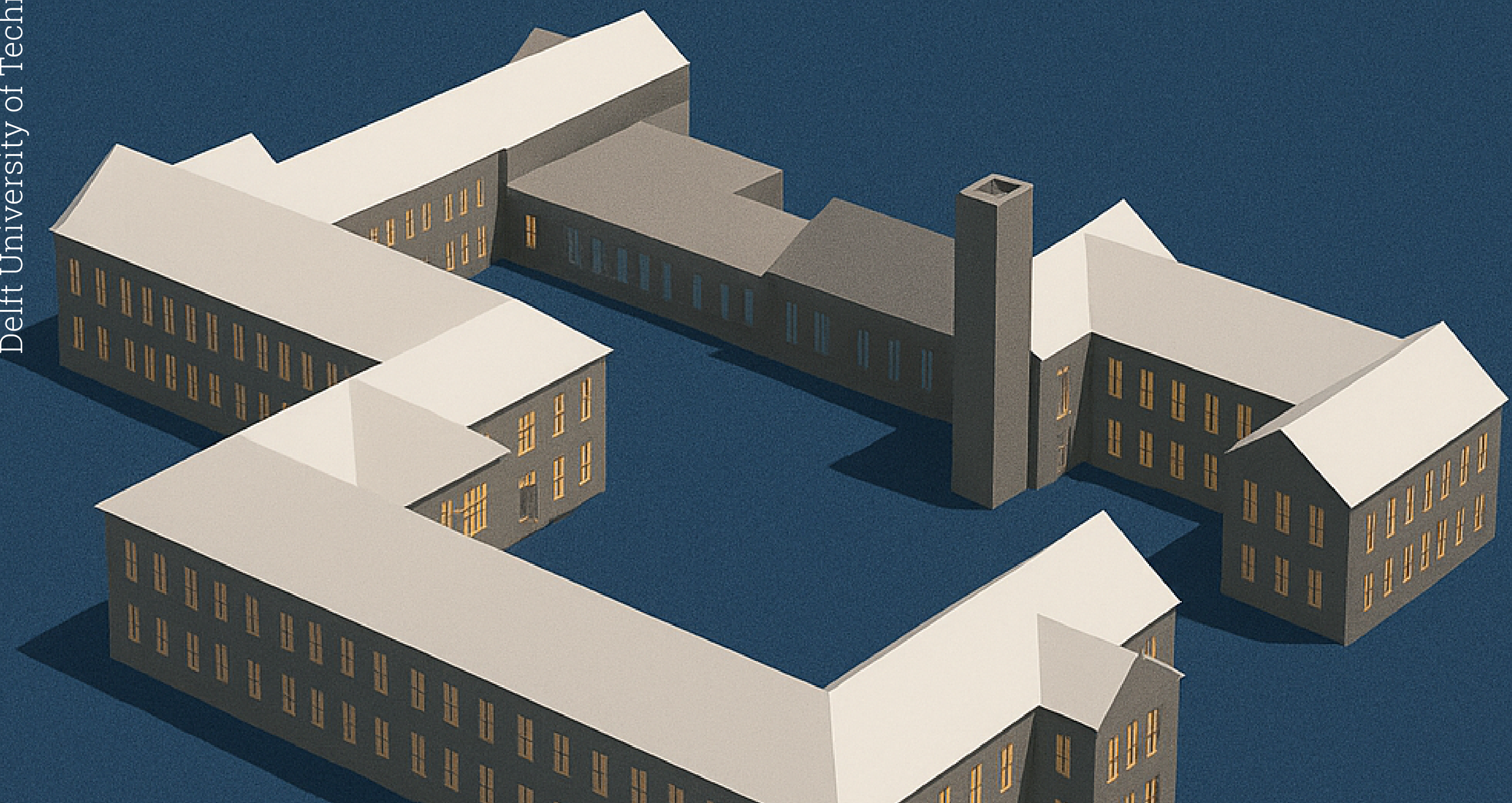
Geospatial Analytics from IoT ecosystem in Built spaces

