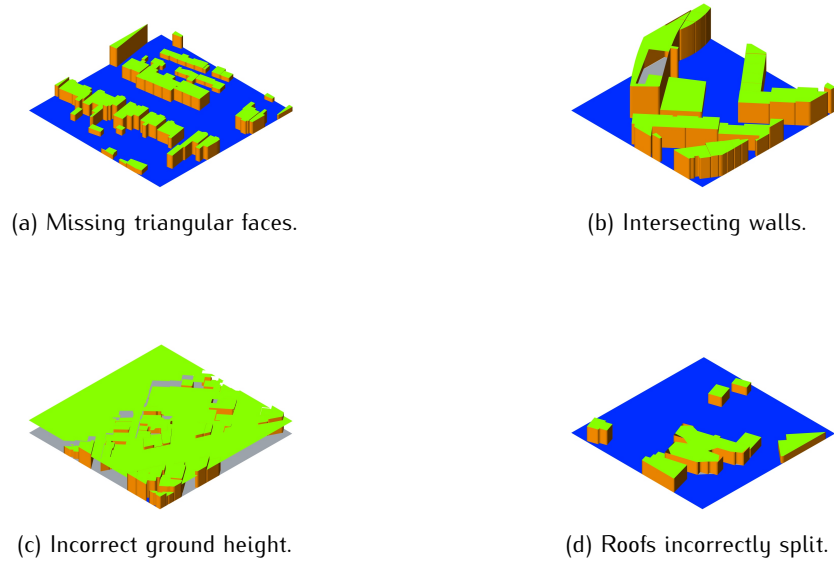


Figure 4.2.: A set of dataset samples with errors in the preprocessing procedure. Blue colors indicate the ground context mesh, orange colors facades and green colors roofs.

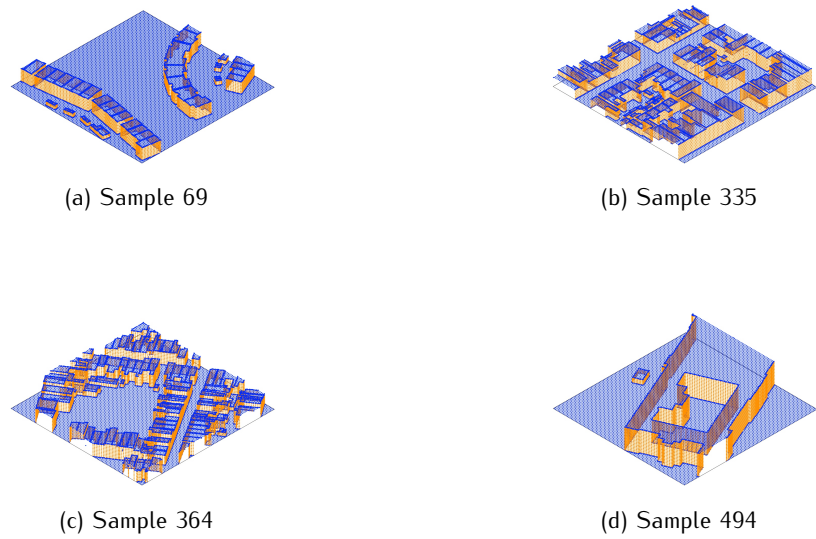


Figure 4.3.: A set of manually picked point clouds, based on the geometry in the dataset. Orange colored points relate to vertical surfaces. Blue colors indicate horizontal surfaces.

4.1.3. Simulation

To gain a more detailed understanding of the irradiation values within the datasets, a histogram was generated for the regular 100m x 100m dataset. This histogram combines the data from the training, validation, and testing sets, providing a comprehensive overview. The dataset encompasses a total of 324,913,022 sensor points, each representing a location where solar irradiation predictions will be made.

As illustrated in Figure 4.4a, the irradiation values are binned into intervals of 100 kWh/m². The distribution reveals that values within the 900–1000 kWh/m² range are significantly more prevalent compared to other ranges. This predominance is due to the full solar exposure received by the geometry at these locations, which naturally results in higher irradiation values.

In total, 151,704,158 sensor points fall within this 900–1000 kWh/m² bin, which constitutes 47.7% of the entire dataset. The skewed distribution towards higher irradiation values suggests that much of the geometry in the dataset receives substantial solar exposure, which could influence the performance of the prediction models and may require careful consideration in the training process to avoid potential biases.

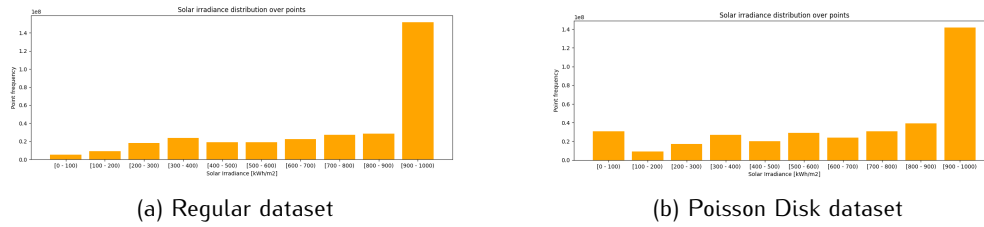


Figure 4.4.: Irradiation distribution for the regular and Poisson Disk 100m x 100m dataset.

Similarly, the random dataset consists of 369,907,058 sensor points, with 141,899,798 of these points falling within the 900–1000 kWh/m² range, accounting for 38% of the dataset (Figure 4.4b). The most notable difference between the two dataset distributions is the substantially larger 0–100 kWh/m² bin in the randomly (Poisson Disk) sampled dataset. This increase is primarily due to the presence of dividing walls in the geometric data, which have not been removed and therefore receive no irradiation.

Based on the dataset distributions, it can be concluded that there is an imbalance in the expected irradiation values. A solution to this problem will be described in section 4.2.5.

4. Analysis

4.1.4. Parallelization

As previously discussed, sequential synthesis of dataset samples was not a feasible option for this project. The optimization achieved through parallelization led to significant speed improvements, as shown in Table 4.2. The results indicate that Poisson Disc sampling slightly outperforms regular point sampling in terms of efficiency. However, it is important to note that regularized meshes were generated in both sampling methods. Further research could potentially reduce overall computation time by applying the Poisson Disc sampling method to simpler, rough geometry.

Dataset	Point Sampling	Processes used)	(cores	Samples	Computation time (s)
100m x 100m	regular	1		100	3.965
100m x 100m	regular	18		100	608
100m x 100m	Poisson Disc	18		100	591
300m x 300m	Regular	12		100	4.401
300m x 300m	Poisson Disc	12		100	4.225

Table 4.2.: Efficiency of generation and simulation methods.

As shown in Figure 4.5, performance scores for each process are recorded in individual log files. If a sample's Ground Space Index (GSI) value is too low, the sample generation is skipped. However, if the GSI value meets the threshold, the generation and simulation processes continue. These log files provide a rough estimate of the timings associated with different preprocessing procedures. Notice that the AcceleRad simulation accounts for the greatest computation time.

```
2024-06-22 20:09:47 INFO: Started computing patch[13].
2024-06-22 20:09:47 INFO: GSI_score 0.0 of sample 13 not high enough to continue generating sample.
2024-06-22 20:09:47 INFO: Finished computing patch[13] in 0.23s.

2024-06-22 20:09:47 INFO: Started computing patch[14].
2024-06-22 20:09:47 INFO: Started preprocessing mesh for patch[14] with GSI value of 0.43
2024-06-22 20:09:56 INFO: Computing sensors for mesh patch[14]
2024-06-22 20:10:01 INFO: Generating model for mesh patch[14] augmentation 0
2024-06-22 20:10:02 INFO: Simulating irradiation model for mesh patch[14] augmentation 0
2024-06-22 20:10:52 INFO: Saving mesh patch[14] and generating visualization
2024-06-22 20:10:53 INFO: Finished preprocessing mesh for patch[14] in 65.57s
2024-06-22 20:10:53 INFO: Finished computing patch[14] in 66.05s.
```

Figure 4.5.: Log with timings for the generation and simulation of a 100m x 100m Poisson Disc sample in a specific parallelization child process.

4.2. Prediction

In this section, the performance of irradiation prediction using PointNet-based neural networks is evaluated. Initially, the networks provided by Qian et al. (2022) were tested on the irradiation datasets described earlier, using the hyperparameters recommended by the original authors. Subsequently, one of these models was selected for sequential hyperparameter tuning. Finally, this section discusses performance in relation to dataset imbalance, inference, and alternative dataset configurations.

4.2.1. Baseline Evaluation

The 100m x 100m regular dataset was evaluated using six models as recommended by Qian et al. (2022): PointNet, PointNet++, PointNeXt-S, PointNeXt-B, PointNeXt-L, and PointNeXt-XL. Figure 4.6 presents the results, with the number of training steps on the x-axis (corresponding to 25 epochs) and the irradiation RMSE on the y-axis. Similarly, Figure 4.7 shows the validation loss for the baseline models.

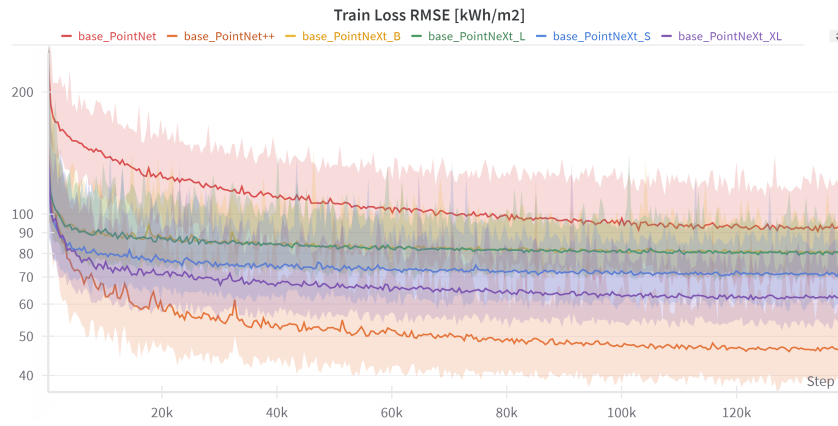


Figure 4.6.: Irradiation training loss expressed in kWh/m² over 25 epochs.

As shown in the results, the PointNeXt-S model performs best on the validation dataset, achieving a validation RMSE of 47.58 kWh/m², followed by PointNeXt-L with a validation RMSE of 52.70 kWh/m². The modifications introduced in the PointNeXt architecture, as evidenced by the low training loss (train RMSE 44.91 kWh/m²) and higher validation loss (val. RMSE 68.09 kWh/m²), demonstrate a performance improvement over the original PointNet++ architecture. The observation that validation loss is lower than training loss for all networks except PointNet++ may be attributed to the dropout layer at the end of the network architecture, which is active only during training.

After training, the models were validated using the 10% test dataset split to confirm the performance suggested by the validation loss. Figure 4.8 presents results from the test phase, which are consistent with the validation outcomes, with the PointNeXt-S architecture achieving the lowest RMSE of 46.23 kWh/m². To assess the general performance of the model, outlier RMSE-per-sample values were plotted in Figure 4.9, showing the highest and lowest RMSE values from the test dataset.

4. Analysis

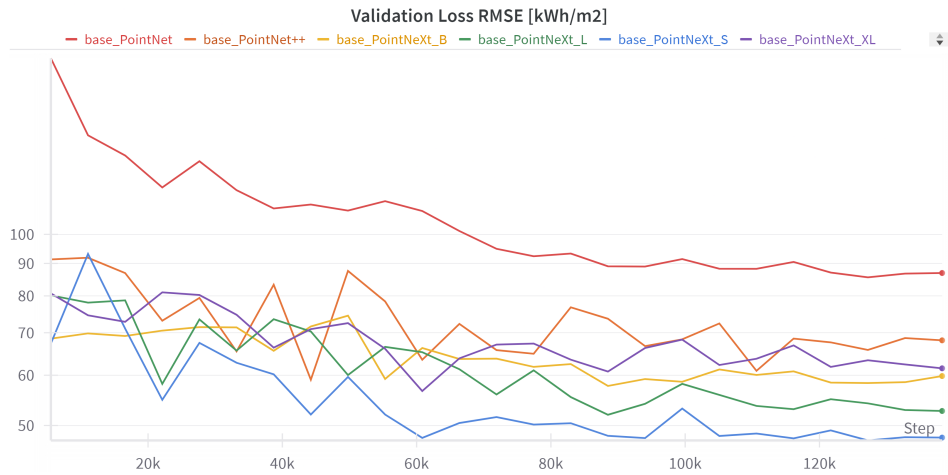


Figure 4.7.: Irradiation validation loss expressed in kWh/m² over 25 epochs.

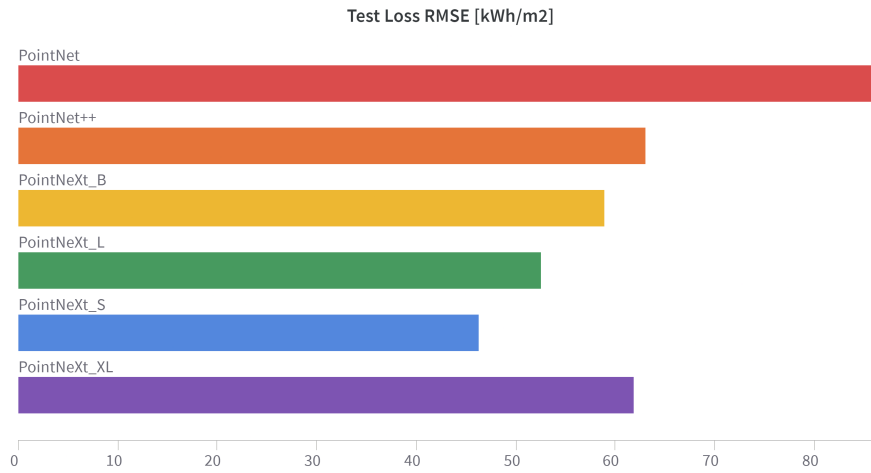


Figure 4.8.: Test loss expressed in kWh/m² for 10% test split.

The overall performance distribution of the network was assessed using both macro-averaged and micro-averaged F1-scores based on the test dataset (figure: 4.10). To compute these F1-scores, the irradiation values were categorized into 100 kWh/m² bins. The results suggest that the PointNeXt-S model provides the best overall coverage across all irradiation values.

In summary, the baseline evaluation identified PointNeXt-S as the top performer, with the highest scores and the lowest average test RMSE. However, initial experiments with alternative hyper-parameter configurations indicated that PointNeXt-S struggles with scalability due to its limited number of layers. As a result, **PointNeXt-L was chosen as the benchmark for further hyper-parameter optimization**. PointNeXt-L achieved an average test RMSE of 52.46 kWh/m², with a minimum RMSE of 19.98 kWh/m² and a maximum RMSE of 104.39 kWh/m².

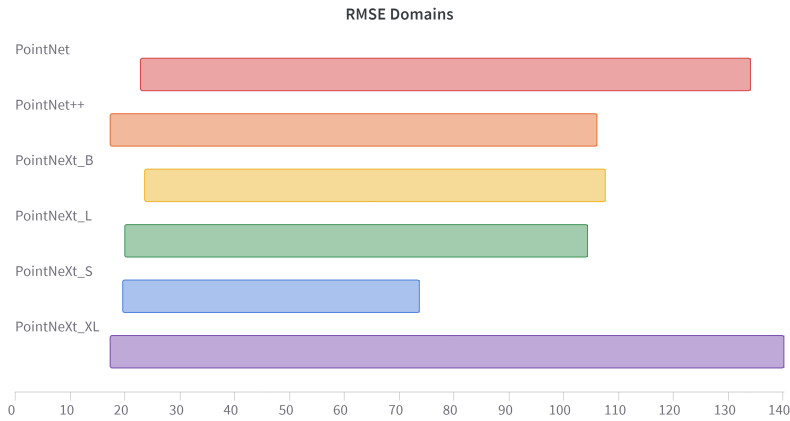


Figure 4.9.: Test loss domain from minimum to maximum expressed in kWh/m² for 10% test split.

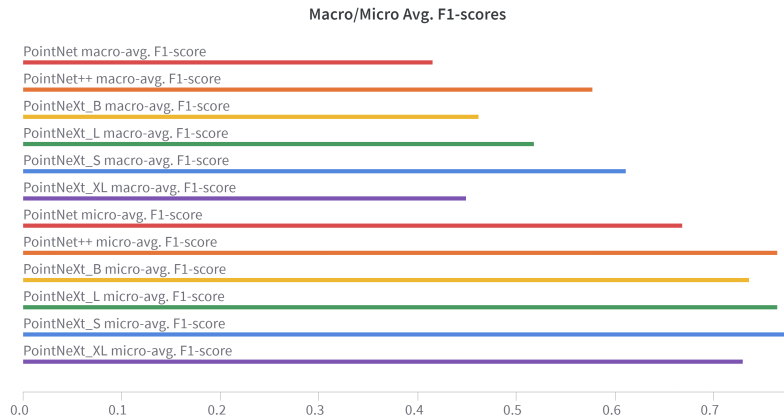


Figure 4.10.: Macro and micro averaged F1-scores based on the test dataset.

4.2.2. Hyperparameter Tuning

In contrast to traditional hyperparameter tuning methods such as grid search and random search, the optimization of PointNeXt for irradiation prediction was conducted through manual adjustment of hyperparameters. This decision was driven by the limitations of available hardware, which made training networks using automated hyperparameter tuning methods impractical. As a result, the network configuration and dataset hyperparameters were adjusted based on empirical observations and intuition.

The overall results of this optimization process are summarized in table 4.3 and 4.4. Each row in the table represents a different model architecture configuration. Modifications included changes to the loss function, dataset characteristics, model-specific hyperparameters, normalization techniques, and the number of training epochs. The final section of the table presents the results from training sessions where the most effective hyperparameters were combined, offering insights into the performance improvements achieved through this manual tuning process.

Hyperparameter Tuning												
Type	Model	Epochs	Hyper-parameters ^a	Dataset	Normalization norm_min, irr	Loss Function	Test RMSE ^b	Train RMSE ^{b,c}	Highest RMSE ^b	Lowest RMSE ^b	Macro avg. F1 score	Micro avg. F1 score
base	PointNet	25	None	100 regular	[-1,1], [-1,1]	MSE	86.87	93.49	134.19	22.94	0.41	0.67
base	PointNet++	25	None	100 regular	[-1,1], [-1,1]	MSE	62.99	45.42	106.18	17.32	0.58	0.76
base	PointNeXt-S	25	None	100 regular	[-1,1], [-1,1]	MSE	46.23	71.33	73.69	19.58	0.61	0.78
base	PointNeXt-B	25	None	100 regular	[-1,1], [-1,1]	MSE	58.92	80.90	107.68	23.64	0.47	0.73
base	PointNeXt-L	25	None	100 regular	[-1,1], [-1,1]	MSE	52.46	80.50	104.39	19.98	0.52	0.76
base	PointNeXt-XL	25	None	100 regular	[-1,1], [-1,1]	MSE	61.89	62.62	140.19	17.29	0.45	0.73
loss	PointNeXt-L	25	None	100 regular	[-1,1], [-1,1]	Delta	58.77	100.48	114.11	18.57	0.44	0.73
loss	PointNeXt-L	25	None	100 regular	[-1,1], [-1,1]	Reduction	54.28	82.10	110.34	22.05	0.49	0.75
loss	PointNeXt-L	25	None	100 regular	[-1,1], [-1,1]	WMSE	53.87	86.90	90.86	28.46	0.53	0.73
dset	PointNeXt-L	25	None	100 random ^d	[-1,1], [-1,1]	MSE	55.03	85.67	175.17	16.43	0.60	0.75
dset	PointNeXt-L	25	None	300 regular	[-1,1], [-1,1]	MSE	76.70	89.18	177.04	23.06	0.36	0.69
dset	PointNeXt-L	25	None	300 random ^d	[-1,1], [-1,1]	MSE	83.22	90.62	176.17	29.18	0.38	0.67
hyperparam	PointNeXt-L	25	expansion - 8	100 regular	[-1,1], [-1,1]	MSE	56.49	78.12	141.46	11.95	0.5	0.74
hyperparam	PointNeXt-L	25	vox_max - 24000	100 regular	[-1,1], [-1,1]	MSE	54.67	79.29	127.92	9.75	0.5	0.76
hyperparam	PointNeXt-L	25	act - leakyReLU	100 regular	[-1,1], [-1,1]	MSE	55.86	82.11	112.88	21.42	0.48	0.75
hyperparam	PointNeXt-L	25	nsample - 128	100 regular	[-1,1], [-1,1]	MSE	37.4	81.32	83.58	17.04	0.58	0.81
hyperparam	PointNeXt-L	25	nsample - 256	100 regular	[-1,1], [-1,1]	MSE	35.85	81.12	123.66	10.22	0.62	0.82
hyperparam	PointNeXt-L	25	nsample - 128	100 regular	[-1,1], [-1,1]	WMSE	32.14	87.53	94.46	11.56	0.74	0.85
hyperparam	PointNeXt-L	25	radius - 0.05	100 regular	[-1,1], [-1,1]	MSE	41.70	77.08	163.73	13.62	0.66	0.83
hyperparam	PointNeXt-L	25	radius - 0.025	100 regular	[0,1], [-1,1]	MSE	32.51	78.41	77.2	15.36	0.63	0.83
hyperparam	PointNeXt-L	25	radius - 0.025	100 regular	[-1,1], [-1,1]	MSE	33.77	78.33	120.85	15.64	0.64	0.83
hyperparam	PointNeXt-L	25	residual - True	100 regular	[-1,1], [-1,1]	MSE	46.34	75.07	72.25	19.09	0.58	0.78
hyperparam	PointNeXt-L	25	stride - [1,2,2,2,2]	100 regular	[-1,1], [-1,1]	MSE	50.23	83.36	90.96	22.02	0.51	0.77
hyperparam	PointNeXt-L	25	stride - [1,3,3,3,3]	100 regular	[-1,1], [-1,1]	MSE	71.39	81.93	133.05	21.19	0.39	0.7
hyperparam	PointNeXt-XL	25	nsample - 128	100 regular	[-1,1], [-1,1]	MSE	30.07	59.57	145.11	77.33	0.73	0.86
normalization	PointNeXt-L	25	None	100 regular	[0,1], [-1,1]	MSE	104.46	92.32	156.28	35.52	0.36	0.66
normalization	PointNeXt-L	25	None	100 regular	[0,1], [0,1]	MSE	93.05	101.57	138.34	33.72	0.39	0.68
normalization	PointNeXt-L	25	None	100 regular	[-1,1], [0,1]	MSE	57.7	80.93	106.39	19.27	0.46	0.75