Simulating Room Temperature Using XGBoost Model

GEO20202: Thesis Report

Vidushi Bhatt

Delft University of Technology



TU Delft

Geospatial Analytics from IoT ecosystem in Built spaces

Developing a System for Indoor Temperature Prediction Using OGC, IFC Semantics and XGBoost

by

Vidushi Bhatt

Student Number
5862124

First Supervisor:	Dr. Azaraksha Rafiee
Second Supervisor:	Justin Schembri
Project Duration:	November, 2024 – July, 2025
Faculty of Architecture the Built Environment, TU Delft	

TU Delft

Preface

During the master's programme Geomatics for the Built Environment at TU Delft, I was introduced to a wide spectrum of geospatial technologies and their applications in urban contexts. The curriculum established GIS and spatial analysis techniques along with some exposure to frontier technologies such as machine learning, real-time sensor networks, and advanced 3D visualization.

As these technologies shape the future of smart cities and data-driven decision-making, I was intrigued by a fundamental question: could similar methods be used to understand and predict aspects of indoor built environment? More specifically, I wondered if parameters such as indoor temperature or air quality could be forecasted using a combination of real-time sensor readings and spatial information embedded in building models.

This thesis represents the outcome of that exploration. It brings together multiple domains including IoT systems, GeoWeb, data analytics, BIM modeling, and machine learning, to investigate how built environment characteristics can interact with live sensor data to support simulation and prediction of indoor environmental conditions.

The journey of completing this research has been both technically challenging and engaging. Exploring how server-backed front-end applications can be integrated with machine learning mechanisms (despite starting with only a foundational understanding of coding and Machine Learning (ML)) was deeply rewarding. It helped me appreciate the capabilities of open-source tools available today and made me realize how much is possible even without deep expertise in machine learning theory. While a strong foundation in core ML theory is substantially valuable, its absence does not have to be an inhibitor for the development of ML-powered tools in geospatial analytics.

While I am aware that this field is evolving rapidly, I hope that the work presented here contributes in a small but meaningful way to the broader dialogue on smart buildings, environmental sensing, and predictive modeling for sustainable urban development.

Vidushi Bhatt, October 2025

List of Abbreviations

API	Application Programming Interface
AR	Augmented Reality
BIM	Building Information Modeling
CBDM	Climate-Based Daylight Modeling
CESIUM	CesiumJS (WebGL-based 3D visualization library)
CNN	Convolutional Neural Network
CSV	Comma-Separated Values
DA	Daylight Autonomy
DHI	Diffuse Horizontal Irradiance
DNI	Direct Normal Irradiance
EPBD	Energy Performance of Buildings Directive
EPC	Energy Performance Certificate
EU	European Union
FROST	Fraunhofer Open Source SensorThings
GH	Grasshopper (Visual Programming Platform for Rhino)
GIS	Geographic Information System
GHI	Global Horizontal Irradiance
glTF	GL Transmission Format
GUI	Graphical User Interface
GWR	Geographically Weighted Regression
IFC	Industry Foundation Classes
IoT	Internet of Things
ISO	International Organization for Standardization
LOD	Level of Detail
LSTM	Long Short-Term Memory
ML	Machine Learning
NREL	National Renewable Energy Laboratory
OBJ	Object File Format (for 3D models)
OGC	Open Geospatial Consortium

ReLU	Rectified Linear Unit
RMSE	Root Mean Square Error
RNN	Recurrent Neural Network
SBI	Solar Beam Irradiance
SDI	Spatial Data Infrastructure
SHGC	Solar Heat Gain Coefficient
SODA	Solar Radiation Data from Meteotest (Linke Turbidity DB)
SPA	Solar Position Algorithm
SRI	Smart Readiness Indicator
SVM	Support Vector Machine
sDA	Spatial Daylight Autonomy
UI	User Interface
UDI	Useful Daylight Illuminance
XGB	XGBoost (Extreme Gradient Boosting)
STA	SensorThings API
MAE	Mean Absolute Error
MSE	Mean Squared Error
R^2	Coefficient of Determination
PostGIS	Spatial Extension for PostgreSQL
IFC2X3	Industry Foundation Classes Schema Version 2X3
IFC4	Industry Foundation Classes Schema Version 4
EPSG	European Petroleum Survey Group (Coordinate Reference System Identifier)
CRS	Coordinate Reference System
RD New	Rijksdriehoekscoördinaten New (Dutch National Grid)
UTC	Coordinated Universal Time
JSON	JavaScript Object Notation
HTTP	Hypertext Transfer Protocol
URL	Uniform Resource Locator
QGIS	Quantum Geographic Information System
PVLib	Python Photovoltaic Library for Solar Analysis

Contents

List of Figures

1 Introduction

Buildings play a critical role in energy consumption, carbon emissions, and occupant well being, making the ability to monitor and predict their indoor environmental conditions increasingly important. In this context, the intention of this thesis is to develop an interoperable system for room level prediction of indoor temperature, using open standards and lightweight computational tools. This work demonstrates a standards based approach for forecasting indoor temperature in uninstrumented spaces by integrating BIM semantics encoded in IFC, the OGC SensorThings API (STA) for IoT interoperability, and a computationally efficient machine learning framework. The emphasis is on leveraging open-source implementations and lightweight ML methods, such as XGBoost, that can be incrementally retrained and readily deployed in real-world contexts.

1.1. Background and Motivation

The drive toward sustainable and resilient buildings has intensified due to their substantial contribution to global energy consumption and greenhouse gas emissions. Beyond environmental considerations, maintaining comfortable and healthy indoor conditions is central to human well being and productivity. However, achieving this balance requires accurate knowledge of indoor environmental dynamics, which are shaped by the interplay of outdoor climate, building characteristics, and occupant behaviour.

Traditionally, two main approaches have been used to understand these dynamics: dense sensor networks for real-time monitoring, and physics based simulations for predictive analysis. Both approaches face practical constraints. Installing and maintaining sensors in every room is costly and intrusive, while physics based simulations require strong assumptions and are computationally expensive for real time applications. These limitations create knowledge gaps in many uninstrumented spaces, constraining data-driven strategies for building management.

Emerging digital infrastructures offer pathways to overcome these challenges. IoT frameworks now enable continuous and standardized data exchange from distributed sensors. BIM encodes geometric and semantic attributes of buildings, such as room volume, orientation, and window placement, providing spatial context that complements sensor observations. Machine learning (ML) methods add a further dimension by learning directly from observed data, capturing nonlinear relationships without relying on simplified physical assumptions. In particular, lightweight ensemble approaches such as XGBoost are well-suited to structured datasets typical at the building level, offering speed, interpretability, and support for incremental retraining.

The motivation for this thesis lies in bridging these complementary domains through an open and interoperable prototype. By combining IFC-based BIM semantics, IoT data managed through the OGC SensorThings API and FROST server, and an XGBoost-based predictive pipeline, this work demonstrates how uninstrumented building spaces can be modeled with readily deployable tools. The emphasis on standards-based integration and lightweight ML reflects not only the technical feasibility but also the practical scalability of the proposed approach for smart building applications.

1.1.1. Introduction to Components: IoT and Geospatial

The Internet of Things (IoT) refers to a network of interconnected physical devices including sensors, actuators, micro controllers, edge computing units, and wireless communication modules that work together to collect, transmit, and sometimes locally process data from the physical world. These embedded systems enable real-time monitoring and control by forming a feedback loop between physical phenomena and digital systems. While sensors gather data such as temperature, humidity, or light intensity, actuators perform actions like adjusting ventilation or controlling lighting based on this data. Microcontrollers and embedded processors manage local decision making and communication, often operating at the edge to reduce latency and offload computation from centralized servers. In the context of buildings, these IoT components are commonly used to monitor indoor environmental conditions such as temperature, humidity, CO_2 levels, noise, and pressure. Such parameters are crucial for maintaining energy efficiency, occupant comfort, and safety.