^a None hyperparameters are *expansion* 4 *vox_max* 10000 *act* ReLU *nsample* 32 *radius* 0.1 *residual* False *stride* [1,4,4,4,4], ^b Unit: kWh/m², ^c Averaged through time weighted EMA (smoothing factor 0.99), ^d Poisson Disc point sampling

Table 4.3.: Hyperparameter tuning part A

Hyperparameter Tuning												
Type	Model	Epochs	Hyper-parameters ^a	Dataset	Normalization norm_min, irr	Loss Function	Test RMSE ^b	Train RMSE ^{b;c}	Highest RMSE ^b	Lowest RMSE ^b	Macro avg. F1 score	Micro avg. F1 score
epochs	PointNeXt-S	100	None	100 regular	[-1,1], [-1,1]	MSE	42.65	72.12	90.22	19.57	0.63	0.8
epochs	PointNeXt-L	100	None	100 regular	[-1,1], [-1,1]	MSE	56.9	77.91	107.51	21.22	0.47	0.74
super	PointNeXt-S	25	nsample - 128 radius 0.025 stride [1,2,2,2,2] residual <i>True</i>	100 regular	[0,1], [-1,1]	MSE	39.44	73.95	155.61	18.8	0.61	0.79
super	PointNeXt-L	25	nsample - 128 radius 0.025 residual <i>True</i>	100 regular	[-1,1], [-1,1]	MSE	32.42	72.54	82.12	15.8	0.69	0.84
super	PointNeXt-XL	25	nsample - 128 radius 0.025 residual <i>True</i>	100 regular	[-1,1], [-1,1]	MSE	24.04	56.52	76.82	10.36	0.79	0.88
super	PointNeXt-L	25	nsample - 128 radius 0.025 residual <i>True</i>	100 random ^d	[-1,1], [-1,1]	MSE	37.27	64.52	78.21	11.51	0.71	0.84
super	PointNeXt-XL	25	nsample - 256 radius 0.05 ^e residual <i>True</i>	100 regular	[-1,1], [-1,1]	MSE	24.85	58.18	136.48	8.8	0.79	0.88
super	PointNeXt-L	25	nsample - 128 radius 0.025 residual <i>True</i>	100 regular	[-1,1], [-1,1]	aWMSE	37.42	59.57	137.06	22.44	0.75	0.83
hyper	PointNeXt-XL	100	nsample - 128 radius 0.025 residual <i>True</i>	100 regular	[-1,1], [-1,1]	MSE	29.42	56.43	158.38	9.81	0.77	0.87
hyper	PointNeXt-XL	100	nsample - 128 radius 0.025 residual <i>True</i>	100 random ^d	[-1,1], [-1,1]	MSE	21.10	58.65	152.19	9.63	0.83	0.90
hyper	PointNeXt-XL	100	nsample - 128 radius 0.025 residual <i>True</i>	300 regular	[-1,1], [-1,1]	MSE	36.71	62.52	127.19	14.22	0.62	0.81
hyper	PointNeXt-XL	100	sample - 128 radius 0.025 residual <i>True</i>	300 random ^d	[-1,1], [-1,1]	MSE	48.53	66.68	127.60	22.28	0.58	0.76

^a None hyperparameters are *expansion 4 vox_max 10000 act ReLU nsample 32 radius 0.1 residual False stride [1,4,4,4,4]* ^b Unit: kWh/m², ^c Averaged through time weighted EMA (smoothing factor 0.99), ^d Poisson Disc point sampling, ^e Increased due to higher nsample

Table 4.4.: Hyperparameter tuning part B

4. Analysis

Normalization

The results presented in table 4.3 and 4.4 indicate that normalization plays a crucial role in the overall performance of the model. Initially, it was hypothesized that normalizing both point coordinates and irradiation output values to the $[0, 1]$ domain would yield the best network performance. However, the findings reveal that normalizing these values to the $[-1, 1]$ domain actually resulted in superior performance, as demonstrated in the baseline models (figure 4.11). This adjustment in the normalization approach proved to be a key factor in optimizing the network's effectiveness.

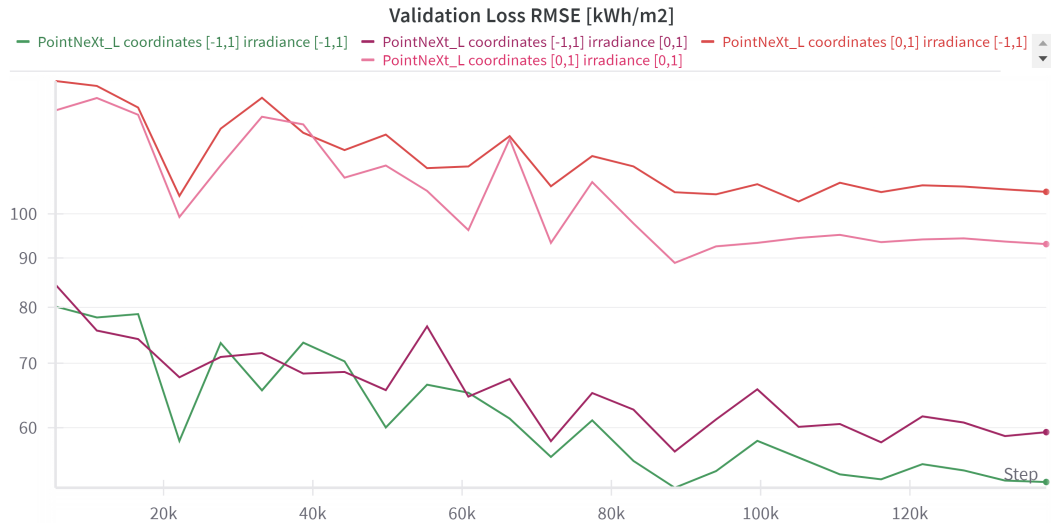


Figure 4.11.: Validation loss over 25 training epochs in kWh/m^2 , considering the normalization of input point coordinates and the output irradiation values.

Radius and Nsample

The adjustments to the radius and nsample hyperparameters have proven to be the most impactful in the optimization process (figure 4.12). Increasing the nsample from 32 to 128 resulted in a reduction in average test [RMSE](#) from 52.46 kWh/m^2 to 37.40 kWh/m^2 , with only minor changes observed in the lowest and highest test [RMSE](#) values. Additionally, reducing the radius from 0.1 to 0.05 achieved an average test [RMSE](#) of 33.77 kWh/m^2 .

Further decreases in the radius did not lead to significant performance improvements. This is likely due to the reduced global receptive field of the network. Similarly, increasing the 'nsample' from 128 to 256 did not enhance performance, possibly because the local context within the given radius contained a limited number of points.

While modifying the nsample and radius hyperparameters affects performance, these changes do not alter the network's size or depth. However, increasing the nsample significantly extends training time due to the larger local aggregation required in the grouping layer.

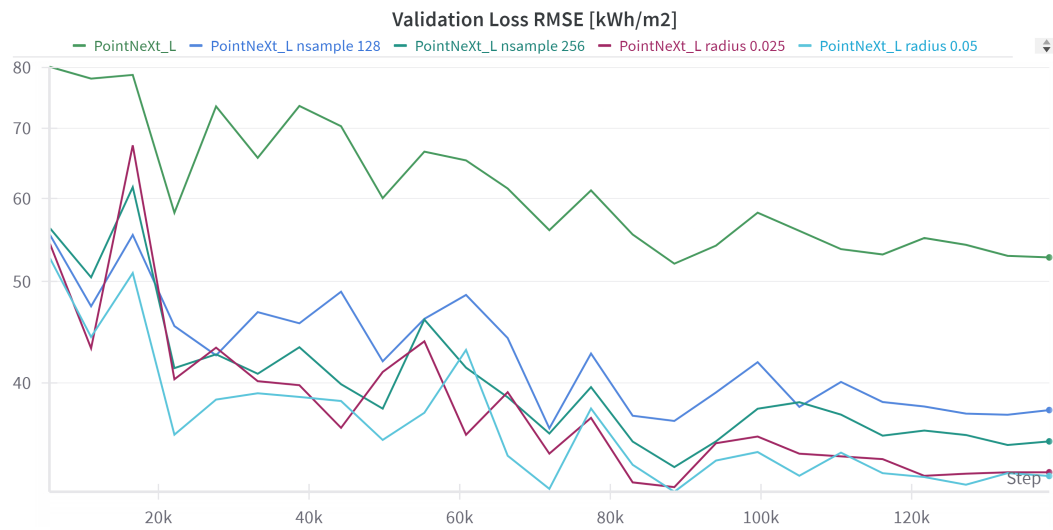


Figure 4.12.: Validation loss over 25 training epochs in kWh/m², considering radius and nsample hyperparameters.

Global Residual Layers

The addition of residual layers between encoder and decoder (which were not recommended by the original author) improved the performance to an average test RMSE of 46.24 kWh/m². Figure 4.13 shows the validation loss during training.

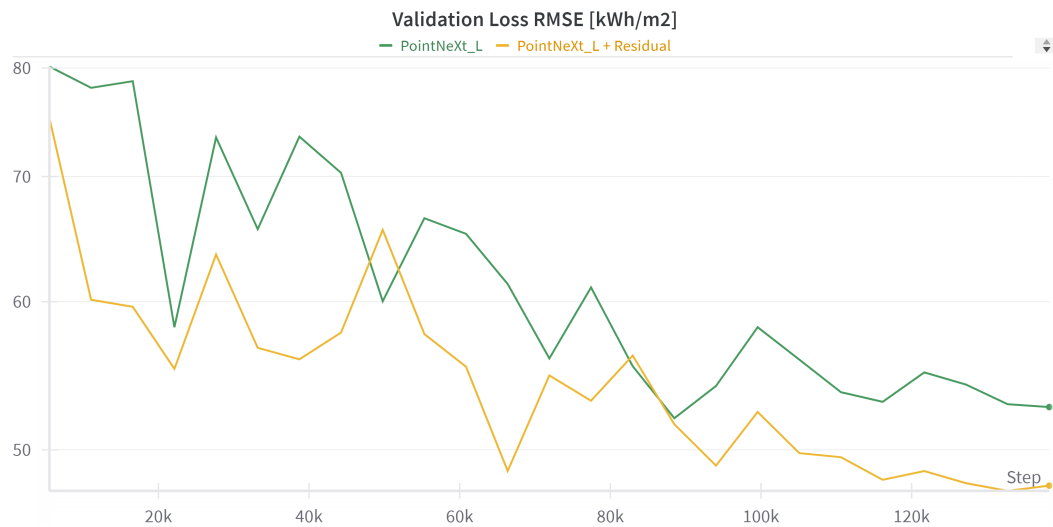


Figure 4.13.: Validation loss over 25 training epochs in kWh/m², considering residual layers between encoder and decoder blocks.

4. Analysis

Stride

Given that the number of points in the dataset is significantly smaller than those used to tune PointNeXt, it was considered beneficial to lower the strides to increase the number of features in the latent space between the encoder and decoder. Reducing the strides from $[1, 4, 4, 4, 4]$ to $[1, 2, 2, 2, 2]$ improved the average RMSE to 46.34 kWh/m².

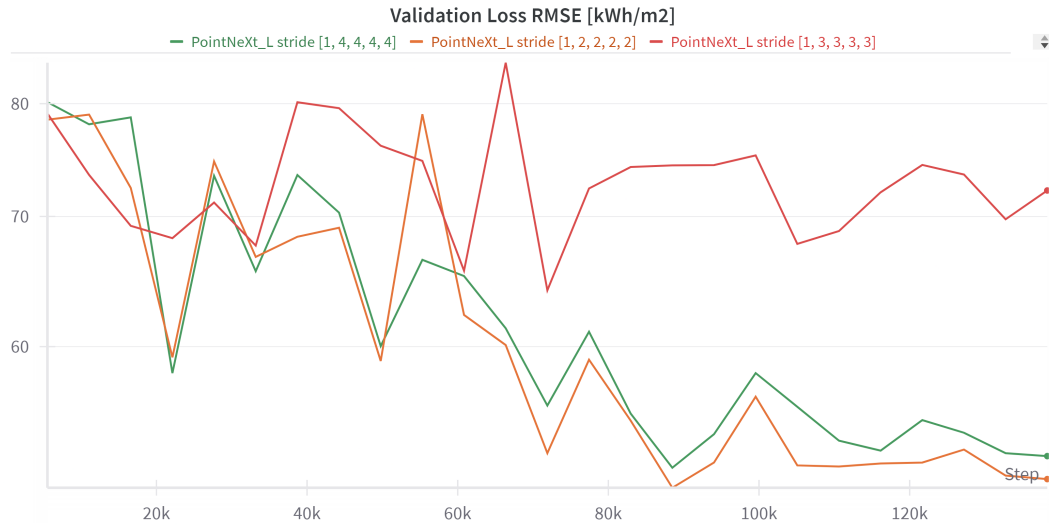


Figure 4.14.: Validation loss over 25 training epochs in kWh/m², considering the strides. Each successive stride in the list, corresponds to a deeper encoder block.

Epochs

Qian et al. (2022) recommend training PointNeXt for 100 epochs. However, training all configurations for 100 epochs would be inefficient for hyperparameter optimization. Therefore, the model configurations in this research were all trained for 25 epochs. Figure 4.15 illustrates the differences between training a network for 25 epochs versus 100 epochs. The default PointNeXt-L configuration appears to perform optimally with only 25 epochs. Nonetheless, when higher nsample values are used, extending the training period to 100 epochs positively impacts both validation and test loss.

Training for more epochs does not always result in better performance, as is illustrated in table 4.3 and 4.4. The PointNeXt-XL network trained on regular samples for 100 epochs shows a higher RMSE than the network trained on 25 epochs. However, this may also be caused by the fact that the testing loop was performed after the training loop finished, instead of using the epoch with the lowest validation RMSE. Figure 4.16 shows that the validation loss of epoch 84 is lower (22.12 kWh/m²) versus the validation loss after 25 epochs.

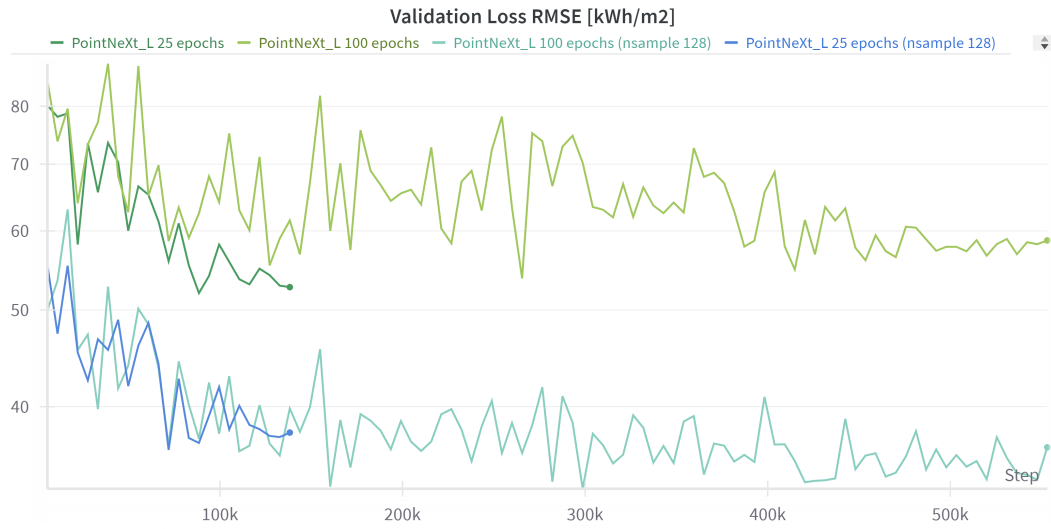


Figure 4.15.: Validation loss over 25 and 100 training epochs in kWh/m². Bottom two graphs have an nsample value of 128 and the top two graphs an nsample of 32.

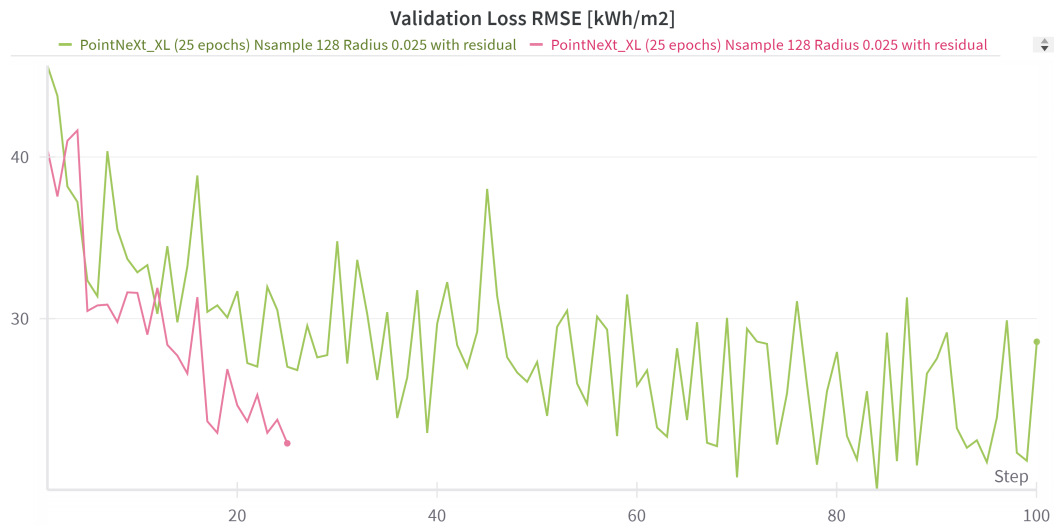


Figure 4.16.: Validation loss over 25 and 100 training epochs in kWh/m² for configuration with optimal hyperparameters.

4. Analysis

Model Scaling

The influence of using a deeper network, PointNeXt-XL with 41.6M parameters compared to PointNeXt-L with 7.1M parameters, is shown in figure X. Employing the deeper PointNeXt-XL model appears to significantly enhance performance, especially when combined with higher hyperparameter settings such as 'nsample' and 'radius'.

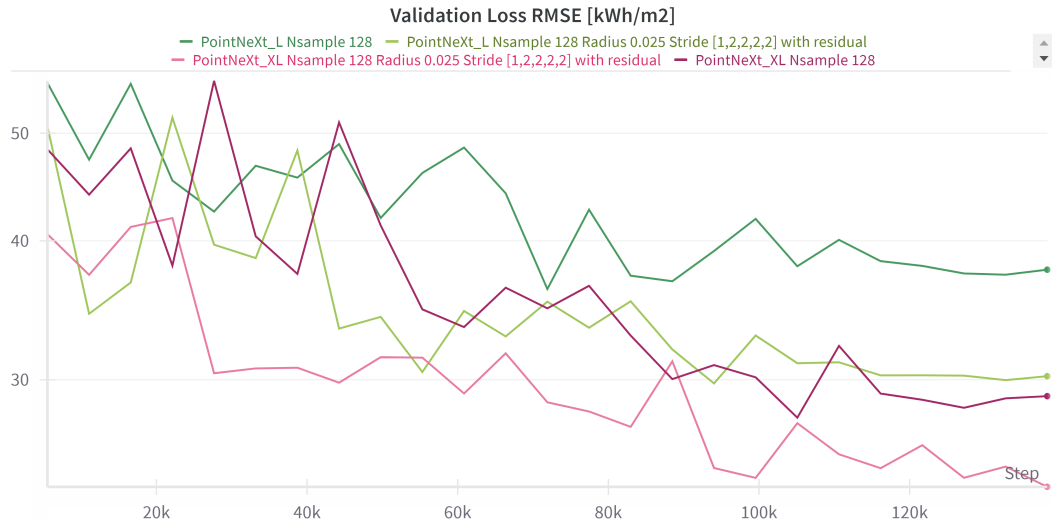


Figure 4.17.: Validation loss over 25 training epochs in kWh/m² for the deeper PointNeXt-XL vs default PointNeXt-L

Other Hyperparameters

Other hyperparameters, as detailed in table 4.3 and 4.4, did not have a significant effect on the performance of the larger PointNeXt model for irradiation prediction (figure: 4.18).

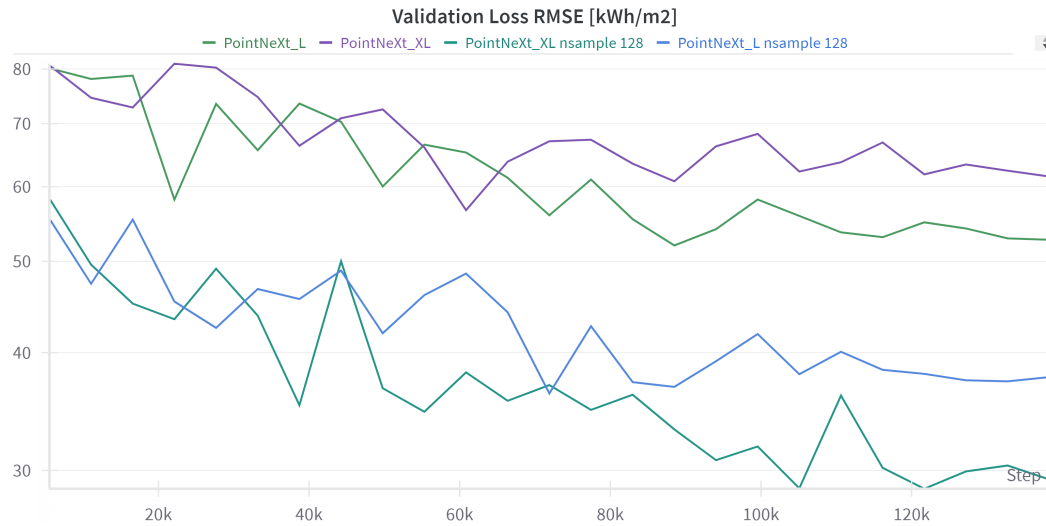


Figure 4.18.: Validation loss over 25 training epochs in kWh/m² for hyperparameters expansion, LeakyReLU (activation function) and maxvox (number of points sampled as input for the forward pass).

4.2.3. Average Performance Improvements

Table 4.5 outlines the enhancements achieved through tuning the most influential hyperparameters. It is evident that increasing the depth of the network does not result in a linear reduction in test loss, and is influenced by the settings of other hyperparameters.

Configuration	Delta	Avg. Test Loss
PointNeXt L default	100%	52.46 kWh/m ²
Nsample 32 → 128	+29%	37.40 kWh/m ²
Radius 0.1 → 0.025	+36%	33.37 kWh/m ²
Stride [[1,4,4,4,4] → [1,2,2,2,2]]	+4%	50.23 kWh/m ²
Residuals False → True	+12%	46.34 kWh/m ²
PointNeXt XL	-18%	61.89 kWh/m ²
PointNeXt L + all above	+42%	30.42 kWh/m ²
PointNeXt XL + all above	+54%	24.04 kWh/m ²

Table 4.5.: Table with influence of different model configurations on average test loss.