Geospatial technologies deal with data that has an inherent spatial or locational dimension data that can be mapped, analyzed, or interpreted in terms of where it occurs. These technologies encompass a wide range of tools and systems, such as Geographic Information Systems (GIS), spatial databases, remote sensing platforms, and visualization environments. Among these, GIS plays a central role as both a framework and a platform for managing spatial information. More importantly, GIS can act as a connector between disparate data sources. For instance, when environmental sensor data (e.g., temperature, humidity) is tagged with spatial coordinates or room-level identifiers, GIS provides the spatial context needed to integrate that data with digital building models, infrastructure layouts, or meteorological datasets. This capability transforms otherwise siloed datasets into actionable spatial knowledge, enabling predictive modeling, resource optimization, and real-time monitoring across domains such as urban planning, building management, and environmental sensing [26, 22]. However, the utility of GIS for building level applications is further enhanced when combined with Building Information Modelling (BIM). Whereas GIS offers the broader spatial context, BIM supplies the fine grained detail of building components and their semantic relationships, allowing multi-scale integration of data.

BIM is a process of creating and managing digital representations of the physical and functional characteristics of built assets throughout their lifecycle [56]. These representations include geometry (3D shape), relationships among building components, metadata such as material properties, construction specifications, and operational features [7, 9]. When enriched with locational context such as geographic coordinates, spatial relationships among rooms, orientation, window placement, and room volume, BIM semantics can become a powerful contributor to a geospatial ecosystem. In this sense, BIM provides not only what a building is composed of, but also where and how its elements relate to external context: for example, which rooms receive sunlight at what angles, which walls face north vs. south, or how airflow might be affected by urban surroundings. Such locational aspects enable BIM-based data to be fused with other sources (e.g., sensor networks, external climate data) for spatially-aware analytics and prediction.

Together, these domains highlight a layered integration of technologies. Sensors provide real-time observations of environmental conditions, which are managed through IoT infrastructures for collection, transmission, and interaction. BIM contributes semantic and structural detail of buildings and their elements, allowing integration with sensor data to generate a spatial framework that reveals context and relationships. Combined, these capabilities enable a holistic view of the built environment, where operational data, structural semantics, and spatial context reinforce one another to support advanced analysis, predictive modeling, and decision making.

1.1.2. Integration of IoT, GIS and BIM

The integration of IoT and geospatial technologies is emerging as a transformative approach in smart building and urban analytics. By linking sensor data to specific physical locations and architectural components, one can enable spatially informed insights and decisions. This combined approach supports a variety of use cases from energy efficient building operations and predictive maintenance to emergency response, occupancy monitoring, and indoor air quality control; For example, associating indoor air quality data with BIM based room geometry can allow facility managers to localize problems and take targeted actions. When sensor readings are spatially visualized and temporally tracked, they can expose hidden patterns such as heat retention in poorly ventilated rooms or correlations between sunlight exposure and CO2 buildup.

A compelling example is provided by the University of Cagliari study on BIM and IoT Sensors Integration [16]. The researchers integrated real-time sensor data into a BIM environment and visualized indoor conditions to flag potential issues such as overheating, fire risk, or poor air quality. This highlights the power of IoT–BIM frameworks not only for monitoring and alerts, but also for visualisation and for setting up control mechanisms. The significance of such integration becomes broader when examined in the context of smart city frameworks and their goals for sustainable urban development. As cities increasingly aim to become more energy efficient, resilient, and responsive to the needs of their inhabitants, integrating data-driven monitoring systems with spatial models like location aware BIM enables more precise, localized, and adaptive decision making. These technologies can support intelligent building management, inform urban policy, and ultimately contribute to reducing environmental impact at both the built and urban scale.