4. Analysis

4.2.4. Visual Evaluation

The following figures present a visual evaluation of the performance of the tuned PointNeXt-L model on the regular 100m x 100m dataset. Figures 4.19, 4.20, and 4.21 display five samples with the highest, median, and lowest average [RMSE](#), respectively.¹

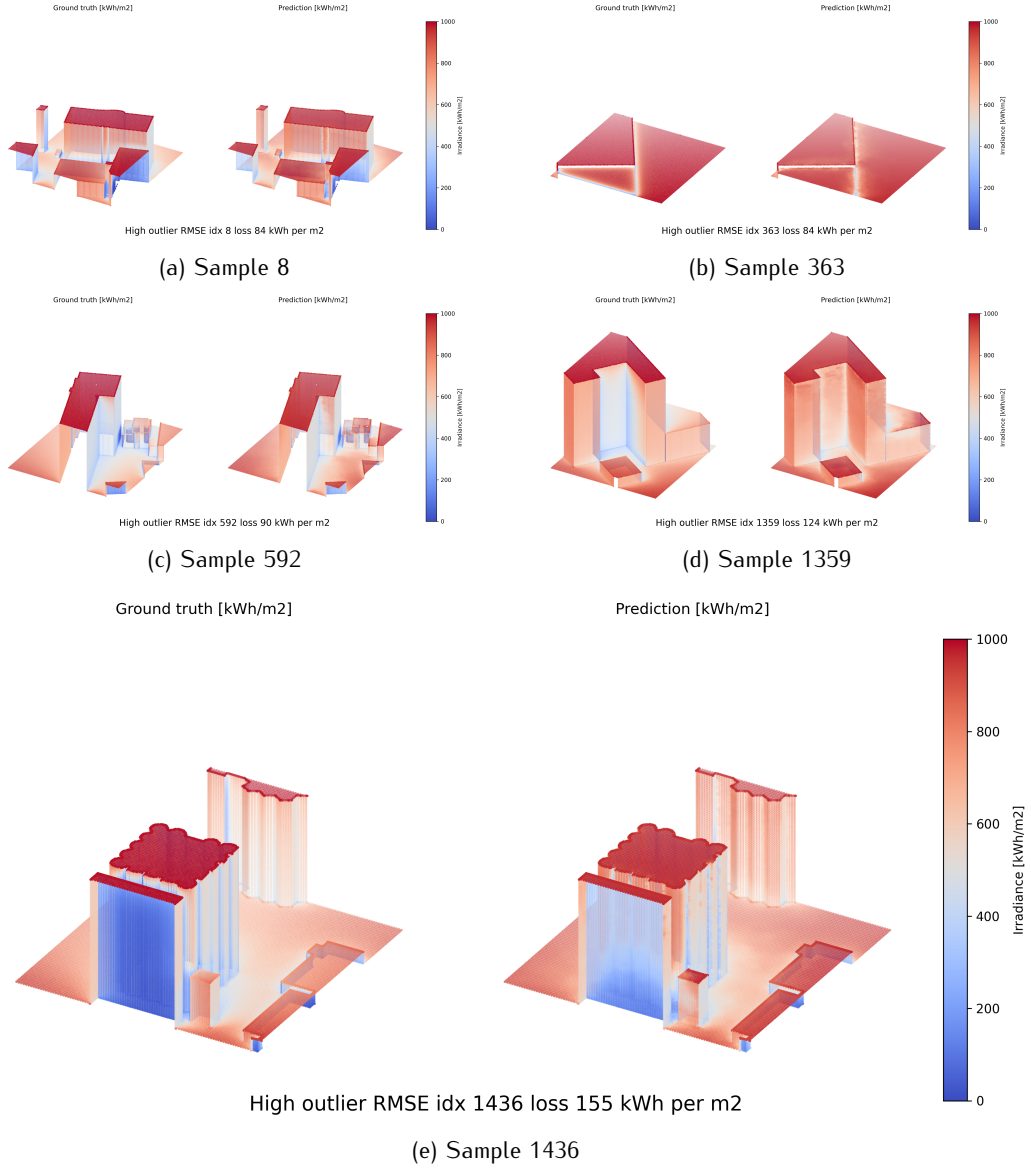


Figure 4.19.: Highest irradiation [RMSEs](#) for test dataset prediction, based on the large PointNeXt model with tuned hyperparameters Nsample 128, radius 0.025, stride [1,2,2,2,2], encoder-decoder residual True.

¹Note that due to a different seed, the errors shown may vary slightly from those in table 4.3 and 4.4.

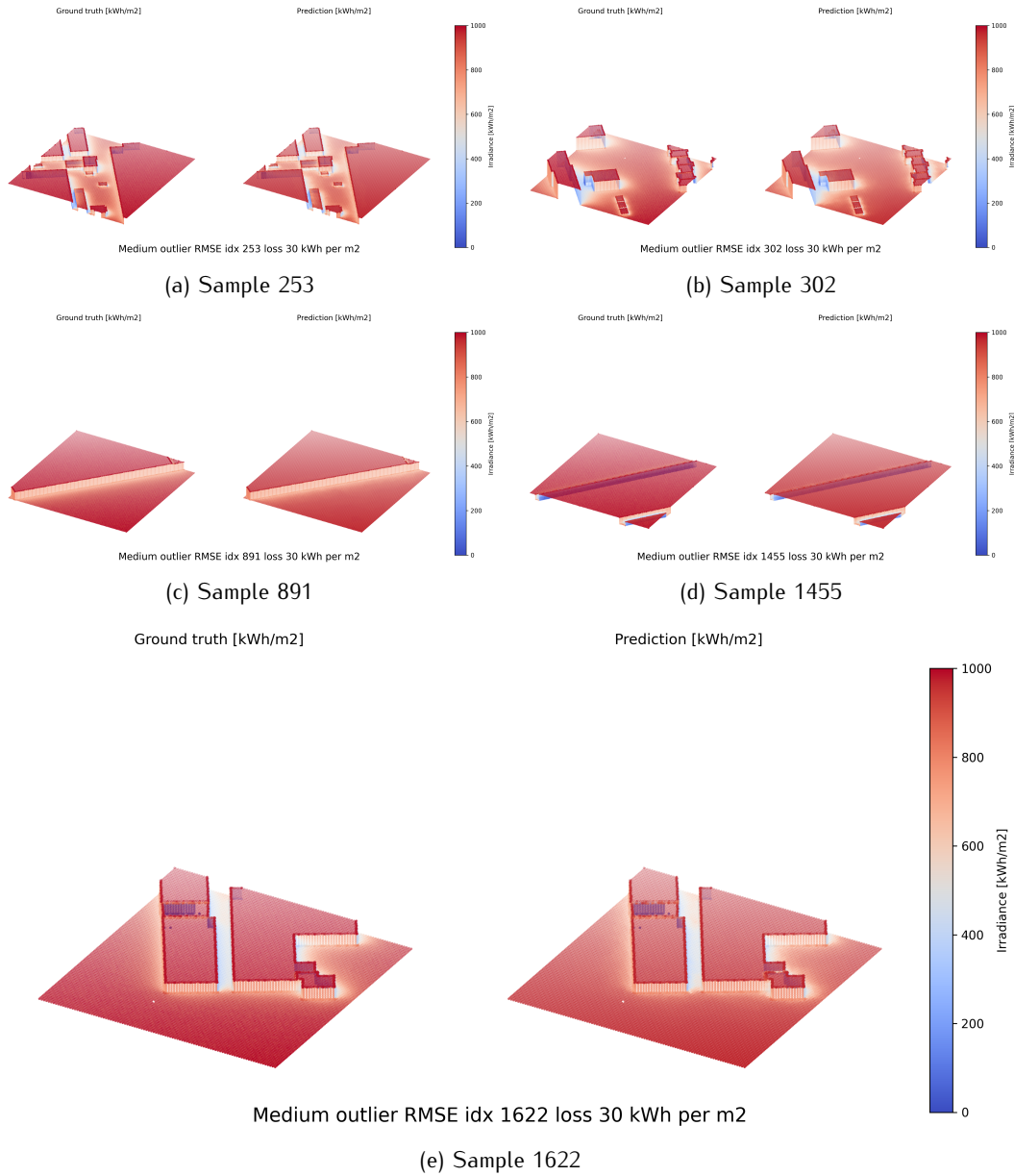


Figure 4.20.: Medium irradiation RMSE's for test dataset prediction, based on the large Point-NeXt model with tuned hyperparameters Nsample 128, radius 0.025, stride [1,2,2,2,2], encoder-decoder residual True.

Based on visual evaluations, the following conclusions about RMSE scores can be drawn:

- **High RMSEs:** These typically occur in building geometries that are significantly higher than average. It is anticipated that these samples are relatively rare in the training dataset, leading to increased errors for such unique geometries.
- **Medium RMSEs:** These correspond to typical building geometries present in the datasets.

4. Analysis

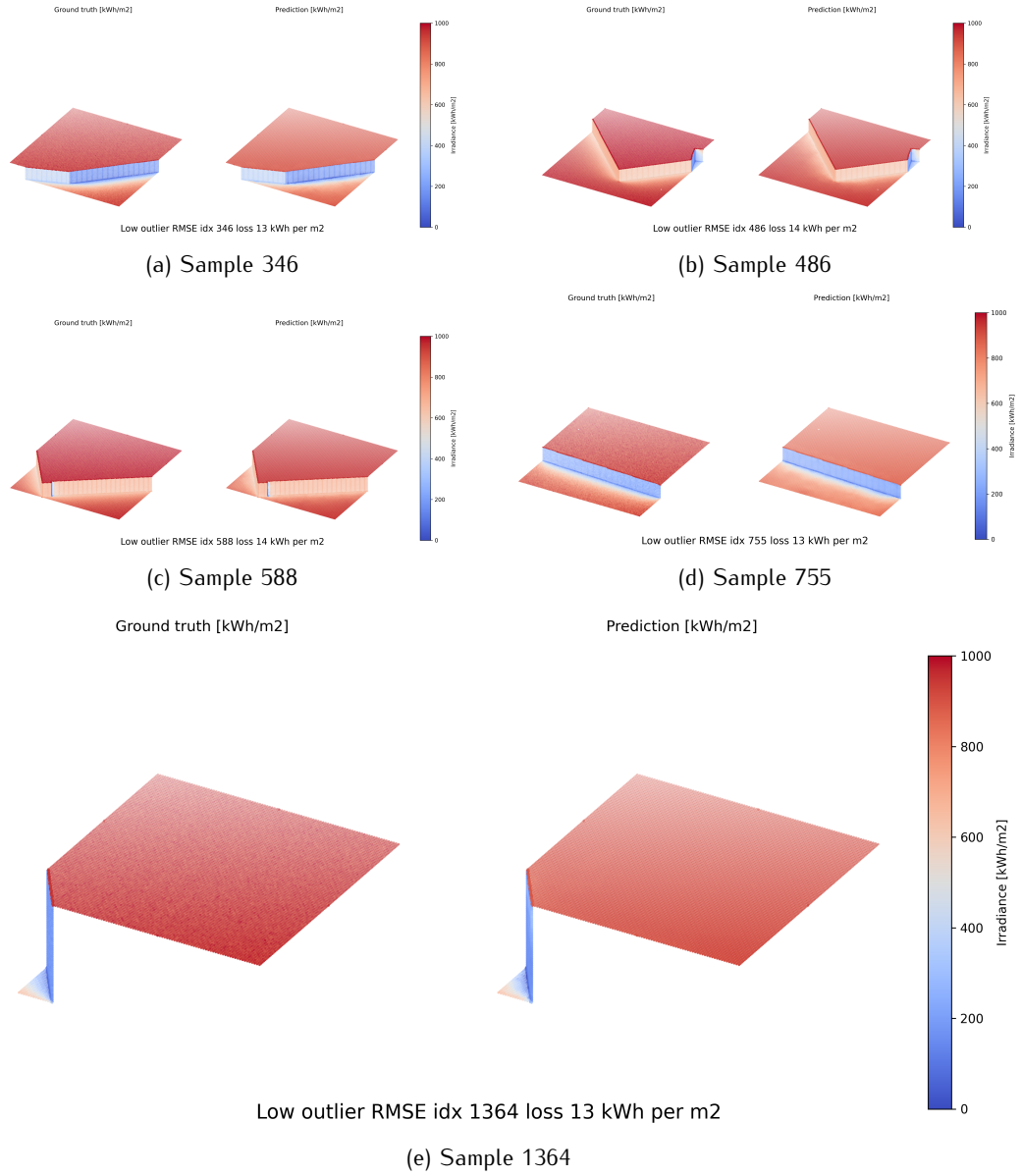


Figure 4.21.: Lowest irradiation **RMSEs** for test dataset prediction, based on the large PointNeXt model with tuned hyperparameters Nsample 128, radius 0.025, stride [1,2,2,2,2], encoder-decoder residual True.

Visually, these samples exhibit minimal apparent errors.

- **Lowest **RMSEs**:** These are usually associated with very simple geometric shapes. Most of these samples have irradiation values within the 900-1000 kWh/m² range and generally feature one or two straight facades.

4.2.5. Imbalanced Dataset Correction

The dataset exhibits significant imbalance, primarily because most irradiation values are concentrated at the higher end of the spectrum (900 – 1000 kWh/m²). This section discusses the performance distribution across different irradiation bins and explores approaches to address this imbalance.

Loss Functions

Figure 4.22 illustrates the validation RMSE for four different loss functions used during the backward pass: MSE (base PointNeXt-L), DeltaLoss, ReductionLoss, and WMSE. For DeltaLoss, a delta value of 0.8 was applied. ReductionLoss was distributed over 10 bins, with the last bin being reduced. For Weighted MSE, a weight was assigned to the last irradiation bin of 0.25.

The loss function comparison indicates that alternative loss functions did not yield improvements over MSE, particularly when using 25 epochs and the default PointNeXt-L configuration. Additionally, using a larger nsample value (128) with MSE and WMSE did not result in significant performance enhancements in average RMSE.

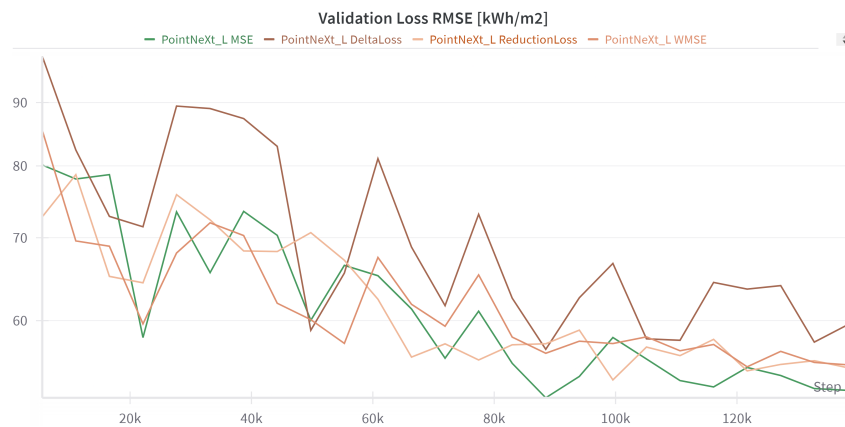


Figure 4.22.: Validation loss over 25 training epochs in kWh/m², considering four loss functions: Mean Squared Error, Delta Loss, Reduction Loss and Weighted Mean Squared Error.

4. Analysis

Distributed Performance

However, improvements in average test [RMSE](#) do not necessarily translate to better accuracy across the entire irradiation spectrum. Figure 4.23 illustrates the accuracy distribution across ten bins, evaluated using the optimized large PointNeXt network.

In figure 4.23:

- **Green bars** represent correct predictions, implying a maximum error of 100 kWh/m².
- **Yellow bars** indicate predictions that are off by one bin, reflecting a maximum error of 200 kWh/m².
- **Red bars** show predictions that are completely incorrect.

The bars are normalized according to the number of points in each bin (denoted as N), providing a clear view of the model's accuracy distribution across different irradiation values.

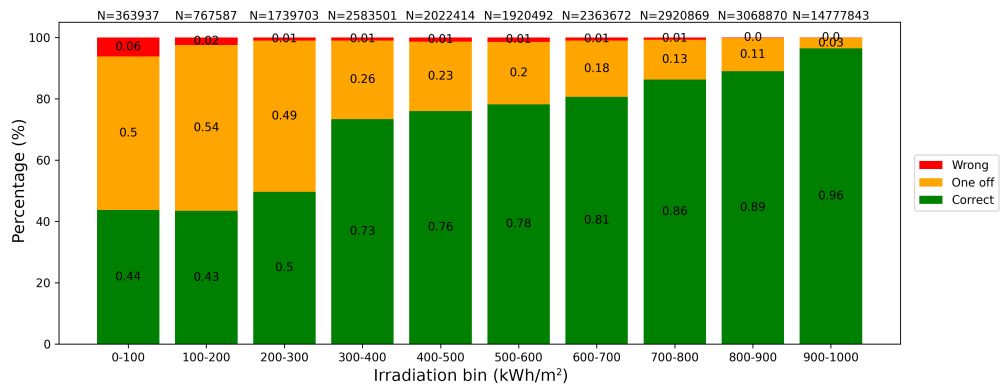


Figure 4.23.: Accuracy over the irradiation spectrum, binned in 100 kWh/m² domains.

Based on the bar plot, accuracy tends to decrease towards the lower end of the irradiation spectrum, which aligns with the dataset's irradiation distribution (see figure 4.4a).

Figures 4.24 and 4.25 illustrate the average [RMSE](#) over the test dataset for samples and points, respectively. Note that the point frequency in figure 4.25 is shown on a logarithmic scale. From this plot, it is evident that 92.8% of all points are predicted within the correct bin, with a maximum error of 50 kWh/m². Errors at the high end of the error spectrum are rare and occur only in exceptional cases.

The distributed performance has been visualized in the figure 4.26, through a confusion matrix. Figure 4.26a shows the ground truth confusion matrix with the corresponding colors and point frequencies.

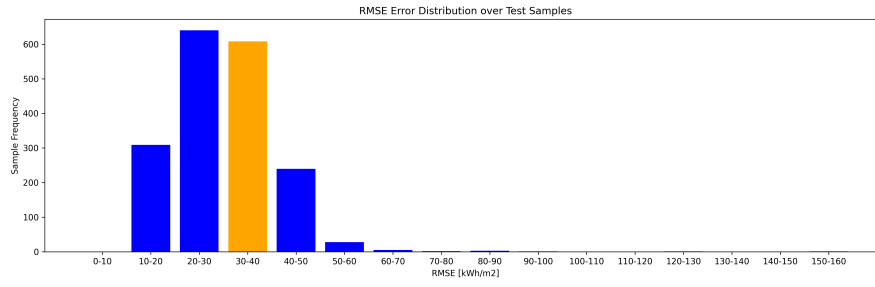


Figure 4.24.: Frequency of RMSEs over samples

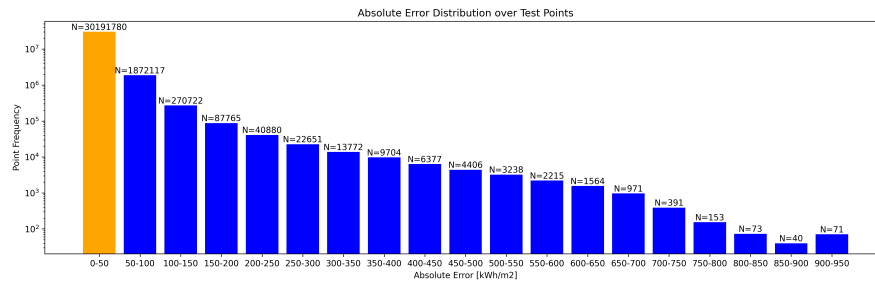


Figure 4.25.: Frequency of RMSEs over points

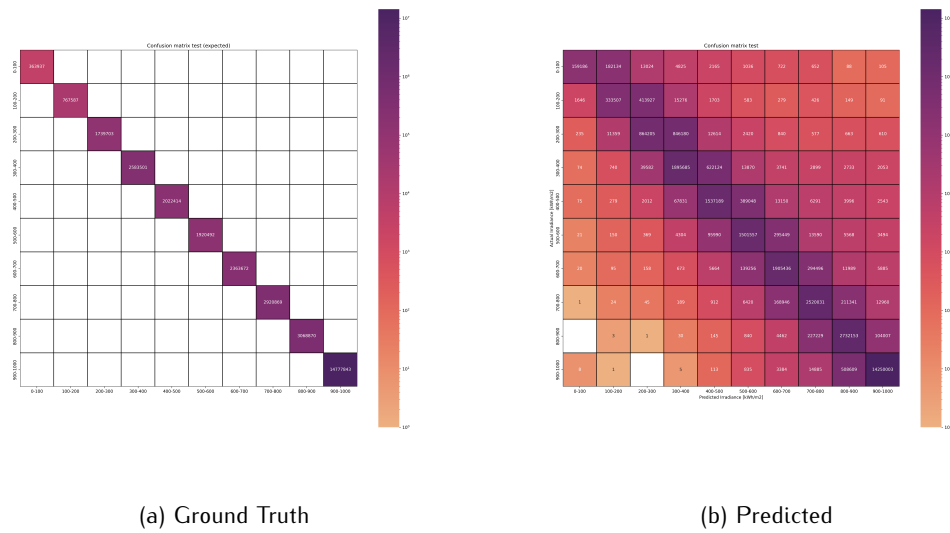


Figure 4.26.: Ground truth and predicted confusion matrix test dataset. Numerical values indicate point frequency.

4. Analysis

Adjusted Weighted MSE

Based on the performance distribution for point irradiation shown in figure 4.23, the weights in the **WMSE** loss function can be adjusted. Specifically, the weight for each expected irradiation value is derived from the error likelihood observed without weights. This adjustment results in a weight vector of $[0.56, 0.57, 0.5, 0.27, 0.24, 0.22, 0.19, 0.14, 0.11, 0.04]$ for the ten irradiation bins.