1.2. Problem Statement

The growing availability of indoor sensors has enabled continuous monitoring of parameters such as temperature, noise, humidity, and CO_2 concentration. However, deploying sensors in every room of a building is costly, intrusive, and often impractical. As a result, many building spaces remain uninstrumented, leaving gaps in knowledge about their environmental conditions. This limitation constrains the ability of facility managers and researchers to develop holistic, data-driven strategies for energy efficiency and occupant comfort.

At the same time, Building Information Modelling (BIM), particularly through the Industry Foundation Classes (IFC) standard, provides detailed semantic and geometric representations of building spaces, including room volume, orientation, and window characteristics. Although BIM holds potential for augmenting sensor based monitoring, its integration with real time observations remains limited in practice. Most existing systems rely on proprietary software or ad-hoc connections between sensors and building models, restricting interoperability and scalability.

Machine learning (ML) techniques, especially ensemble methods such as XGBoost, offer the ability to learn from historical sensor data and predict future environmental conditions. Yet, current applications at the building scale often emphasize energy demand forecasting or comfort optimization, with fewer studies addressing room level prediction of conditions in non-instrumented spaces. Table 2.1 presents a comparative analysis of case studies that integrate IoT, BIM, and predictive modeling, highlighting their respective focus areas, methodological choices, and limitations in relation to the objectives of this thesis.

The problem addressed in this thesis therefore lies in the absence of an open, standards-based framework that combines IoT observations, BIM semantics, and machine learning to predict indoor environmental conditions in uninstrumented spaces. Specifically, this work investigates how the OGC SensorThings API (STA) can be employed for interoperable sensor data exchange, how IFC-based BIM models can supply spatial and semantic attributes, and how these inputs can be integrated into an ML pipeline to predict indoor temperature in rooms without sensors. The development and evaluation of such a prototype form the central challenge of this thesis.

1.3. ML as an Approach for Predictive Modelling

Predicting indoor environmental conditions has traditionally relied on physics-based and statistical methods. Physics-based simulation tools such as EnergyPlus and TRNSYS explicitly model heat transfer, solar radiation, material properties, and HVAC dynamics to estimate indoor conditions [15, 55]. While these approaches are grounded in thermodynamic principles and provide detailed control over physical parameters, they are computationally intensive and demand precise input data that may not always be available. Moreover, their reliance on assumptions regarding occupant behavior and HVAC operation often leads to discrepancies between simulated and actual conditions [29].

Statistical methods, such as autoregressive integrated moving average (ARIMA) models and linear regression, have also been applied for short-term indoor temperature forecasting [46]. These approaches are lightweight and interpretable but are limited in their ability to handle nonlinear interactions among variables, such as the combined effects of solar radiation, room geometry, and outdoor climate. As buildings represent highly dynamic systems where environmental factors interact in complex and often nonlinear ways, purely statistical models tend to underperform in real-world applications [43].

Machine learning (ML) has emerged as a compelling alternative for predictive modeling in built environments. Unlike purely physics-based or statistical approaches, ML can learn directly from observed data, capturing nonlinear relationships and interactions between features without requiring explicit physical formulations. Studies demonstrate the suitability of ML for indoor temperature forecasting, energy demand prediction, and comfort modeling, showing superior performance compared to conventional statistical baselines. Paul et al. [48] evaluated multiple algorithms, including Random Forests, Support Vector Machines, and Neural Networks, for room-level indoor temperature prediction, providing strong evidence of ML's suitability in this exact context. Similarly, Allam, Kassem, and Elagouz [4] implemented a lightweight cloud-based IoT framework that employed linear regression and basic ML techniques for forecasting temperature and humidity, demonstrating feasibility even in resource constrained setups. Related applications extend this evidence base: Imran, Iqbal, and Kim [35] applied predictive optimization to IoT-driven task scheduling for reducing residential energy consumption, while Xin et al. [61] proposed a CNN–LSTM hybrid model for spatio-temporal energy demand forecasting at city scale. Complementing these case studies, review articles such as Liu et al. [43] and Ahmad, Chen, and Guo [1] emphasize that ML approaches consistently outperform traditional statistical methods in building energy and environmental prediction tasks, underlining their growing importance for short term indoor environmental forecasts. Algorithms such as Support Vector Machines (SVM), Random Forests, and Artificial Neural Networks (ANN) have been applied to building-level datasets with promising results, particularly when sensor data are abundant and high-resolution [48, 4].