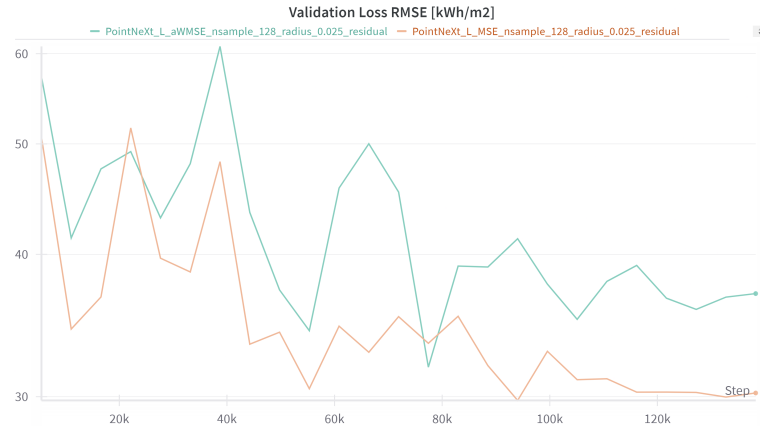


Figure 4.27.: Validation loss over 25 training epochs in kWh/m², for the **MSE** and adjusted **WMSE** loss functions

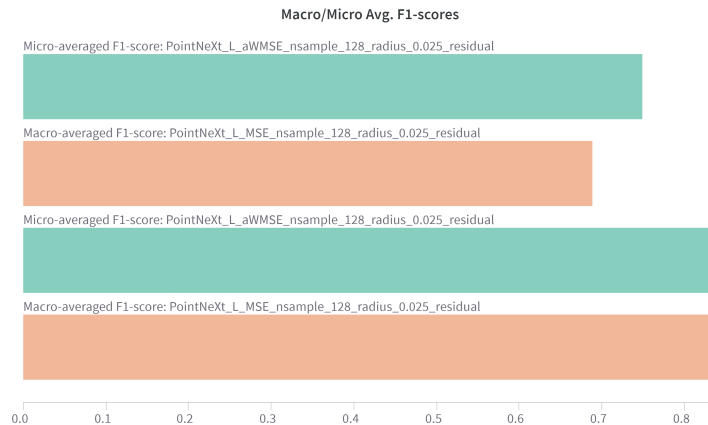


Figure 4.28.: Macro and micro avg. F1 scores for MSE and adjusted Weighted MSE loss function.

Based on the validation scores shown in figure 4.27, significant improvements are not observed. The most notable change is the increase in the macro-averaged F1 score, as illustrated in figure 4.28. However, the recall for each individual bin in the test dataset did not show substantial changes when transitioning from **MSE** to the adjusted **WMSE** loss function.

The recall scores for the lower end of the irradiation spectrum showed improvement with the Adjusted Weighted Mean Squared Error (**aWMSE**) loss function. However, the recall for the

highest bin decreased. This decrease can be attributed to the low weight (0.04) assigned to this bin and the fact that it contains fewer point samples compared to other bins.

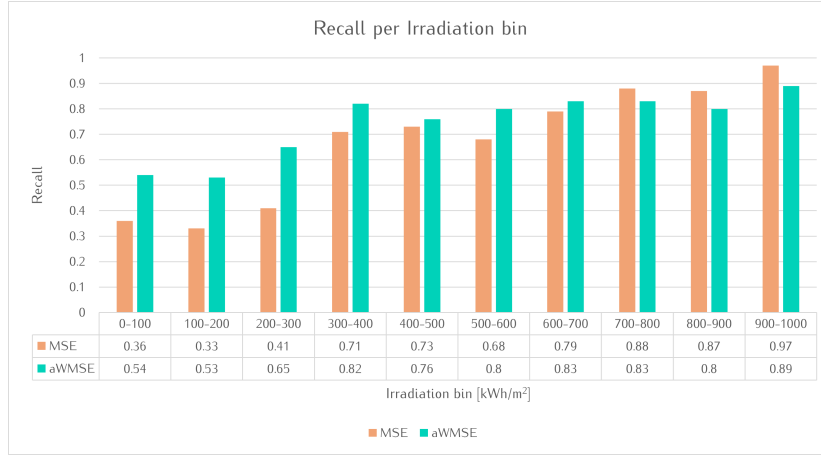


Figure 4.29.: Recall for MSE and aWMSE loss function.

4.2.6. Network Inference Optimization

Table 4.6 presents the average inference times for various network configurations, evaluated on the entire test dataset. Inference was performed on a GPU, leveraging CUDA-optimized grouping layers within the network architecture. The evaluation was conducted on a Windows laptop equipped with an Intel i7-9750H CPU, 24 GB of DDR4 memory, and an NVIDIA Quadro P2000 GPU with 4 GB of VRAM.

Configuration	Avg. Test Inference (s)
PointNeXt-L Base	0.184
PointNeXt-XL Base	0.291
PointNeXt-L Super	0.721
PointNeXt-XL Super	13.38

Table 4.6.: Inference on test dataset for several PointNeXt architectures on a laptop with dedicated GPU.

It is evident that base model architectures (nsample 32, radius 0.1, stride [1,4,4,4,4], no residuals) exhibit relatively faster inference times compared to more advanced configurations (nsample 128, radius 0.025, stride [1,4,4,4,4] with residuals). The choice of stride [1,4,4,4,4] over [1,2,2,2,2] was influenced by the limited VRAM available and its substantial effect on both inference and training times. Additionally, the extended inference time observed with the PointNeXt-XL Super model can be attributed to the constrained VRAM of the used GPU.

4.2.7. Experiment 1: Random Dataset

Two types of datasets were created: regular point samples and Poisson Disc point samples. For this experiment, a network with hyperparameters set to `nsample 128`, `radius 0.025`, `100 epochs`, and residual connections trained on Poisson Disc point samples. This model achieved an average test [RMSE](#) of 21.10 kWh/m², which is notably lower than the model trained on regular samples (avg. test [RMSE](#) 24.04 kWh/m²).

Comparing the performance of networks based solely on average test and validation RMSE is not feasible in this case due to the inherent differences between the datasets. The Poisson Disc point sampling approach does not remove dividing walls between buildings, which reduces pre-processing time. However, this results in the model predicting a larger number of zero irradiation values for these walls, as they do not receive solar exposure from outside. Consequently, direct comparison between test datasets is not valid.

A more accurate evaluation method involves visual inspection of the plotted point clouds and corresponding irradiation values. Figure [4.30](#), [4.31](#) and [4.32](#) provide an overview of samples with the highest, median and lowest irradiation errors in the Poisson Disc dataset.

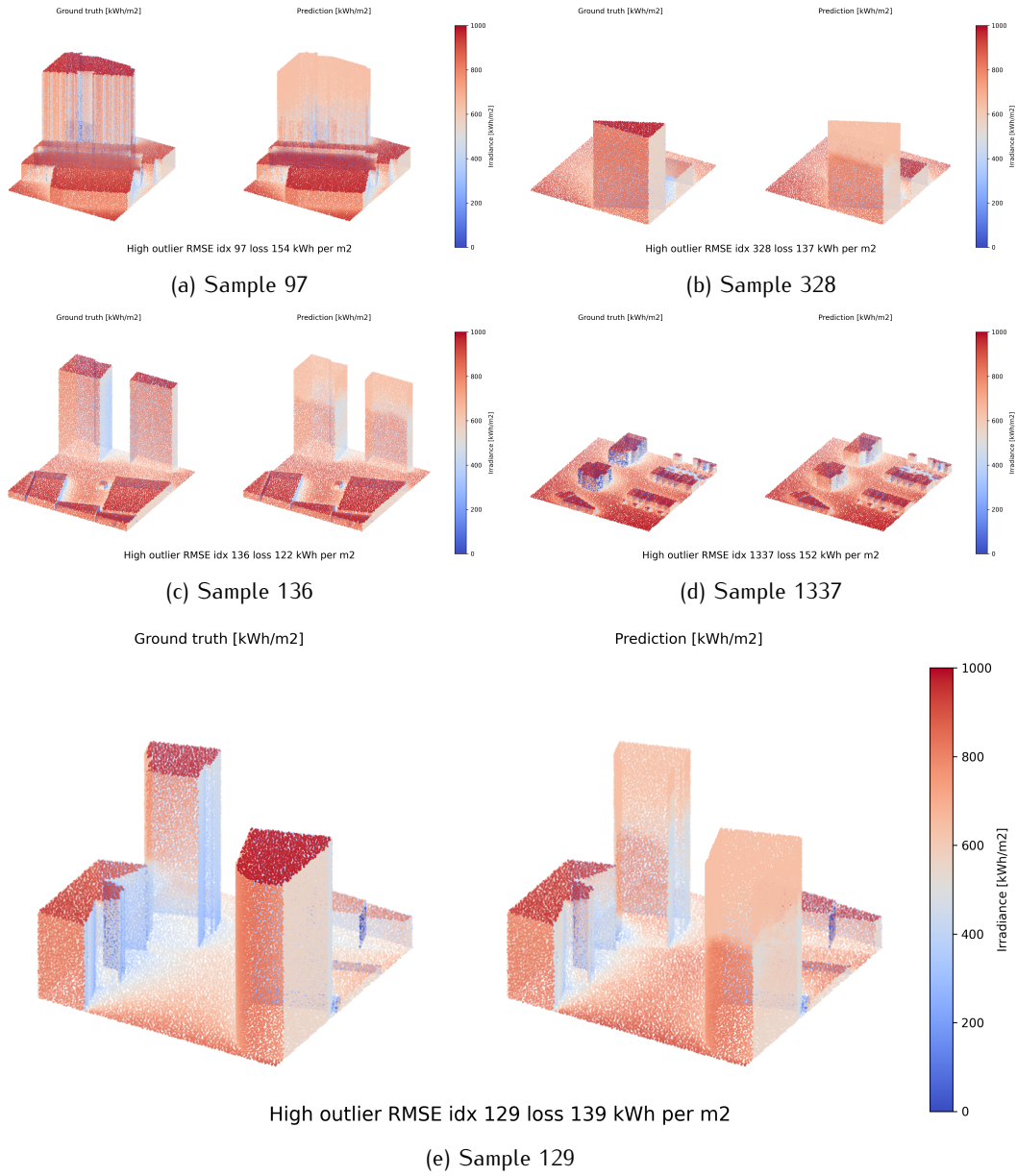


Figure 4.30.: Highest irradiation $RMSE$ s for test dataset (Poisson Disk) prediction, based on the large PointNeXt model with tuned hyperparameters nsample 128, radius 0.025, encoder-decoder residual True.

4. Analysis

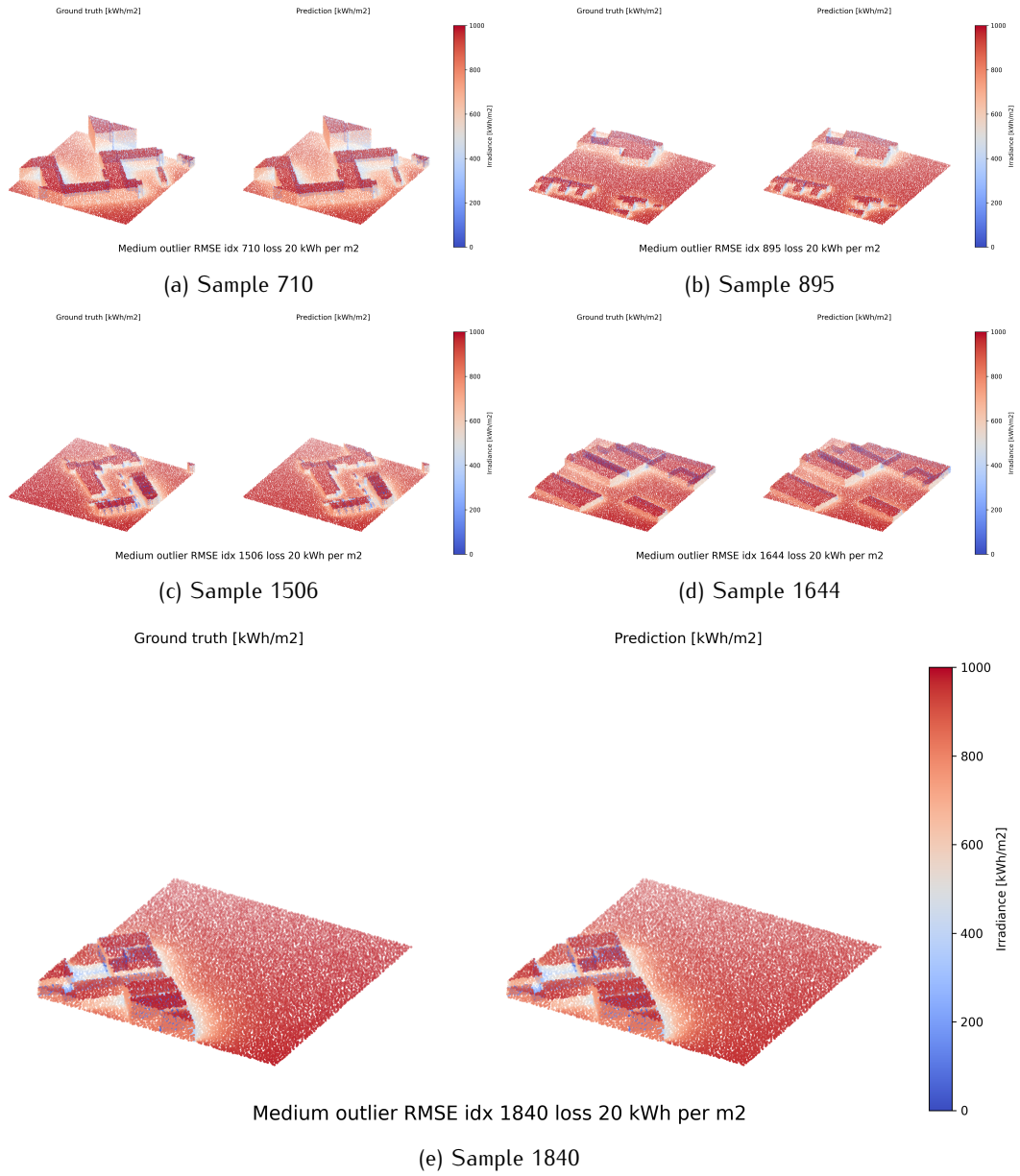


Figure 4.31.: Median irradiation RMSEs for test dataset (Poisson Disk) prediction, based on the large PointNeXt model with tuned hyperparameters $n_{\text{sample}} 128$, radius 0.025, encoder-decoder residual True.

4.2. Prediction

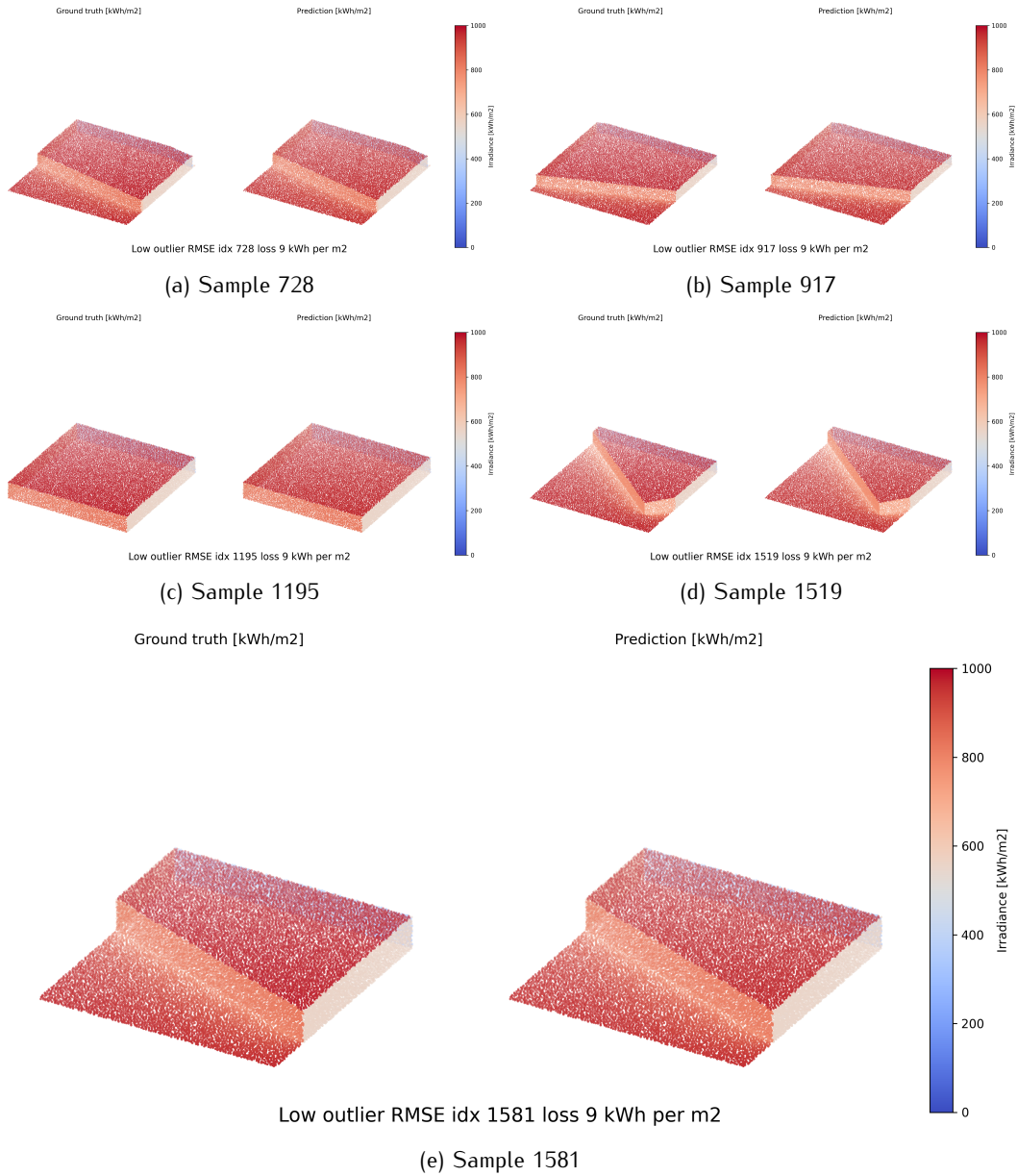


Figure 4.32.: Lowest irradiation **RMSEs** for test dataset (Poisson Disk) prediction, based on the large PointNeXt model with tuned hyperparameters nsample 128, radius 0.025, encoder-decoder residual True.

4. Analysis

Furthermore, the accuracy for individual irradiation bins, errors of irradiation and sample RMSEs can be analyzed as can be seen in figure 4.31, 4.32 and 4.33 respectively.

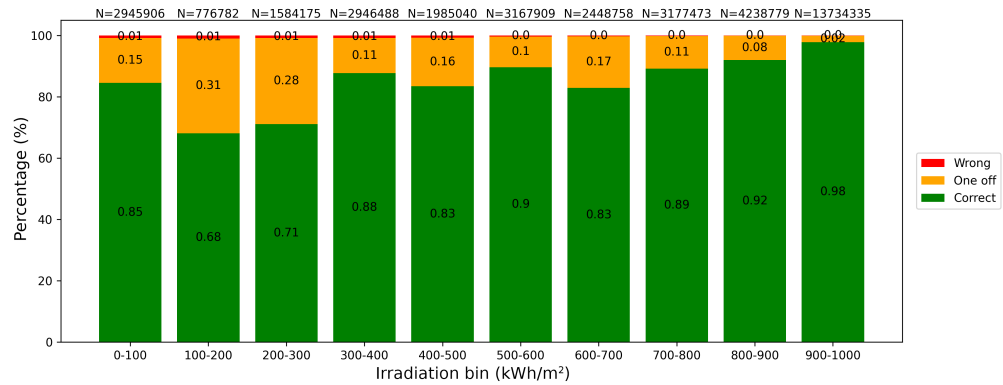


Figure 4.33.: Test accuracy of individual irradiation bins for a network trained on a Poisson Disk dataset. X-axis expressed in kWh/m², Y-axis in %.

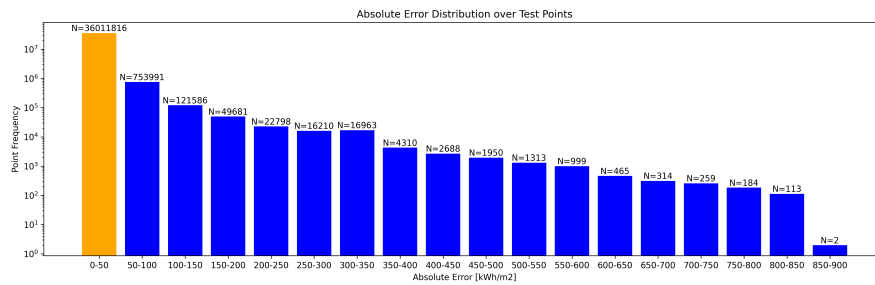


Figure 4.34.: Error of individual points for a network trained on a Poisson Disk dataset.

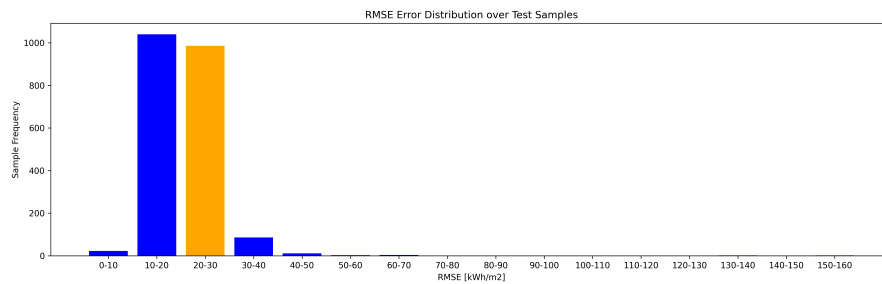


Figure 4.35.: RMSE for individual test samples in the Poisson Disk test dataset.

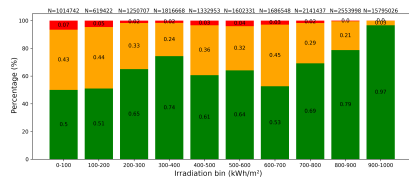
4.2.8. Experiment 2: Sample Size

While the initial training experiments focused on 100x100m samples, it is valuable to assess performance on larger scale samples. To this end, a dataset of 300x300m was created, using both regular and Poisson Disk point samplings. As with the previous approach, point coordinates were normalized to the $[-1, 1]$ domain. For both datasets, a network with the following hyperparameters was used: nsample 128, radius 0.025, 100 epochs, and residual connections set to True.

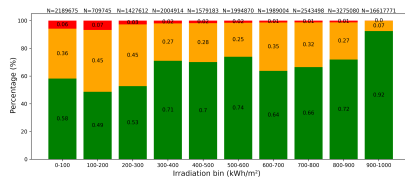
Figures 4.37 and 4.38 illustrate the highest errors observed for the regular and Poisson Disk 300m x 300m datasets, respectively. Similarly to experiment 1, avg. RMSE values can not be used for comparison with 100x100m samples, due to the significantly greater number of points per sample.

While it is clear that the network is able to understand general patterns in the 300x300m dataset, high outliers show different issues. For example, figure 4.37e shows how the model struggles with long distance influence of buildings casting shadows. This is most likely caused by the limited receptive field of the network.

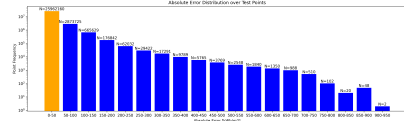
An overview of bin accuracies, individual sample and point errors are given in figure 4.36.



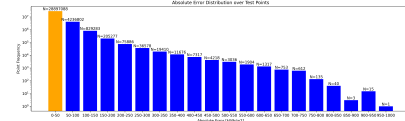
(a) Bin accuracy regular 300x300m test dataset



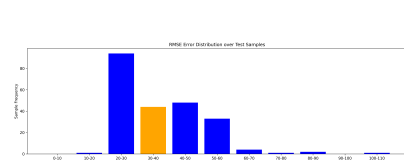
(b) Bin accuracy Poisson Disk 300x300m test dataset



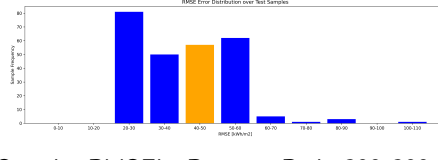
(c) Point errors regular 300x300m test dataset



(d) Point errors Poisson Disk 300x300m test dataset



(e) Sample RMSE's regular 300x300m test dataset dataset



(f) Sample RMSE's Poisson Disk 300x300m test dataset

Figure 4.36.: Bin accuracy, point RMSEs and sample RMSEs for regular 300x300m dataset (left) and 300x300m Poisson Disk dataset (right)

4. Analysis

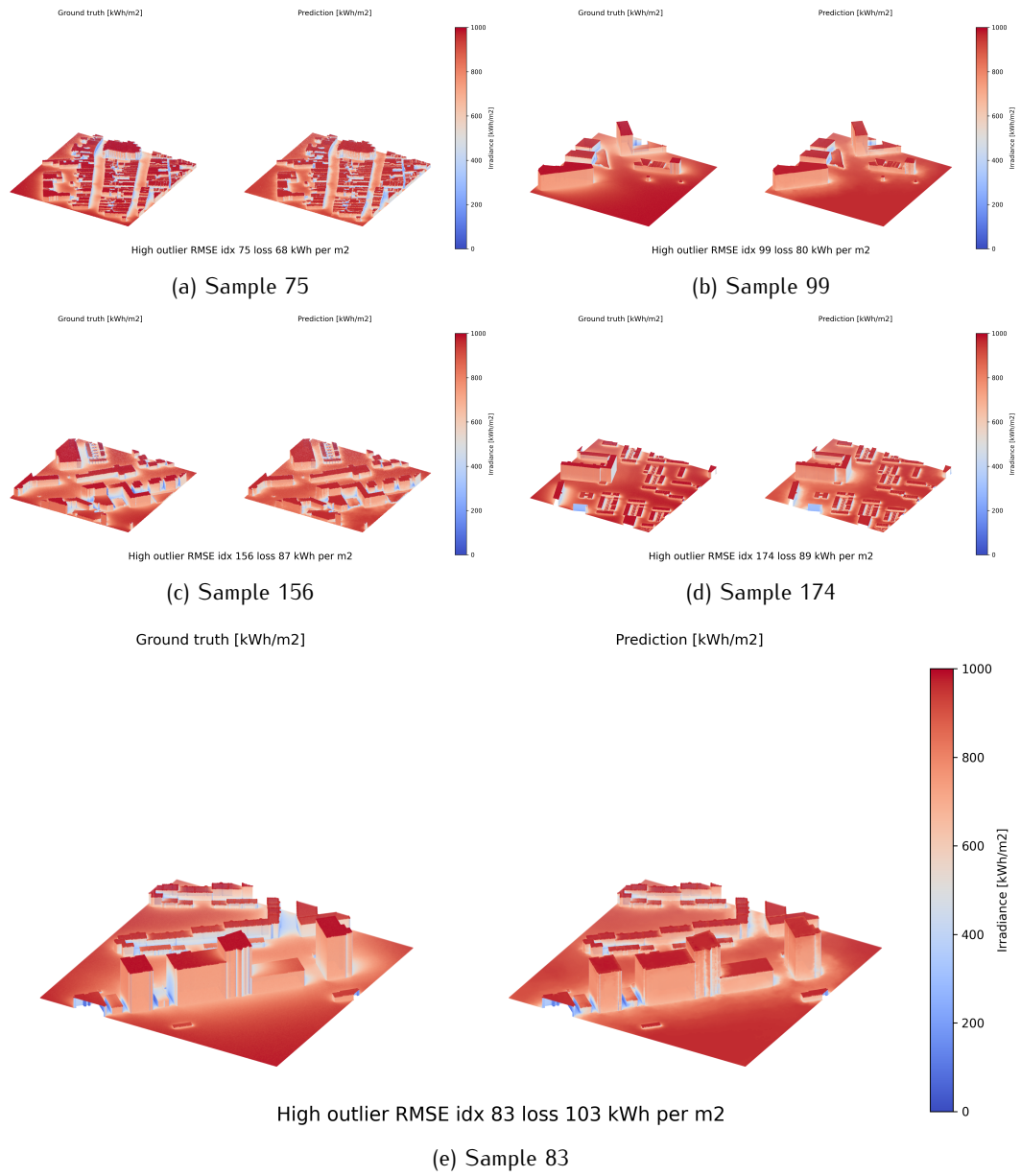


Figure 4.37.: Highest irradiation **RMSE**'s for 300x300m test dataset prediction, based on the large PointNeXt model with tuned hyperparameters Nsample 128, radius 0.025, encoder-decoder residual True.

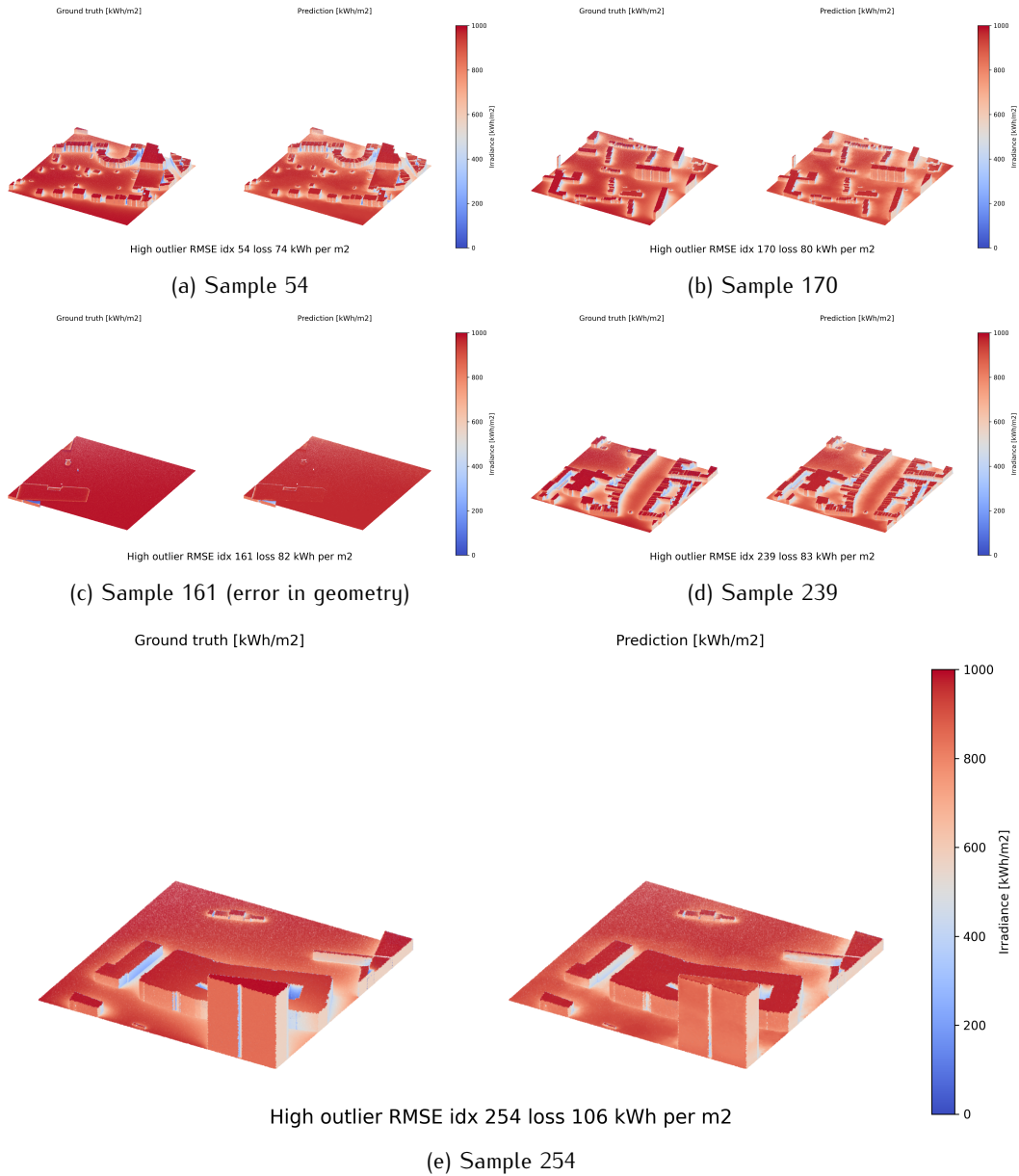


Figure 4.38.: Highest irradiation **RMSE**'s for 300x300m test dataset (Poisson Disk) prediction, based on the large PointNeXt model with tuned hyperparameters Nsample 128, radius 0.025, encoder-decoder residual True.

4. Analysis

4.2.9. Conclusion

In this section, the performance of various PointNet-based architectures for predicting irradiation has been analyzed. The results indicate that a finely tuned PointNeXt-XL model achieves an average [RMSE](#) of 24.04 kWh/m² on the regular dataset and 21.10 kWh/m² on the Poisson Disc dataset, without applying dataset imbalance corrections. Evaluation of larger scale samples (300m x 300m) revealed new issues, most likely due to the limited receptive field of the network. Additionally, the finetuned PointNeXt-XL model has a relatively high inference time of 13.38 seconds compared to 0.721 seconds for the finely tuned PointNeXt-L network, when run on a laptop with a dedicated laptop [GPU](#). While correcting for dataset imbalance can improve accuracy in the lower end of the irradiation spectrum, it may negatively impact performance at the higher end.

4.3. Interaction

Through the proposed server-client system (see Figure 4.39), it is possible to boot a trained neural network, waiting to receive a new sample from the client in Grasshopper. For this experiment, a laptop with an Intel® Core i7-9750H CPU, 24GB of RAM, and an NVIDIA Quadro P2000 GPU was utilized.²

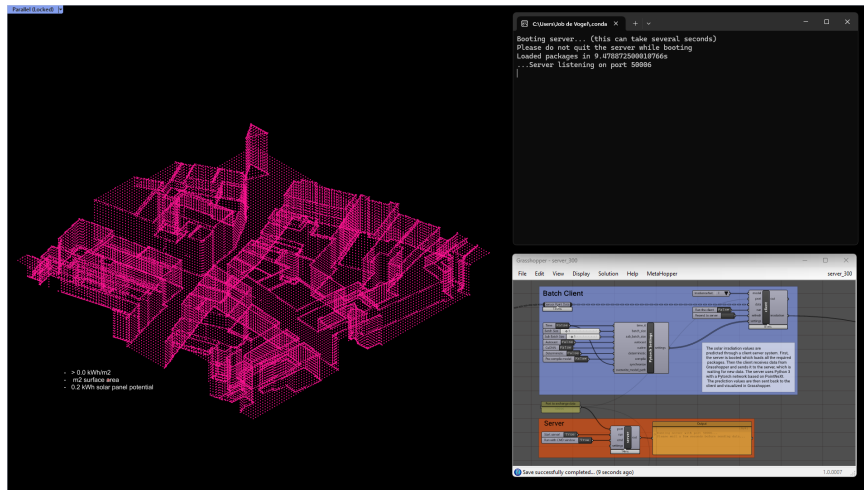


Figure 4.39.: Interaction environment in McNeel Rhino. The Grasshopper script controls the data flow from download to visualization. The CMD window in the top right is the server which will predict the irradiation values.

Download Context

The first step in the interaction workflow (see Figure 4.40) involves downloading and extracting a sample from the 3D BAG. For demonstration purposes, a sample from the city of Groningen was chosen as an example. The download speed for this process varies depending on the sample size, local network speed, and the complexity of the geometry.

Preprocessing

Within the Grasshopper dataflow (see Figure 4.41), a distinction is made between contextual geometry and the design. This separation helps to optimize preprocessing speed. Preprocessing involves all procedures related to mesh discretization, computation of sensor points, and calculation of normal vectors.

²It was essential to conduct the experiments on a typical design laptop rather than high-performance hardware, in order to simulate real-world conditions where the workflow would be applied.

4. Analysis

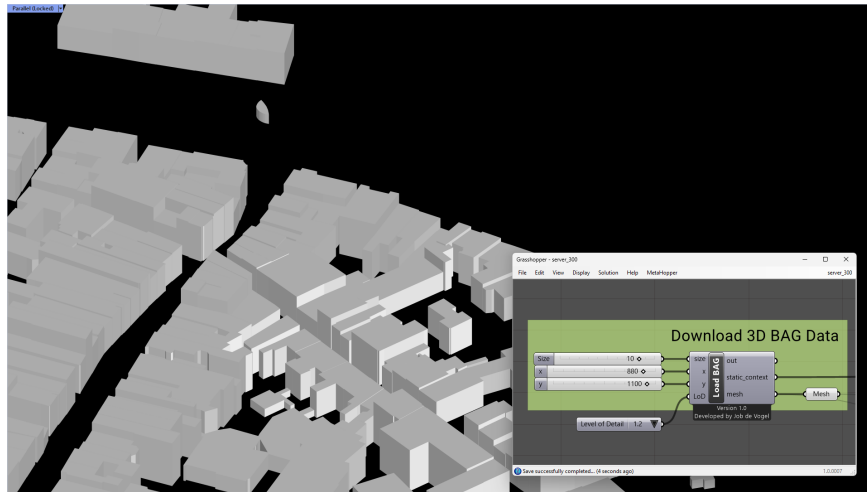


Figure 4.40.: A Python script in Grasshopper downloads 3D BAG data based on given coordinates. The **LoD** input can be used to specify the type of geometry.

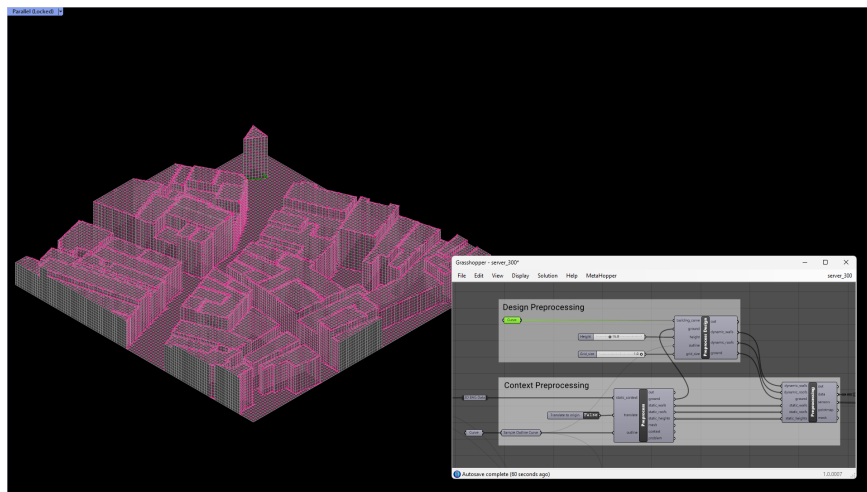


Figure 4.41.: Preprocessing the context and design separately. Pink dots indicate the positions of the sensor points. The gray underlying geometry is a quad-triangle mesh with regularized faces.

Server Activation

After preprocessing, the user can activate the server through a single Grasshopper node (see Figure 4.42). Upon startup, all necessary Python packages are preloaded. The user has the option to select a port for the server, decide whether the CMD window should be visible, and specify optional settings for loading the model in PyTorch. Additionally, the server can be run on an external device if the local machine lacks sufficient processing power.

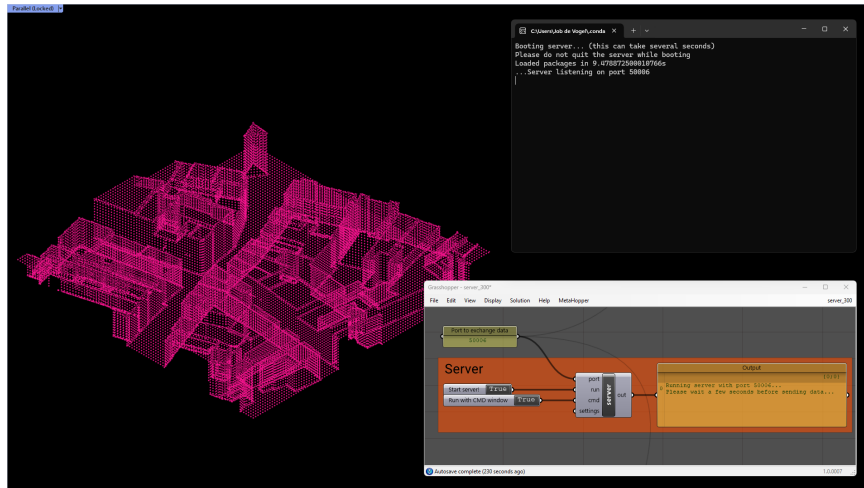


Figure 4.42.: The server waiting for a call to predict irradiation on a given point cloud.

4. Analysis

Sending Data through Client

The client first receives information regarding the model type and the associated PyTorch settings. To optimize inference, multiple point clouds are padded to similar dimensions, allowing them to be processed in batches. The initial run may take longer because the model needs to be loaded and, depending on the settings, converted to machine code. After inference, the irradiation values are transmitted back from the server to the client, where they can be visualized in Rhino (see Figure 4.43).

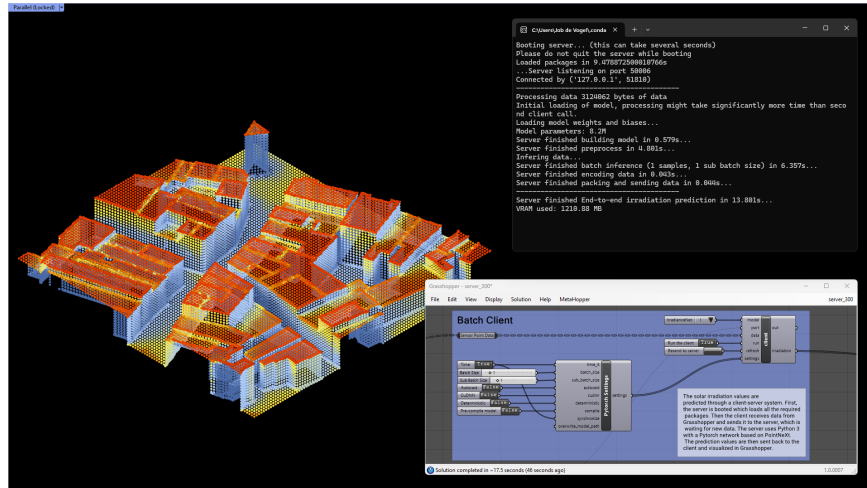


Figure 4.43.: The server processing an input from the client in the Grasshopper environment.

Irradiation Visualization

In addition to visualizing the irradiation as a colored point cloud, users can also apply color to the mesh faces using the mesh visualizer (see Figure 4.44). However, this option is only available for meshes that use the regular point sampling method.

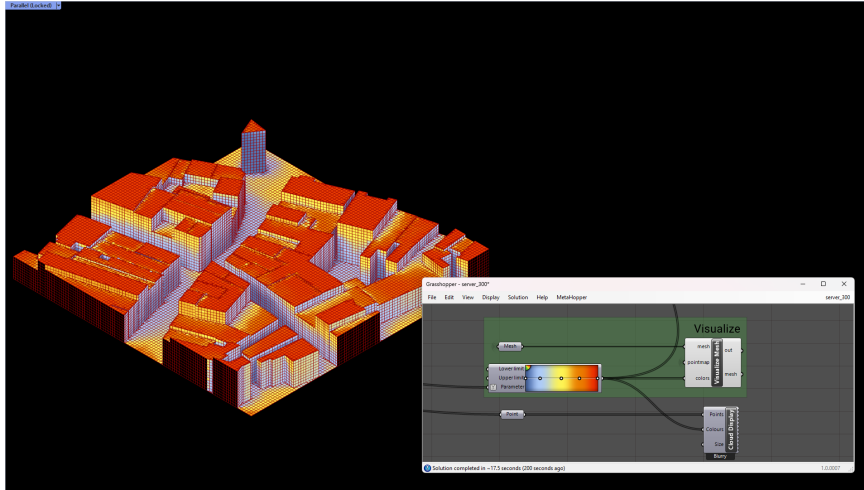


Figure 4.44.: The server waiting for a call to predict irradiation on a given point cloud.

Horizontal/Vertical Irradiation Extraction

Additionally, a Grasshopper script was implemented to extract irradiation values based on a threshold for both vertical (see Figure 4.45) and horizontal (see Figure 4.46) surfaces. By incorporating an average annual solar panel efficiency, this script allows for the estimation of potential power generation.

$$power_generation_potential = \sum_{i=0}^n irradiation_i \cdot face_area_i \cdot panel_efficiency \quad (4.1)$$

in which i is the index of the available mesh faces above the threshold irradiation (kWh).