Among ML methods, ensemble-based approaches have shown consistent advantages for structured data prediction. Random Forests provide robustness by averaging across multiple trees, thereby reducing variance and overfitting [12]. Gradient boosting methods further improve accuracy by sequentially correcting the errors of previous learners, capturing complex feature interactions. Extreme Gradient Boosting (XGBoost), in particular, has gained traction in environmental modelling and time-series prediction due to its computational efficiency, support for missing values, and ability to incorporate heterogeneous features [14, 34]. Compared to deep learning methods such as CNNs or LSTMs, which typically require large-scale datasets [61], XGBoost is well-suited for medium-sized, structured datasets typical of room-level building applications.

This thesis therefore employs XGBoost regression as the core predictive model. Its suitability arises from three factors: (i) ability to incorporate heterogeneous predictors such as room volume, solar inflow, and external temperature, (ii) efficiency for periodic retraining on moderate datasets, and (iii) interpretability of feature importance, which ensures transparency in linking spatial and environmental variables to predictions. By adopting this approach, the study balances predictive accuracy with computational feasibility while maintaining compatibility with the standards-based data infrastructure developed in this research.

1.4. Research Objectives and Contributions

This study aims to develop a geospatial IoT system that captures, analyses, and visualizes indoor environmental parameters, specifically indoor temperature. It proposes a prototype system that uses open standards such as IFC for BIM representation and OGC SensorThings API for sensor data exchange. Data from Netatmo sensors, deployed in the BK Building at TU Delft, is collected at room level and stored in a FROST server. A key objective is to spatially map this sensor data onto a 3D BIM model (hosted on CesiumJS) to explore relationships between indoor conditions and building characteristics like volume and solar exposure.

The study attempts to predict environmental conditions using XGBoost, a supervised machine learning algorithm. The approach supports environments where direct sensor deployment is not feasible, offering a simulation based proxy.

The research is guided by three core questions:

- How can datastreams from IoT devices be stored, processed, and analyzed for deriving geospatial insights?
- Which measurable aspects of the built environment can be tested for correlation with indoor environmental quality?
- Can a simulation model based on building and sensor data be used to predict environmental conditions in non-instrumented spaces?

By exploring these questions, the study contributes to bridging the gap between IoT technologies and spatial data infrastructures. It lays the groundwork for smarter buildings using open source frameworks and tools.

Several environmental and spatial parameters can influence the internal temperature of a room. Numerous studies in building science and indoor environmental quality highlight the key drivers of indoor thermal conditions. Solar radiation particularly through glazed facades is frequently cited as a dominant factor affecting indoor temperatures, both through passive heating and overheating risks in perimeter zones [6, 5]. External climatic variables such as outdoor air temperature, humidity, wind speed, and pressure also significantly contribute to a room's thermal dynamics, as they directly affect energy exchange between indoor and outdoor environments [57]. Finally, the building's own thermal characteristics, especially room volume and thermal mass, play a critical moderating role, dampening temperature fluctuations and providing thermal inertia [58]. Together, these environmental inputs, spatial attributes, and building physics form the foundation of predictive modeling in room-scale thermal analysis.