4. Analysis

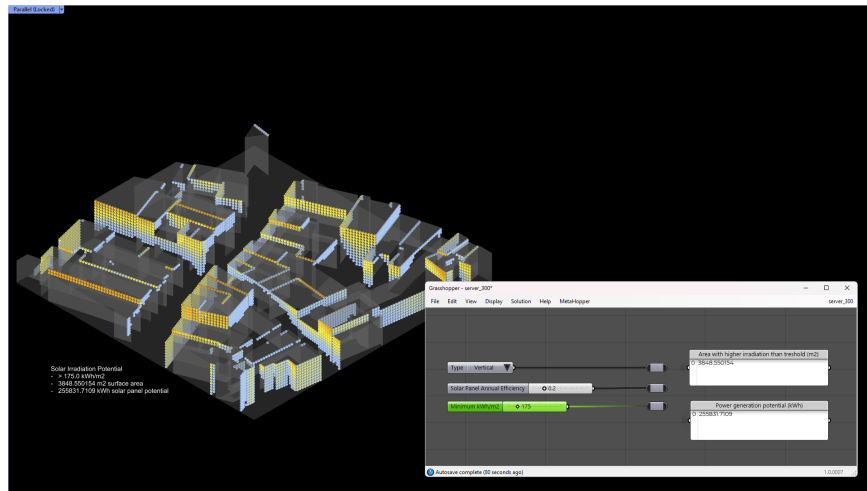


Figure 4.45.: Vertical irradiation values higher than 175 kWh/m²

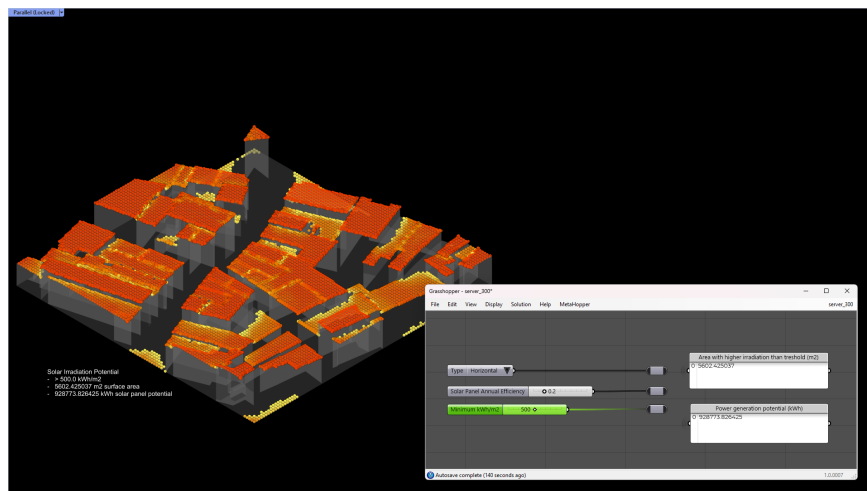


Figure 4.46.: Horizontal irradiation values higher than 500 kWh/m²

4.3.1. Optimization

By integrating a brute-force optimization algorithm with the proposed irradiation predictor, one can fine-tune design parameters based on a fitness function as described in section 3.6.4. Figure 4.47 demonstrates how this procedure could be applied. The enhanced speed of irradiation prediction, compared to simulation, facilitates testing numerous design alternatives more efficiently. In this specific example, batch optimization was not yet implemented due to the limited GPU VRAM on the local machine.

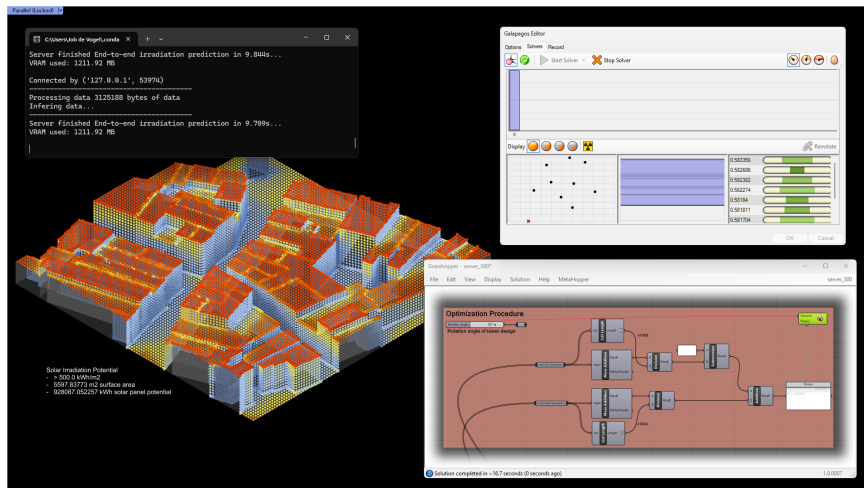


Figure 4.47.: Optimization procedure in which the irradiation on horizontal surfaces is maximized, and the irradiation on vertical surfaces is minimized.

4.3.2. Overall Grasshopper Script

Figure 4.48 shows the integrated Grasshopper script used for the entire workflow. For clarity, some wires are hidden. A substantial portion of the script's functionality is implemented through Python within the Grasshopper nodes.

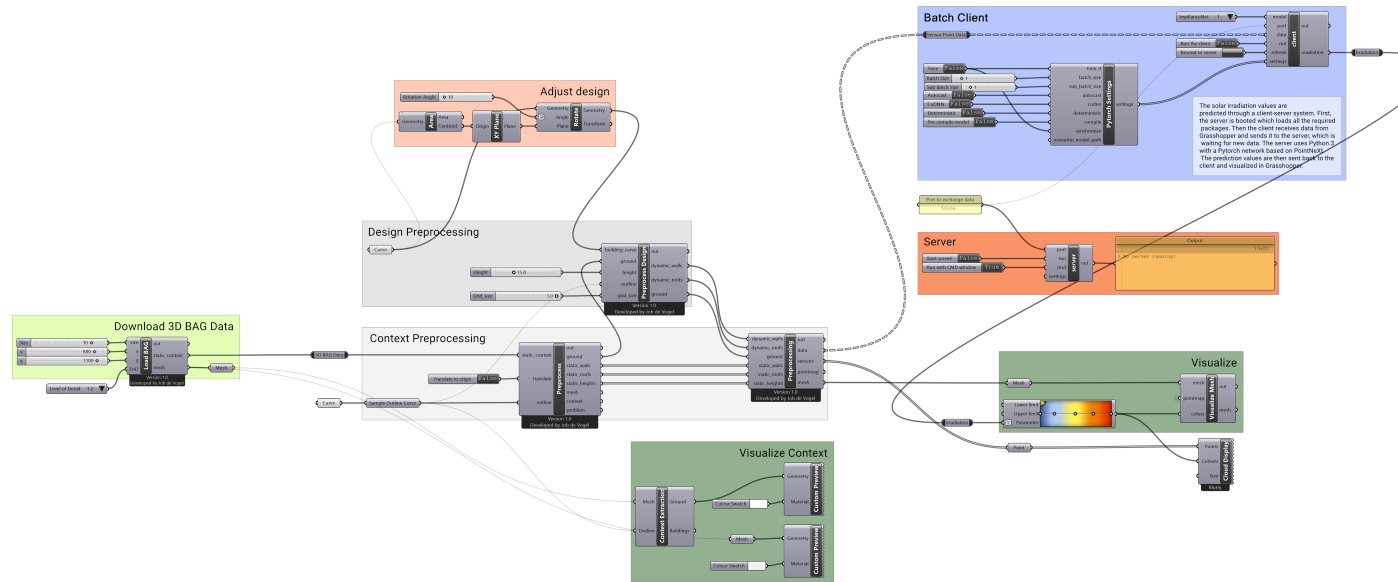


Figure 4.48.: Grasshopper script used for the prediction and optimization of solar irradiation on urban designs. For clarity, the horizontal/vertical surface irradiation extraction script is not included. This part of the script was fully developed in Grasshopper, and simply extracts irradiation values based on the normal direction of the sensor point.

4.3.3. Future Design Framework

Figure 4.49 illustrates how this research could contribute to future design frameworks. Whereas simulations are currently used as evaluative tools to analyse buildings, generative models could aid architect in giving concrete advice on how to change the building. The design framework could consist of multiple 3D views in which one is used by the designer, multiple as prediction visualizations and one by a generative AI model which provides suggestions how to change the design.

This process is deemed possible due to the descriptive nature of encoder networks which compress features from a building. The so called latent space of these encoders (which is the compressed representation) can be combined to give an overall representation of the design, considering multiple design factors. A decoder could then be used to reconstruct the design to alternative options with better performance, based on experience from other building designs.

Other performance features such as financial costs and practical feasibility could also be included as indicated in figure 4.50.

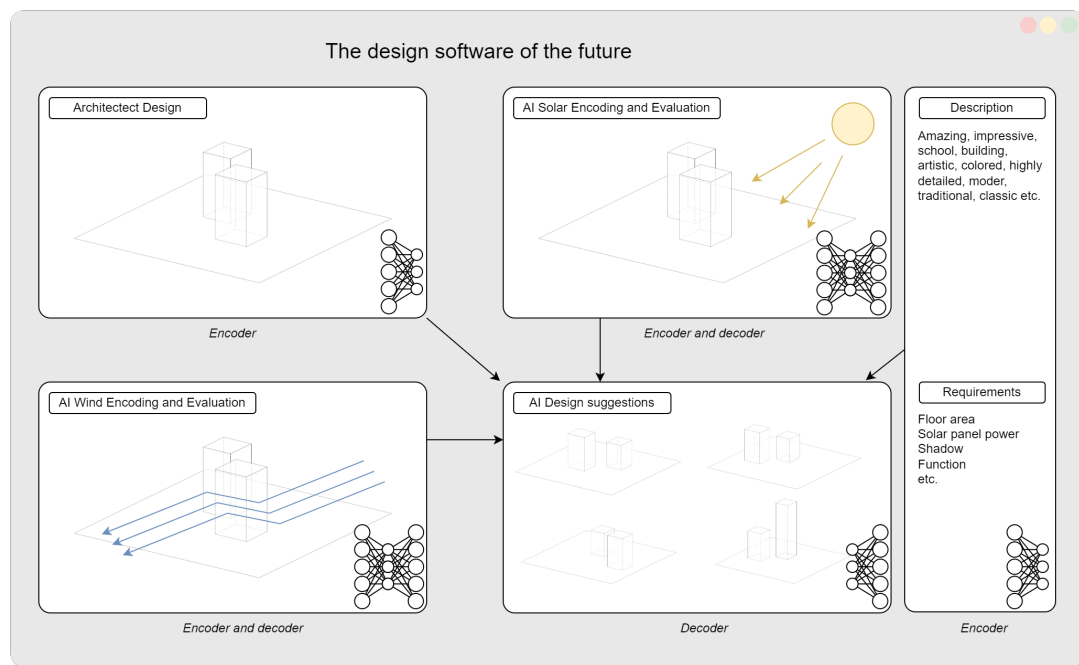


Figure 4.49.: Future research could investigate the potential of transforming the suggested predictive model to a generative neural network. Combining multiple performance features and style/program suggestions by the designer, could lead to a new design framework, as is visible in this figure. On the top left: the initial design. Top right: predictions on solar performance. Bottom left: predictions on wind performance. Right: suggestions by the designer regarding textual description and requirements. Bottom right: alternative suggestions by a generative model based on all inputs together.

4. Analysis

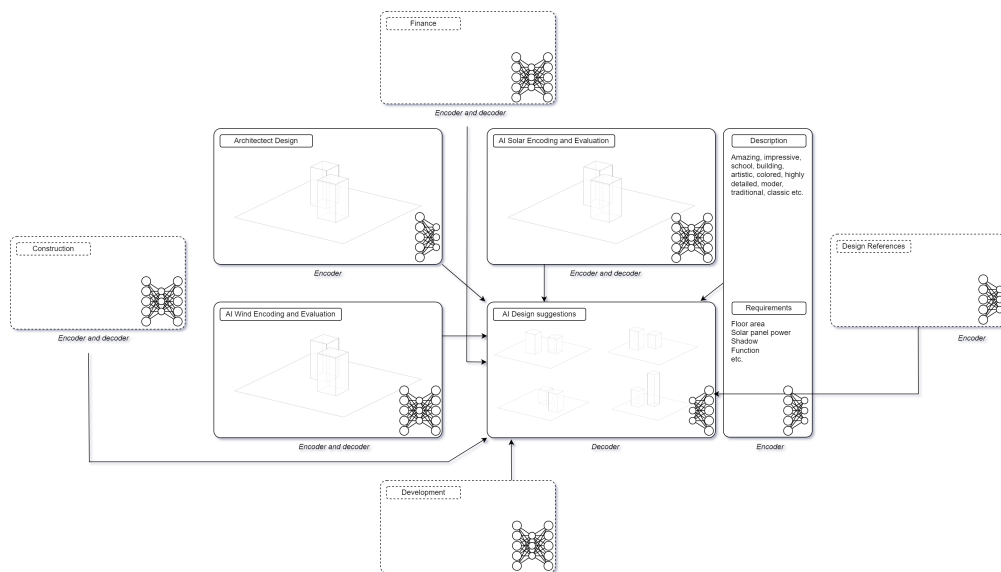


Figure 4.50.: A generative model could encapture many other relevant building domains as well, such as financial costs and practical feasibility.

5. Discussion

In this chapter, the outcomes of the research on dataset generation, simulation, and parallelization, as well as the prediction of solar irradiation and the interaction with the model, are thoroughly discussed. The discussion is structured around the main research question and its associated subquestions. Furthermore, the methods implemented in this thesis are critically analyzed, addressing their limitations, followed by recommendations for future research.

5.1. Research Questions

This thesis aimed to answer the central research question: "How can one predict annual solar irradiation on 3D urban geometry using deep neural networks?" To address this, five subquestions were formulated and will be answered based on the research results.

How do simulation models compute solar irradiation on 3D geometry?

Chapter 2.1 details how simulation models, particularly the 2-phase method, calculate solar irradiation on 3D geometry. The 2-phase method utilizes a non-linear and partially stochastic equation based on ray tracing principles to approximate the solar energy or light received by geometric surfaces. By generating a daylight coefficient matrix, this method efficiently estimates solar irradiation across many points in parallel, making it a robust tool for large-scale urban geometry simulations.

What type of input dataset is required to predict solar irradiation using deep neural networks?

The input dataset required for predicting solar irradiation with deep neural networks is closely tied to the type of network employed. As discussed in Chapters 2.3 and 2.4, various approaches exist for transforming 3D data into formats compatible with specific network architectures. This research opted for PointNet-based architectures, which necessitate a point cloud dataset including geometric normal directions and expected irradiation values. These networks offer several advantages:

- The inputs effectively describe both the shape of the geometry and the locations requiring irradiation predictions.
- The relatively low-dimensional inputs, compared to other network types, enhance scalability.
- PointNet-based architectures have demonstrated near state-of-the-art (SOTA) performance on similar tasks, such as S3DIS segmentation.
- The selected architecture (PointNeXt) is relatively straightforward to tune compared to typically used Transformer models.

5. Discussion

How can one compute a dataset with 3D urban geometry with solar irradiation values efficiently?

The thesis proposed a parallelized solution, integrated with the AccelerRad GPU-based simulator, to efficiently generate four datasets with a substantial number of samples. This approach allowed for the efficient computation of 3D urban geometry datasets with corresponding solar irradiation values, significantly reducing the time and computational resources required.

Can deep neural networks predict solar irradiation values accurately, faster than a simulation model?

The research demonstrated that deep neural networks could predict solar irradiation values more quickly than traditional simulation models. The implemented model exhibited faster prediction times while maintaining accuracy comparable to low-parameter simulation models. However, the model faced limitations, including reduced geometric detail, long distance geometric relationships and challenges with location and time invariance. Future research could focus on enhancing the model accuracy and addressing dataset imbalances across the irradiation spectrum.

How can architects interact with the deep learning model in a user-friendly ecosystem?

A client-server system was developed to facilitate user interaction with the deep learning model. Geometry preprocessing occurs on the client side, while irradiation prediction is handled on the server side. This system was integrated into McNeel Grasshopper, a parametric design software, making it accessible to designers without programming expertise. However, the client-server principle is also applicable in other design software. The research demonstrated how designers could efficiently interact with the model to optimize their designs, and the system's flexibility allows for easy integration into other design frameworks.

Main research question: *"How can one predict annual solar irradiation on 3D urban geometry using deep neural networks?"*

The research concluded that it is feasible to predict solar irradiation on 3D urban geometry by training PointNet-based deep neural networks on simulated irradiation values. While the model exhibits variability in accuracy across the irradiation spectrum and faces limitations in handling complex geometric details, they offer a promising approach to efficiently predicting solar irradiation, providing a valuable tool for architects and urban planners.

5.2. Generation

In this research, multiple irradiation datasets were developed using the 3D BAG dataset, with preprocessing methods that generated either regular or random (Poisson Disk) patterns of sensor points on 3D urban geometry. These point clouds, paired with normal directions, served as abstract descriptions of the geometric shapes and indicators for the positions where solar irradiation was to be simulated and predicted.

5.2.1. Regular Mesh Preprocessing Limitations

The findings of this thesis highlight several limitations of the regular sampling technique. Primarily, errors often arise from Boolean splitting operations. Moreover, handling more detailed geometry requires customized algorithms to manage various geometric shapes, which complicates the process. As a result, the Poisson Disk point sampling method is recognized as a more effective solution for geometric preprocessing. While predictions on randomly sampled points may lead to higher irradiation errors, the robustness and generality of the Poisson Disk method outweigh

these concerns. Unlike during dataset development, where invalid samples can be removed, such flexibility is unavailable during actual model usage. Therefore, the Poisson Disk sampling method is considered the best choice.

Proper visualization of irradiation values on a design is crucial. With the Poisson Disk method, the random distribution of points means that they do not align with the faces of the underlying mesh. Typically, corresponding mesh faces are colored based on irradiation values, as seen in tools like Ladybug. For practical implementation, it is recommended to explore alternative methods for coloring the mesh. A potential approach could involve generating an excess of sensor points and then mapping the closest irradiation values to each mesh face centroid.

5.2.2. Poisson Disk Sampling Optimization

In the current proposed preprocessing methods, points are sampled for each individual 100m x 100m sample (or 300m x 300m). For Poisson Disk sampling however, it is possible to accomplish similar results by taking the entire 3D BAG file, due to linear scaling of the algorithm time complexity. This will most likely result in a significant speed-up of the preprocessing procedure.

In the preprocessing phase of Poisson Disk point sampling, an alternative approach to mesh discretization should be considered to enhance efficiency. Currently, the meshes are discretized into a regular grid of quad meshes, which is an unnecessary step that adds to the computation time without contributing significantly to the overall process.

5.2.3. Augmentation

Several augmentation techniques were proposed to enhance the dataset generation process's efficiency. These included translational, rotational, and scaling augmentations to increase the number of samples derived from a single 3D BAG file. While useful for evaluative purposes, these methods may introduce misleading data for future generative deep learning models, as they train the network on geometry that doesn't exist in reality. Thus, the author advises against using augmentation in building physics prediction applications.

5.2.4. Framework Limitations

In this research, most geometric operations were performed using the Rhino.Inside API, due to the ease of use and the integration with the Ladybug/Honeybee package. However, potentially problematic for the code implementation in other software packages, is the required accessibility to a McNeel Rhino license to make use of the code. To this aim, one could think of the following solutions:

- Alter the Honeybee code package to deal with non-Rhino mesh and point formats, thus allowing use of CPython/C++ based geometric operations;
- Directly transform the geometric data to Radiance/Accelerate readable formats using CPython/C++ based geometric operations.

5. Discussion

This change to the proposed method could lead to better integration into other design software (such as Autodesk Revit, ArchiCAD or Vectorworks), allow computation on other operating systems (such as Linux and Windows Server), better optimization of the preprocessing phase and more accurate boolean intersection algorithms for mesh discretization.

5.2.5. Further Research

For generation, the following research steps are suggested to reach production level accuracy and usability of solar irradiation prediction:

1. Optimize Poisson Disc point sampling for entire 3D [BAG](#) files;
2. Add points at specific parts of the geometry, e.g. edges and corners;
3. Explore alternative colored mesh visualization methods, considering that faces do not correspond with sensor point locations;
4. Rewrite the code so it does not depend on Rhino based Python packages.

5.3. Simulation

The ground truth irradiation values predicted in this thesis are based on simulated data generated by the AcceleRad simulation engine. Ideally, real observed data would be used, but this was not available in the context of this research. The use of synthetic irradiation data and the methods employed in this thesis come with several limitations.

5.3.1. Accelerad vs Radiance

As the original authors of Accelerad have validated, discrepancies exist between the simulated values by Accelerad and Radiance. Largest errors are found in illuminance predictions, which are based on the same mathematics as irradiation prediction. Therefore, it was important to define the best settings through a parameter convergence test.

To limit the scope of this research, exceptional large differences between Accelerad and Radiance were accepted for the purpose of this thesis. The reduced computation time of Accelerad outweighs the importance of most accurate predictions, also considering that neural network based predictions make small errors as well.

5.3.2. Direct vs Indirect Irradiation

Based on empirical evidence, it was concluded that indirect irradiation only has a small contribution to the total irradiation, but costs the most time to compute. Therefore, future researchers may consider to only predict the indirect component using deep neural networks, and compute the direct part using traditional methods with optimized matrix operations on [GPU](#).

5.3.3. Materials

In the current stage of this research, materials have not been accounted for in the simulation and prediction of irradiation, due to lack of open source data. However, it is important to note that materials have a great influence on the transmittance-reflectance term of the solar irradiation computation. Therefore, it is advised to append material features to the dataset with reflectance, specular and roughness values for each sensor point.

5.3.4. Further Research

For simulation, the following research steps are suggested to reach production level accuracy and usability of solar irradiation prediction:

1. Add material features to the dataset;
2. Consider subtracting the direct irradiation component and predict the indirect component only.
3. Finetune Accelerad specific simulation parameters to match the Radiance simulation values.

5.4. Parallelization

The parallelization procedure in this thesis has led to a computation time reduction from 3.965s to 608s for 100 regular 100m x 100m samples on a high-end desktop.¹ Without parallelization, generating the entire dataset would have taken more than a week of computation.

5.4.1. Further Research

Further improvements could be achieved by optimizing the AcceleRad/Radiance simulation speed. Currently, samples are sent sequentially to AcceleRad in each process. This could potentially be optimized by calling AcceleRad only once per process and sending batches of models for simulation.

5.5. Prediction

The results of this thesis have demonstrated that the loss for irradiation prediction generally decreases during the training of PointNet-based networks. Comparing the accuracy of this implementation with earlier research is challenging due to differences in dataset types and specific limitations inherent to each study. However, it is evident that the proposed solution is significantly more scalable than previous efforts and offers promising accuracy that can be further optimized in future work.

¹Assuming linear time scaling, it is expected that a 300m x 300m regular dataset would take a similar amount of time to be computed sequentially.

5. Discussion

5.5.1. Practical Implications of "21 kWh/m² RMSE"

This thesis has developed a deep learning model capable of predicting irradiation values with an average test [RMSE](#) of 21 kWh/m². When this error is compared to the parameter convergence test for AcceleRad, it is roughly equivalent to a simulation conducted with relatively low parameter settings. However, the practical implications of an error of this magnitude are not immediately clear. The following example provides more insight into these implications:

Consider a scenario in which urban geometry in Rotterdam receives a maximum irradiation of 960 kWh/m². The goal is to install solar panels at locations where this maximum irradiation can be captured. Assuming the solar panels have an average efficiency of 20% at optimal orientation and angle, and given that the power consumption of the household is 3,500 kWh annually, 19 m² of solar panels is required to meet the energy needs:

$$required_solar_panel_area = \frac{3500}{0.2 \cdot 960} \approx 19m^2$$

Each square meter of solar panel area produces:

$$production_per_m2 = 0.2 \cdot 960 = 192kWh$$

With an associated prediction error of:

$$error_per_m2 = 0.2 \cdot 21 = 4.2kWh$$

The expected loss due to inaccurate predictions is:

$$total_error = 19 \cdot 4.2 = 79.8kWh$$

This amount is roughly equivalent to the power consumption of the household over approximately 8 days².

5.5.2. Time and Location Invariance

The proposed solution faces two significant challenges: it lacks invariance to both time and location. The time invariance issue could potentially be addressed by employing transfer learning, where two additional temporal features (start date and end date) are introduced in the final layers of the network, just after the decoder. The location invariance problem could be tackled using two possible approaches:

- **Transfer learning:** This approach would involve adding two input features representing normalized longitude and latitude to the final layer. The network should then be retrained on new samples that include simulations based on multiple locations.
- **Predict daylight coefficients:** Instead of predicting the final irradiation outcome directly, the model could predict 145 daylight coefficients for each sensor point. These coefficients could then be multiplied by a sky matrix specific to a given location to compute the final irradiation values.

²In this example, an average prediction error is assumed. As described, prediction errors can sometimes be higher. For reference: the results of this thesis have shown that in 2.7% of the cases, the sensor point specific irradiation [RMSE](#) is higher than 50 kWh/m².

5.5.3. Improving Mode Coverage

As demonstrated, some predictions by the proposed network result in significant irradiation errors, indicating poor mode coverage—where the network is only able to make accurate predictions for specific types of samples. Improving the network’s ability to predict all cases accurately could be approached from two perspectives: enhancing the network and improving the dataset.

- **Attention based networks:** The analysis in this thesis indicates that further deepening the proposed network does not yield significant performance gains. However, exploring Transformer networks, which leverage attention mechanisms, could potentially improve predictions on the dataset, as has been demonstrated with point cloud segmentation. Specifically, attention-based models might better recognize long-distance relational patterns between geometry, such as a tall tower casting a shadow on a distant building.
- **Urban typology clustering:** This approach assumes that specific geometric features lead to higher irradiation errors, often due to the absence of similar examples in the training dataset. In the network proposed in this thesis, sample features are compressed into vectors of shape (1, 39, 3) for the point coordinates and (1, 512, 39) for other features within the latent space. By applying unsupervised clustering algorithms and dimensionality reduction techniques, samples can be categorized into groups with overlapping properties. A relationship between irradiation error and these groups might then be identified. The dataset could be improved by synthesizing more geometric samples within groups associated with high irradiation errors (oversampling) or reducing the size of common groups (undersampling), leading to better irradiation predictions for uncommon geometric types.

Urban typology clustering could prove beneficial not only for irradiation prediction but also for other building physics prediction models. Understanding how urban typology impacts prediction accuracy can help determine when a simulation is necessary, particularly in cases where the probability of incorrect predictions is high.

5.5.4. Geometric Level of Detail

The model developed in this thesis demonstrates the ability to predict irradiation on low-detail buildings. However, this capability does not necessarily extend to more detailed geometry, such as LoD 2.2 in the 3D BAG dataset. Further research is required to determine whether the model can effectively learn and predict irradiation on more complex datasets with higher geometric detail.

5.5.5. Further Research

For model optimization, the following research steps are suggested to achieve production-level accuracy and usability in solar irradiation prediction:

1. Investigate the accuracy on higher LoDs geometry, specifically using the Poisson Disk point sampling method.
2. Experiment with Transformer networks for irradiation prediction.
3. Explore the relationship between urban typology and prediction accuracy through unsupervised clustering and dimensionality reduction techniques.

5. Discussion

4. Apply transfer learning to adjust the final layers of the network, addressing issues related to location and time invariance.

5.6. Interaction

The client-server process proposed for the interaction model offers an efficient solution with minimal overhead. Empirical evidence indicates that this approach significantly reduces processing time compared to other *subprocess* methods.

5.6.1. External Hardware

One key advantage is that the server code does not need to run on the same device as the client. Although in this thesis, the server was operated on the same local machine as the client, it can also be run on external hardware, such as a supercomputer. This flexibility is beneficial not only for this specific project but for all deep learning projects in the architectural field. However, running the server externally is particularly relevant only when the model's inference time is substantial.

5.6.2. CPU Inference

Currently, the prediction model operates on a GPU. Future adjustments to the code, particularly focusing on the C++ implementations of the FPS and ball query algorithms, could enable the model to run on a CPU. This shift has the potential to lower inference time, making the system more versatile and accessible.

5.6.3. Implementation Other Design Software

Since the socket package utilizes a standard network communication protocol, developing the client in other programming languages is straightforward. The model's design features are transmitted as byte packages to the server, a process that can be easily implemented in other languages like C# or C++. Consequently, this approach is applicable to other design software, as long as the preprocessing algorithm can be executed within the respective program.

The current implementation of the interaction model in Grasshopper requires users to have a basic understanding of parametric design. In the future, a separate, web-based Graphical User Interface (GUI) could make the implementation even more user-friendly, broadening its accessibility and ease of use.

5.6.4. Optimization

This research has demonstrated that optimizing buildings for solar irradiation using brute-force algorithms is more feasible due to the reduced prediction time. However, while effective, this approach is not particularly sophisticated and could benefit from further enhancement through generative models. The primary challenge in this area lies in defining what constitutes "good" versus "bad" design in the context of solar irradiation.

Alternatively, the prediction model could be leveraged to assess more practical aspects of a design. As illustrated, this could pertain to evaluating the power potential of solar installations, assessing heat loads on windows, or reducing urban heat island effects. Generative models could then be employed to modify the design for specific objectives. For instance, adjusting the orientation of a building to minimize vertical heat loads could be an application of this approach. This would allow for more intelligent and purpose-driven optimization, moving beyond simple brute-force methods.

5.7. Hardware Improvements vs AI

Given that the performance of CPUs and GPUs have massively improved over the last decades, it could be argued that simulations will be able to reach the same speeds as the proposed deep learning model, without the risk of biased errors. Therefore, it could be stated that developing neural networks as replacement for simulation models is not a feasible approach for future building optimization.

However, it is important to understand the fundamental difference between simulation and AI prediction approaches. Whereas simulation models are able to compute certain output values, they do not have a deep understanding between the relationship between input and output. AI model on the other hand, are able to parameterize the problem and intrinsically learn how input changes affect the output. This deeper understanding could potentially be used in future generative models, which are not only able to predict, but also advice on changes based on earlier experience.

5.8. Future Research Questions

In general, the following future research questions are proposed to build upon the work of this thesis:

- How do urban typologies relate to errors in the prediction of solar irradiation?
- To what extent can deep neural networks predict solar irradiation on detailed urban models?
- How can solar irradiation be predicted and simulated on urban models that include dynamic objects such as trees?
- How can a dataset of observed irradiation values on urban geometry be developed?
- Can deep neural networks predict solar irradiation while accounting for time and location?

5. Discussion

For long-term research aimed at creating a more holistic model that integrates multiple aspects such as building physics, finance, construction, and user experience, the following steps are recommended:

- Further refine the work presented in this thesis to achieve production-ready accuracy³;
- Develop a model that clusters urban typologies, regardless of the building physics component being predicted;
- Use similar methodologies to explore predictions in other areas, such as wind patterns, acoustics, comfort, and financial implications;
- Transition from predictive models to generative models to optimize building geometry;
- Combine encoders used in prediction to create a holistic generative model for building design.

³The author recommends that the model should be able to predict annual irradiation with maximum errors smaller than the average power usage of a household over one day.

6. Conclusion

This research aimed to explore the feasibility of using deep neural networks to predict annual solar irradiation on 3D urban geometry. By developing a synthesized dataset derived from the 3D [BAC](#) and solar irradiation simulations, a PointNet-inspired neural network was successfully trained. The findings suggest that it is indeed possible to predict solar irradiation with an average [RMSE](#) of 21 kWh/m² on low-resolution buildings, offering a significantly faster alternative to traditional simulation models. The method proposed in this thesis has been integrated into a user-friendly ecosystem, enabling non-programmers to easily incorporate the tool into their design processes.

While previous research in solar irradiation prediction primarily focused on 2D projections, averaged irradiation values, and surrogate building geometry, this thesis advances the field by training the model on 3D geometry based on real buildings, targeting predictions at specific sensor points. Moreover, the research introduces methods for preprocessing geometric data for neural network training, optimizes computational efficiency, and presents an innovative server-client system for easier interaction with the model.

At its current stage, the proposed method can be applied for city-scale annual solar irradiation estimation and can be integrated into conceptual optimization processes that focus on solar heat performance metrics. However, it is crucial to continue evaluating the results with traditional simulation models, as the proposed solution may still have errors and biases.

Despite the promising results, several limitations need to be addressed before the method can be routinely applied in practice. Questions remain about the model's accuracy when applied to more detailed geometry, its handling of material properties, and the potential for making the model time- and location-invariant. Future research should explore whether Transformer-based networks and techniques such as dimensionality reduction and urban morphology clustering could help resolve these issues. Additionally, researchers could investigate whether the method proposed in this thesis could be extended to other building performance metrics.

The conversion of predictive models into generative models also presents an exciting avenue for research, potentially transforming how computers assist architects and designers in their work.

This research demonstrates that neural networks offer a valid and promising alternative to traditional building physics simulations. This direction merits further exploration to empower architects and designers in creating a more sustainable built environment.

7. Reflection

This chapter is a reflection on both the development and implications of this research. There will be a focus on academic relevance, societal impact, ethics and personal growth. Furthermore, AI tools which have been used for the development of this thesis, will be mentioned in section 7.3.

7.1. Academic Relevance

AI is a relatively new field of study within the architecture domain. Rapid advances are being made to implement deep neural networks in solving architecture and engineering-related problems. However, the lack of observed data is a widely known issue in the field. This research contributes to addressing this problem by proposing a method to train neural networks on synthesized irradiation based on real urban geometry.

Furthermore, this research prompts a reevaluation of the roles of designers and architects in the future. To what extent is it possible to replace certain tasks performed by architects, and how will the interaction with AI change their work in the future? This research suggests that AI models can, at the very least, make predictions about physics-related properties of urban designs. Further research in this field may determine whether it is possible for AI models to provide holistic advice on making buildings more sustainable or financially attractive.

7.2. Societal Impact

The publication of this research may directly impact the optimization strategies that architects and engineers use in daily practice. With the decreased computation time for solar performance, it becomes feasible to computationally optimize designs with significantly more iterations. This could result in a stronger focus on sustainability within the design field.

As advanced AI models become more prevalent in architecture, there may be a shift in focus to the social aspects of the profession. Since AI models can overtake repetitive tasks and provide advice on integral designs, architects may have more time to discuss options with their clients.

7.3. Ethics

Training deep neural networks requires significant amounts of energy. At the current stage of this research, a network would need to be developed for individual locations. Therefore, it is deemed essential that the networks become location-invariant to reduce the overall computation time.

7. Reflection

However, energy usage during the design phase is small compared to the energy consumed during construction, maintenance, and usage. Thus, the increased power usage due to model training is justified by the expected decrease in eventual energy consumption.

Although AI models are fast, they can also be incorrect. The recent rise of AI models introduces the risk of excessive human trust in these models. It is essential that building practitioners always verify the results and advice provided by AI models. As developer, transparency about potential errors and inaccuracies in the models' predictions can help improve trust and reliability.

For the development of this thesis, several AI frameworks have been used. ChatGPT 3, 3.5 Turbo and 4o have been used (OpenAI et al., 2023), advising and altering Python code, and used for the thesis grammar and sentence structure. By no means, AI has been used for the generation of text with new content and information. All text which was corrected by ChatGPT has been extensively checked by the author. For the presentation and cover of this thesis, Midjourney (Midjourney (V5), 2023) has been used for the generation of inspiring artwork.

7.4. From the Author: A Personal Reflection

Working on this thesis for so long has been quite a journey. Many years ago, this thesis began to take shape out of a frustration with having to use poorly optimized simulation models during my Bachelor's studies. Since it has taken so long to reach the final stage of this project, there was ample time to reflect on the role of AI in the field of architecture and design. The experience gained from this thesis has provided me with a stronger opinion in this debate and expert knowledge in the optimization domain, which I can apply to other fields.

The results of the project exceeded my expectations. Initially, I hoped for a model that would roughly suggest irradiation values on buildings, but I believe the model's performance is far more promising. It was gratifying to see positive results, especially since other academics were initially critical of the thesis's concept.

The scale of this thesis sometimes felt overwhelming, as many research fields were involved to achieve the final outcome. Although I knew it would be challenging to work in the Computer Vision domain as an architecture student, I may have underestimated the complexity of the code I was trying to manage. In retrospect, I would recommend future students focus only on dataset generation or neural network prediction instead of attempting both.

Due to the involvement and combination of various research fields, it was sometimes challenging to communicate my ideas and vision to my mentors. Each mentor had their own expertise, but lacked knowledge of the other domains, making it difficult to see the whole picture. Additionally, I would have appreciated more guidance in developing the code itself. Fortunately, many kind PhD students and students from other faculties were willing to have coffee and sit together to give me practical suggestions while reading through my code.

On the other hand, I am grateful to my mentors for pushing me to develop a stronger narrative around my thesis—why I am striving for this goal, how it would impact the field, and what more we could do in the future. I also appreciate that my mentors continued to guide me throughout this project, even when the final result was delayed due to external factors.

A. Regular Point Sampling Method

Neural networks generally perform better on regular data compared to irregular data due to the consistency and similarity in input structures that the network can recognize. This principle is relevant to the two data preprocessing techniques implemented in this thesis: irregular point clouds based on Poisson Disc sampling and regular sampling. This appendix details the procedure for generating a regular point grid on 3D BAG data samples.

A.1. 3D BAG Mesh Format

In the 3D BAG dataset, meshes are typically defined by faces per surface, where each rectangular surface is discretized into two triangular mesh faces, maintaining planarity. A straightforward approach to point sampling might involve using the centroids or vertices of these triangular faces. However, this method would result in an irregular point cloud with varying corresponding surface areas, as illustrated in figure A.2. Ideally, a procedure should be in place that generates a regular point grid with quad mesh faces that are square and have an area approximately equal to a user-defined parameter value (e.g., 1.0 m^2 in this context). Additionally, it is preferred that each mesh face corresponds to a sensor point, enabling the visualization of irradiation per face.

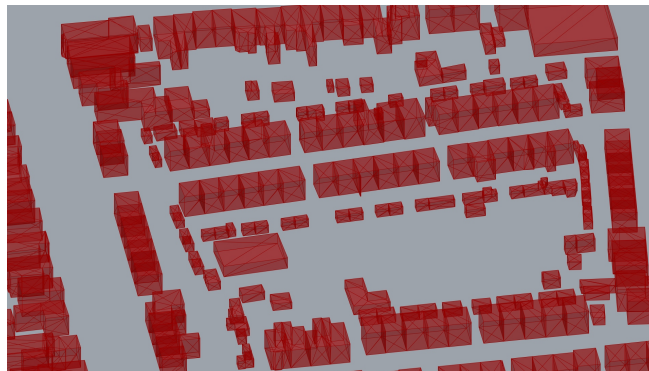


Figure A.1.: A 3D BAG mesh with triangular faces for each building surface. (Image by author)

A.2. Mesh Discretization

The *Rhino.geometry* package offers several techniques to discretize existing meshes into different configurations while maintaining the original shape. However, none of the provided solutions were suitable for this thesis. The main issues were either excessively long processing times or the

A. Regular Point Sampling Method

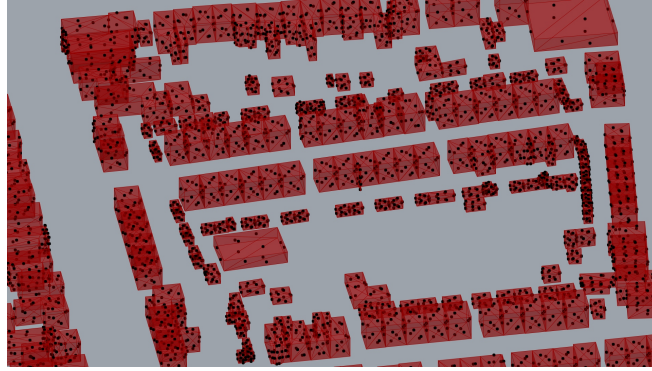


Figure A.2.: A sensor grid based on the centroids of the triangular faces. (Image by author)

generation of meshes that were incompatible with the requirements of this study. Consequently, this thesis proposes an alternative procedure described in this appendix. While all figures are visualized using McNeel Rhino, the steps outlined are fully implemented in Python to minimize visualization overhead and improve efficiency.

A.3. Sample Outline

An outline is created given a sample size (in this case 100m x 100m). All geometry within this outline will be considered for the simulation and prediction. The sample outline is a flat plane in the x,y direction. The z-direction indicates the height of the buildings.

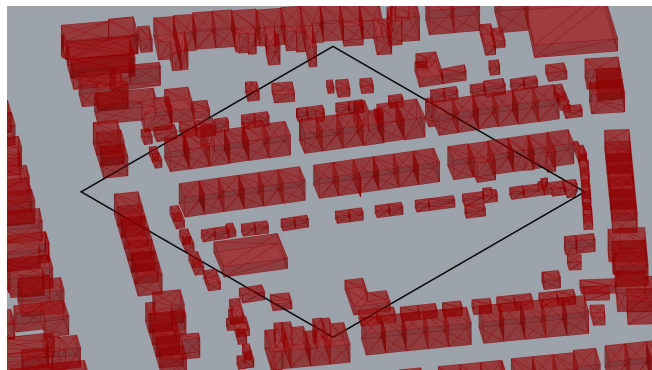


Figure A.3.: A sample outline of 100m x 100 m in which the building geometry will be extracted for a sample. (Image by author)

A.4. Building Component Extraction

The buildings are split in facades and roofs based on the normal directions. If the normal is equal to the vector $\{0,0,1\}$, it is assumed that the face belongs to a roof. Otherwise it is part of a façade.

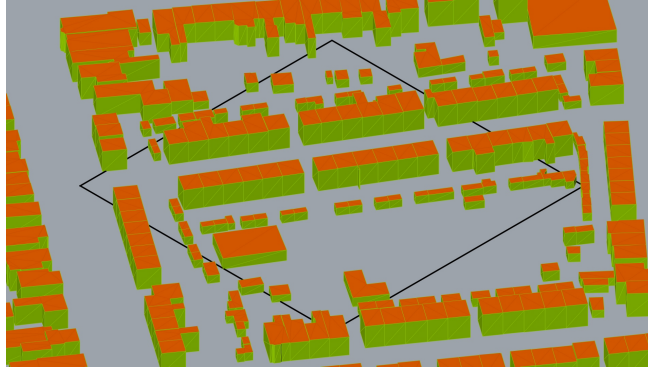


Figure A.4.: The extracted roofs (orange) and facades (green) from the mesh based on the normal directions. (Image by author)

A.5. Ground Levelling

All facades are leveled to the ground plane. The vertices with a z-value of 0, are used to extract outlines from the buildings. Next, all building outlines outside the sample outline are removed. If the building outline intersects with the sample outline, it is split and reconstructed to a closed polyline, within the sample boundary. In some special cases, buildings also have courtyards. A specifically designed algorithm makes a distinction between outer walls and inner courtyards.

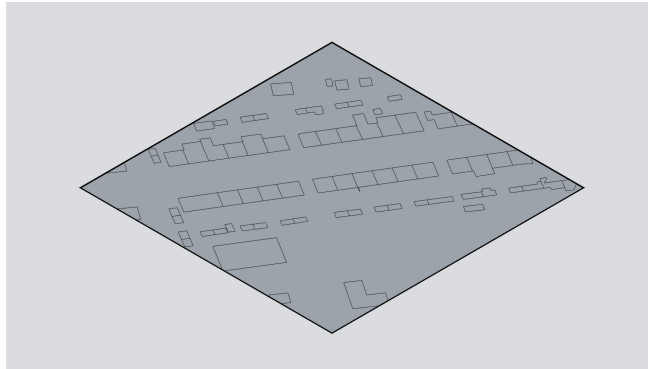


Figure A.5.: Based on the lowest z-values of the building facades, outlines are extracted. Outlines that cross the sample outline are reconstructed. (Image by author)

A.6. Mesh Face Quadrangulation

A quad mesh is generated from the sample outline, based on a given grid size (1.0m x 1.0m). Next, the building outlines are used to sequentially split the geometry. Courtyard outlines are used to re-split the extracted elements. These mesh faces are joined with the general ground mesh. Figure A.7 shows that triangular faces occur when the quad mesh is split. The density of the faces is higher and the area is lower than the preferred 1.0 square meter. Within the scope of this thesis, it was not deemed possible to implement an algorithm that would avoid the occurrence of triangle mesh faces at the splitting regions.

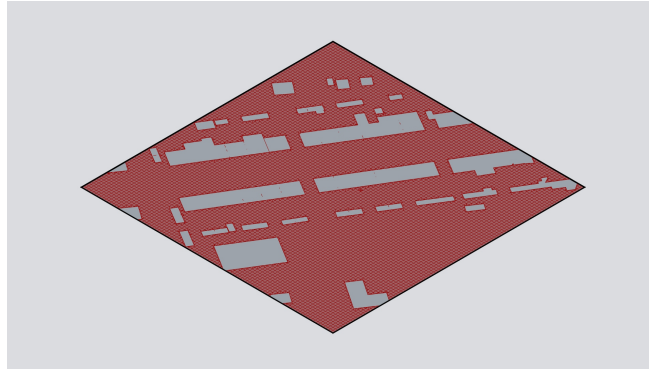


Figure A.6.: A ground regular ground mesh with mainly quad faces. (Image by author)

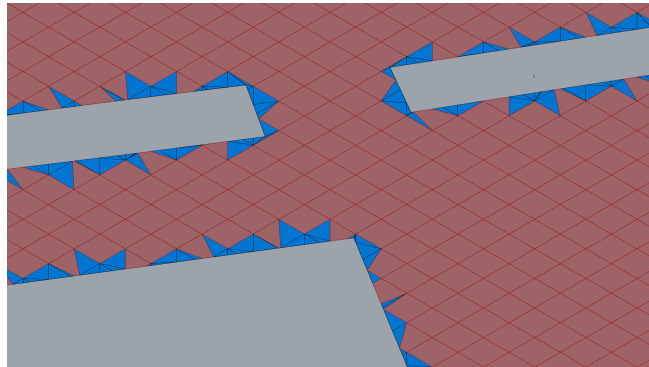


Figure A.7.: Triangular faces near the outlines of the buildings. (Image by author)

A.7. Roof Levelling

The extracted elements are moved to the height the building, based on the maximum z-value of the original building mesh vertices.

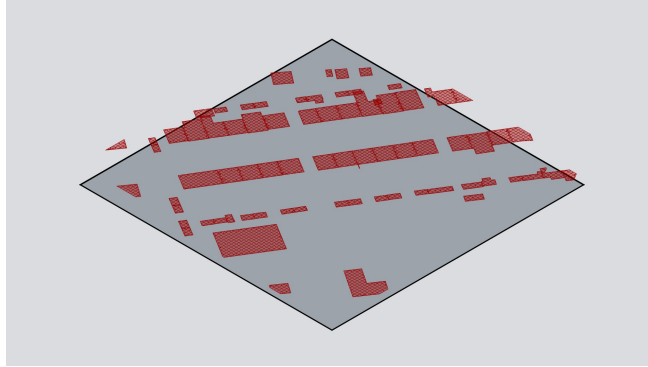


Figure A.8.: Splitted mesh elements are moved to the height of the building as roof. The height is found by taking the maximum z-value. (Image by author)

A.8. Facade Mesh Generation

For each building outline, including the corresponding courtyards, the line segments are split into points based on the grid size. Next, the points are moved upwards in the z-direction multiple levels, to reach the final height of the building, with intermediate steps approximately equal to the grid size. The points are used as vertices for a quad mesh, with faces of approximately one square meter.

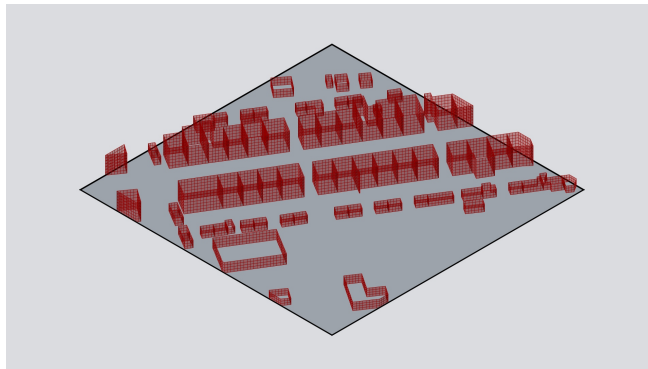


Figure A.9.: Building walls based on the outlines of the buildings (Image by author)

A.9. Combining Mesh Elements

Finally, the building elements and ground are merged into one non-manifold mesh. This mesh can be used for sensor point extraction and visualization purposes. By using the built-in functions in

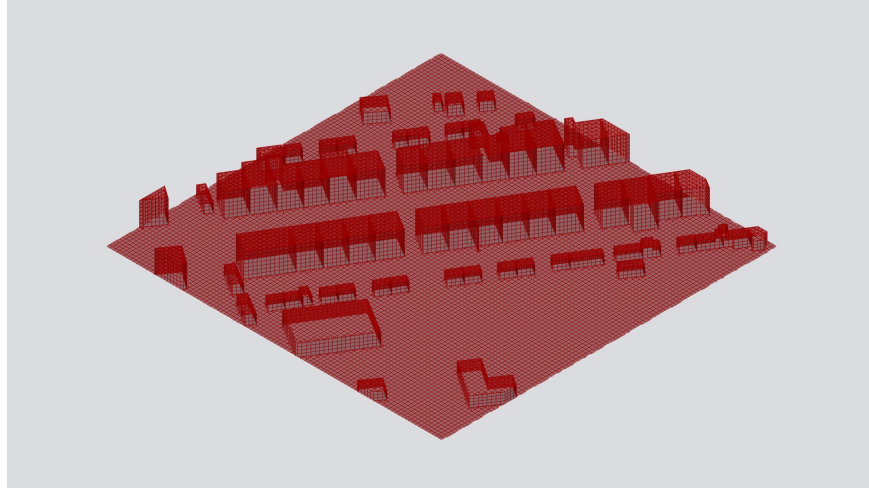


Figure A.10.: Roofs, walls and ground are merged into one non-manifold mesh. (Image by author)

the Rhino.geometry package, the constructed regular mesh is reshaped to a reduced rough mesh with less faces and vertices, which is used for the simulation procedure.

A.10. Regular Point Sampling

In the second part of the regular point sampling procedure, sensor points are extracted based on the face centroids of the mesh. These sensor points are offset based on the normal directions of the geometry (0.1m in this context). Some points however, belong to dividing walls which are not visible after simulation or prediction of irradiance. Therefore, a ray-tracing algorithm was implemented, to delete points that are deemed irrelevant for the prediction.

A.11. Dividing Wall Sensor Point Removal

For each sensor point, a ray is shot upwards in the z-direction. If the ray hits a roof from another building roof within half the grid-size, the mesh face is partially covered by the roof of another building. Otherwise, if the ray hits within a longer distance, the mesh face is fully covered vertically by another building.

The cross product is taken between the normal direction of the geometry and positive/negative z-direction. If the ray hits a wall from another building within half the grid-size, it is assumed that the mesh face is partially covered by another building. If it hits at a longer distance, it is fully covered by another building.

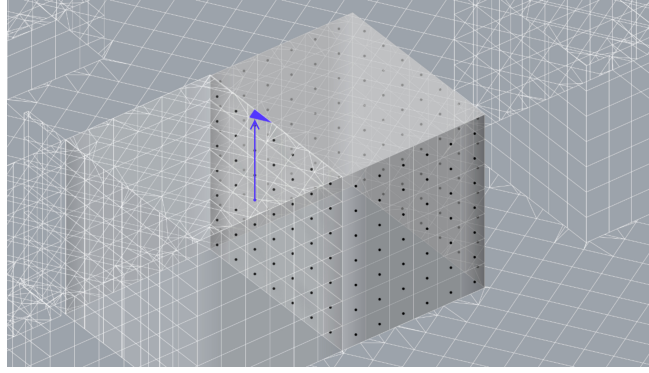


Figure A.11.: Given a sensor point (blue) a ray is shot upwards in the z-direction. Since it is hitting the blue face of another roof, it is covered by another building. (Image by author)

If none of the rays hit other geometry, the corresponding sensor point is kept. If one of the rays hits within half the grid size, it is also kept, but moved in the direction of the ray, plus the sensor offset distance. Finally, if all rays hit both another roof and wall, within a longer distance than half the grid size, it is fully covered by another building, thus the corresponding sensor point is removed.

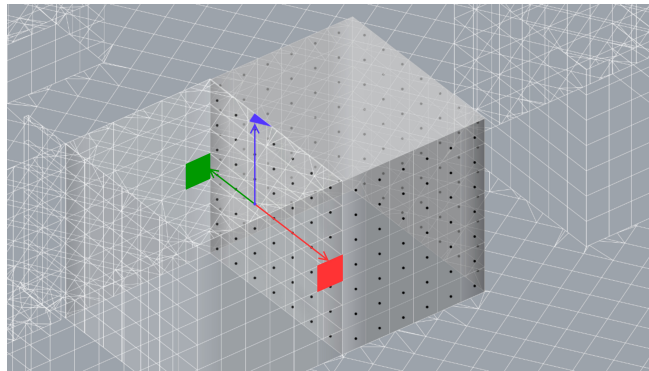


Figure A.12.: Rays are shot horizontally to conclude if the corresponding point's face is partially or fully covered by another facade. (Image by author)

This procedure is useful to limit the number of sensor points that have to be simulated/predicted. However, the time complexity of the algorithm is $O(n^2)$ considering the number of sensor points and buildings. Therefore, it is time consuming to remove dividing wall sensor points within the preprocessing phase. Future research could reduce the time complexity to make the algorithm more efficient.

For each generated samples, multiple checks are validated before keeping or deleting a sample. First of all, the **GSI** has to be higher than 0.1, which is based on the area of the building outlines. Secondly, the total area of the roofs and ground combined should be equal to the expected total area of the sample ($100 \times 100 = 10.000 \text{ m}^2$ in this context), plus a given tolerance. If this is not the case, it is assumed that the generation of the sample was incorrect. This usually happens due to incorrect assumptions regarding outlines being outer building polylines or courtyard outlines. Furthermore, errors regularly occur within the boolean splitting procedure of the ground mesh.

A. Regular Point Sampling Method

Further research is required to consider slanted roofs, overhangs and other more irregular geometry from higher levels of detail in the 3D [BAG](#). However, similar approaches as described earlier can be used.

B. Point Sampling Techniques

Alternative solutions to sensor points sampling exist, and may be combined with the generation of corresponding visualization meshes. The author suggests the following methods that may be considered for the purpose of this project.

1. **Regular point sampling** as discussed in appendix [A](#).

- **Advantages:** point clouds are an accurate representation of the geometric shape. Furthermore, point clouds generally have a regular grid. The procedure results in corresponding mesh faces that can be used for visualization purposes.
- **Disadvantages:** small triangular mesh faces occur next to building outlines. The procedure is not very efficient and errors occur relatively often. In the current stage of the research, the method is only applicable for low detail geometry
- **Solutions:** more accurate and efficient boolean intersection algorithms should be used. The ray tracing algorithm for sensor point cloud reduction (in relation to all buildings) may be optimized by using a binary tree approach. For more detailed geometry, handcrafted algorithms may handle slanted roofs and other building elements.

2. **Rough surface regular point sampling.** Instead of discretizing the meshes first, the original mesh from the 3D BAG is used. By generating a flat regular grid, points may be projected in the z-direction on horizontal surfaces. For the vertical elements, a similar approach can be used as described for option 1.

- **Advantages:** significantly more efficient and easier to implement.
- **Disadvantages:** results in coarser point clouds which are less descriptive for the geometric shape. It does not result in a mesh which can be used for visualization.

3. **Poisson Disc Sampling.** An approach which randomly samples points on the surface, with approximately equal distances, as described in the main report. This approach does not generate a usable mesh for visualization and will result in more irregular inputs to predict irradiation for.

- **Advantages:** efficient procedure. It can also be used for more irregular geometry;
- **Disadvantages:** there is no mesh which can be used for visualization directly.

4. **Poisson Disc Sampling with mesh reconstruction.** The point cloud is generated using the Poisson Disc Sampling method. The point cloud is then used to reconstruct a mesh for visualization in which each point corresponds to a mesh face.

- **Advantages:** includes an efficient point sampling strategy and mesh for visualization.
- **Disadvantages:** reconstructed mesh may not be the same as the original dataset.

B. Point Sampling Techniques

In this thesis, regular point sampling and Poisson Disc sampling have been proposed as potential solutions. Future research could show whether other methods such as described may perform similarly, better, or worse, also considering the irradiation prediction error.

Bibliography

- Alammar, A., Jabi, W., & Lannon, S. (2021). Predicting Incident Solar Radiation on Building's Envelope Using Machine Learning. *SimAUD 2021 Symposium on simulation for architecture + urban design*.
- Anderson, D., & Mcneill, G. (1992, August). *ARTIFICIAL NEURAL NETWORKS TECHNOLOGY A DACS State-of-the-Art Report* (tech. rep.). Rome Laboratory. New York.
- Arvo, J. (1986). Backward Ray Tracing. *Developments in Ray Tracing, SIGGRAPH '86 Course Notes, Volume 12*.
- As, I., & Basu, P. (2021). *The Routledge Companion to Artificial Intelligence in Architecture* (2021st ed., Vol. 1). Routledge.
- Bourgeois, D., Reinhart, C. F., & Ward, G. (2008). Standard daylight coefficient model for dynamic daylighting simulations. *Building Research and Information*, 36(1), 68–82. <https://doi.org/10.1080/09613210701446325>
- Brembilla, E., & Mardaljevic, J. (2019). Climate-Based Daylight Modelling for compliance verification: Benchmarking multiple state-of-the-art methods. *Building and Environment*, 158, 151–164. <https://doi.org/10.1016/j.buildenv.2019.04.051>
- Castro Pena, M. L., Carballal, A., Rodríguez-Fernández, N., Santos, I., & Romero, J. (2021). Artificial intelligence applied to conceptual design. A review of its use in architecture. *Automation in Construction*, 124, 103550. <https://doi.org/10.1016/j.autcon.2021.103550>
- Galanos, T., Chronis, A., & Vesely, O. (2024). City Intelligence Lab.
- Garbin, C., Zhu, X., & Marques, O. (2020). Dropout vs. batch normalization: an empirical study of their impact to deep learning. *Multimedia Tools and Applications*, 79(19–20), 12777–12815. <https://doi.org/10.1007/s11042-019-08453-9>
- Geisler-Moroder, D., Lee, E. S., & Ward, G. J. (2017). Validation of the Five-Phase Method for Simulating Complex Fenestration Systems with Radiance against Field Measurements. *15th International Conference of the International Building Performance Simulation Association*. <https://doi.org/10.26868/25222708.2017.401>
- Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative Adversarial Networks. *Communications of the ACM*, 63(11), 139–144. <https://doi.org/https://doi.org/10.1145/3422622>
- Han, J. M., Choi, E. S., & Malkawi, A. (2022). CoolVox: Advanced 3D convolutional neural network models for predicting solar radiation on building facades. *Building Simulation*, 15(5), 755–768. <https://doi.org/10.1007/s12273-021-0837-0>
- Hanocka, R., Hertz, A., Fish, N., Giryas, R., Fleishman, S., & Cohen-Or, D. (2019). MeshCNN: A Network with an Edge. *ACM Trans. Graph*, 1(1). <https://doi.org/10.1145/3306346.3322959>
- Heckbert, P. (1993). Finite Element Methods for Radiosity. *Proceedings of 20th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '93)*.
- Huang, C., Zhang, G., Yao, J., Wang, X., Calautit, J. K., Zhao, C., An, N., & Peng, X. (2022). Accelerated environmental performance-driven urban design with generative adversarial network. *Building and Environment*, 224, 109575. <https://doi.org/10.1016/j.buildenv.2022.109575>

Bibliography

- Ioffe, S., & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *ICML'15: Proceedings of the 32nd International Conference on International Conference on Machine Learning*, 448–456.
- Isola, P., Zhu, J.-Y., Zhou, T., & Efros, A. A. (2016). Image-to-Image Translation with Conditional Adversarial Networks.
- Jones, N. L., & Reinhart, C. F. (2017). Speedup Potential of Climate-Based Daylight Modelling on GPUs. *Proceedings of Building Simulation 2017: 25th Conference of IBPSA*, 975–984. <https://doi.org/10.26868/25222708.2017.259>
- Jones, N. L., & Reinhart, C. F. (2022). Validation of Gpu Lighting Simulation in Naturally And Artificially Lit Spaces. *Proceedings of Building Simulation 2015: 14th Conference of IBPSA*, 14. <https://doi.org/10.26868/25222708.2015.2461>
- Kharvari, F. (2020). An empirical validation of daylighting tools: Assessing radiance parameters and simulation settings in Ladybug and Honeybee against field measurements. *Solar Energy*, 207, 1021–1036. <https://doi.org/10.1016/j.solener.2020.07.054>
- Ladybug Tools. (2024a). Honeybee Primer. <https://docs.ladybug.tools/honeybee-primer>
- Ladybug Tools. (2024b). Ladybug Primer. <https://docs.ladybug.tools/ladybug-primer>
- Lila, A., Jabi, W., & Lannon, S. (2021). Predicting solar radiation with Artificial Neural Network based on urban geometrical classification. *Proceedings of Building Simulation 2021: 17th Conference of IBPSA*, 902–909. <https://doi.org/10.26868/25222708.2021.30796>
- Ma, L., Stückler, J., Kerl, C., & Cremers, D. (2017). Multi-View Deep Learning for Consistent Semantic Mapping with RGB-D Cameras. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.
- Mardaljevic, J., Heschong, L., & Lee, E. (2009). Daylight metrics and energy savings. *Lighting Research & Technology*, 41(3), 261–283. <https://doi.org/10.1177/1477153509339703>
- Maturana, D., & Scherer, S. (2015). VoxNet: A 3D Convolutional Neural Network for real-time object recognition. *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 922–928. <https://doi.org/10.1109/IROS.2015.7353481>
- Midjourney (V5). (2023). Midjourney. <https://www.midjourney.com/>
- M.Matter, N., & G.Gado, N. (2024). Artificial Intelligence in Architecture: Integration into Architectural Design Process. *Engineering Research Journal*, 181(0), 1–16. <https://doi.org/10.21608/erj.2024.344313>
- Mokhtar, S., Beveridge, M., Cao, Y., Drori, I., Balasubramanian, V. N., & Tsang, I. (2021). Pedestrian Wind Factor Estimation in Complex Urban Environments. *Proceedings of The 13th Asian Conference on Machine Learning, PMLR 157*, 157, 486–501. <https://doi.org/10.48550/arXiv.2110.02443>
- Nakhaee, A., & Paydar, A. (2023). DeepRadiation: An intelligent augmented reality platform for predicting urban energy performance just through 360 panoramic streetscape images utilizing various deep learning models. *Building Simulation*, 16(3), 499–510. <https://doi.org/10.1007/s12273-022-0953-5>
- OpenAI, Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., Avila, R., Babuschkin, I., Balaji, S., Balcom, V., Baltescu, P., Bao, H., Bavarian, M., Belgum, J., ... Zoph, B. (2023). GPT-4 Technical Report.
- OpenStreetMap contributors. (2017). Planet dump retrieved from <https://planet.osm.org>.
- Peters, R., Dukai, B., Vitalis, S., van Liempt, J., & Stoter, J. (2022). Automated 3D Reconstruction of LoD2 and LoD1 Models for All 10 Million Buildings of the Netherlands. *Photogrammetric Engineering & Remote Sensing*, 88(3), 165–170. <https://doi.org/10.14358/PERS.21-00032R2>
- Ploennigs, J., & Berger, M. (2023). AI art in architecture. *AI in Civil Engineering*, 2(1), 8. <https://doi.org/10.1007/s43503-023-00018-y>

- Qi, C. R., Su, H., Mo, K., & Guibas, L. J. (2016). PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 652–660.
- Qi, C. R., Yi, L., Su, H., & Guibas, L. J. (2017). PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. *NIPS'17: Proceedings of the 31st International Conference on Neural Information Processing Systems*, 5105–5114.
- Qian, G., Li, Y., Peng, H., Mai, J., Hammoud, H. A. A. K., Elhoseiny, M., & Ghanem, B. (2022). PointNeXt: Revisiting PointNet++ with Improved Training and Scaling Strategies. *36th Conference on Neural Information Processing Systems (NeurIPS 2022)*. <https://doi.org/10.48550/arXiv.2206.04670>
- Reinhart, C. F., & Walkenhorst, O. (2001). Validation of dynamic RADIANCE-based daylight simulations for a test office with external blinds. *Energy and Buildings*, 33(7), 683–697. [https://doi.org/10.1016/S0378-7788\(01\)00058-5](https://doi.org/10.1016/S0378-7788(01)00058-5)
- Riegler, G., Ulusoy, A. O., & Geiger, A. (2016). OctNet: Learning Deep 3D Representations at High Resolutions. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 6620–6629.
- Robinson, D., & Stone, A. (2004). Irradiation modelling made simple: the cumulative sky approach and its applications. *Plea2004 - The 21st Conference on Passive and Low Energy Architecture*, 19–22.
- Rombach, R., Blattmann, A., Lorenz, D., Esser, P., & Ommer, B. (2021). High-Resolution Image Synthesis with Latent Diffusion Models. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 10674–10685. <https://doi.org/10.1109/CVPR52688.2022.01042>
- Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2015*, 234–241. <https://doi.org/10.48550/arXiv.1505.04597>
- Stevens, E., Antiga, L., & Viehmann, T. (2020). *Deep Learning with PyTorch*. Manning Publications Co.
- Subramaniam, S. (2017, October). *Daylighting Simulations with Radiance using Matrix-based Methods* (tech. rep.). <https://unmethours.com>
- Tehrani, A. A., Veisi, O., Fakhr, B. V., & Du, D. (2024). Predicting solar radiation in the urban area: A data-driven analysis for sustainable city planning using artificial neural networking. *Sustainable Cities and Society*, 100, 105042. <https://doi.org/10.1016/j.scs.2023.105042>
- Tregenza, P. R., & Waters, I. M. (1983). Daylight coefficients. *Lighting Research & Technology*, 15(2), 65–71. <https://doi.org/10.1177/096032718301500201>
- Tsangrassoulis, A., & Bourdakis, V. (2003). Comparison of radiosity and ray-tracing techniques with a practical design procedure for the prediction of daylight levels in atria. *Renewable Energy*, 28(13), 2157–2162. [https://doi.org/10.1016/S0960-1481\(03\)00078-8](https://doi.org/10.1016/S0960-1481(03)00078-8)
- Vahdat, A., & Kreis, K. (2022, April). Improving Diffusion Models as an Alternative To GANs, Part 1. <https://developer.nvidia.com/blog/improving-diffusion-models-as-an-alternative-to-gans-part-1/>
- Vecchio, G., Prezzavento, L., Pino, C., Rundo, F., Palazzo, S., & Spampinato, C. (2023). MeT: A graph transformer for semantic segmentation of 3D meshes. *Computer Vision and Image Understanding*, 235, 103773. <https://doi.org/10.1016/j.cviu.2023.103773>
- Wang, P. S., Liu, Y., Guo, Y. X., Sun, C. Y., & Tong, X. (2017). O-CNN: Octree-based convolutional neural networks for 3D shape analysis. *ACM Transactions on Graphics*, 36(4). <https://doi.org/10.1145/3072959.3073608>
- Wang, P. S., Liu, Y., & Tong, X. (2020). Deep Octree-based CNNs with Output-Guided Skip Connections for 3D Shape and Scene Completion. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 266–267.

Bibliography

- Ward, G., & Shakespeare, R. (2011). *Rendering with Radiance, The Art and Science of Lighting Visualizations*. Randolph M. Fritz.
- Ward, G. J., Bueno, B., Geisler-Moroder, D., Grobe, L. O., Jonsson, J. C., Lee, E. S., Wang, T., & Rose Wilson, H. (2022). Daylight simulation workflows incorporating measured bidirectional scattering distribution functions. *Energy and Buildings*, 259, 111890. <https://doi.org/10.1016/j.enbuild.2022.111890>
- Wu, W., Fu, X.-M., Tang, R., Wang, Y., Qi, Y.-H., & Liu, L. (2019). Data-driven interior plan generation for residential buildings. *ACM Transactions on Graphics*, 38(6), 1–12. <https://doi.org/10.1145/3355089.3356556>
- Yue, Y., Yan, Z., Ni, P., Lei, F., & Qin, G. (2024). Promoting solar energy utilization: Prediction, analysis and evaluation of solar radiation on building surfaces at city scale. *Energy and Buildings*, 319, 114561. <https://doi.org/10.1016/j.enbuild.2024.114561>
- Yuksel, C. (2015). Sample Elimination for Generating Poisson Disk Sample Sets. *Computer Graphics Forum*, 34(2), 25–32. <https://doi.org/10.1111/cgf.12538>
- Zhang, Y., Schlueter, A., & Waibel, C. (2023). SolarGAN: Synthetic annual solar irradiance time series on urban building facades via Deep Generative Networks. *Energy and AI*, 12, 100223. <https://doi.org/10.1016/j.egyai.2022.100223>
- Zhao, H., Jiang, L., Jia, J., Torr, P., & Koltun, V. (2021). Point Transformer. *IEEE Access*, 9, 134826–134840. <https://doi.org/doi:10.1109/ACCESS.2021.3116304>
- Zhu, J.-Y., Park, T., Isola, P., & Efros, A. A. (2017). Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. *2017 IEEE International Conference on Computer Vision (ICCV)*, 2242–2251. <https://doi.org/10.1109/ICCV.2017.244>

Colophon

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