1.5. Scope and Limitations

This thesis project focuses on analysing the patterns, correlations, and prediction of indoor temperature at the room (IFC space) level. Other indoor environmental parameters such as CO_2 concentration and noise levels have been excluded from the scope of this project. However, since the developed system is based on interoperable standards and indoor sensors are capable of providing a variety of parameters, indoor temperature could be readily substituted by CO_2 or noise to enable similar analyses and predictions. Additionally, factors such as the influence of HVAC systems and human behaviour have not been explicitly modelled; however, since the XGBoost prediction model used for predicting values is trained on observed data points, it is expected to capture some of these effects indirectly.

The temporal extent of the study is constrained by the limited timeframe of the thesis, which was less than one year. As a result, seasonal variations that might have provided additional insights into indoor environmental dynamics could not be incorporated. Furthermore, all three sensors used for data collection were installed within the same building. While this setup allowed a focused investigation of the effects of room volume and solar inflow on indoor temperature, it also meant that aspects such as building material properties and their influence on indoor conditions remained outside the scope of this work.

With regard to system implementation, the use of the OGC SensorThings API (STA) is limited to demonstrating interoperability and structured data exchange between IoT observations and building models within a prototype setting. Although the STA specification supports broader applications such as large-scale deployment, enterprise-level integration, and performance benchmarking [42, 60], these aspects are not addressed in this thesis. The objective here is to showcase the feasibility of linking sensor observations with building semantics in a standardised manner, rather than to evaluate the scalability or operational performance of the standard in complex environments.

Similarly, Industry Foundation Classes (IFC) provide a comprehensive, open standard for representing building information throughout the entire lifecycle, including design, construction, and facility management [13, 21]. In this thesis, however, IFC has been utilized in a more limited way, focusing on room-level semantics such as volume, orientation, and window placement, which are relevant to indoor temperature prediction. IFC was also partially employed for visualization through Cesium Tiles, but the rendered model is not fully interactive, as Cesium's 3D Tileset format does not currently support direct object selection or feature tagging at the level of individual rooms. Instead, room selection is facilitated through a dropdown list.

2 Literature Review

2.1. Related Work

The integration of IoT technologies into energy systems and built environments has been a growing area of research, particularly in the context of improving efficiency, sustainability, and decision-making. This section explores related literature across three key themes that align with the core components of this thesis: (1) IoT adoption in sustainable infrastructure, (2) predictive modeling using sensor time-series data, and (3) integration of spatial analytics into environmental simulations.

Fragkos et al. [23] emphasize that the adoption of IoT technologies can significantly reduce energy demand and support broader sustainability goals. By enabling real-time monitoring and control, IoT can contribute to smarter energy consumption in built structures, institutional management, transport sectors and beyond. These endeavors would align with the EU's low-carbon pathway to 2050 and demonstrate the potential of technology-driven solutions to reduce fossil fuel dependency and promote flexible, resilient energy systems. This broad vision sets the foundation for exploring more localized, room-scale implementations such as those proposed in this thesis.

Building on the need for smarter infrastructure, Alavi et al. [2] highlight key design principles that are critical to effective IoT deployment: system interoperability, contextual awareness, and real-time feedback mechanisms. These principles directly inform the technical backbone of this thesis, which uses the OGC SensorThings API, a geospatial database (PostGIS), and real-time streaming through the FROST server using FastAPI functionalities. This study underscores the value of integrating spatial data with sensor networks to support evidence-based decision-making, a concept adapted herein for building-scale environmental monitoring.

The integration of Building Information Modeling (BIM) with real-time IoT sensor data is a relatively new but rapidly evolving research area. Desogus et al. [17] present a data integration framework using low-cost IoT sensors and Revit-based BIM models, enabling dynamic visualization of indoor conditions such as temperature and luminance. Their system leverages tools like Dynamo and custom APIs to achieve real-time synchronization between the BIM environment and sensor readings. Similarly, several studies have demonstrated the potential of linking environmental sensors with 3D spatial models for applications like energy audits, retrofit decision-making, and occupant comfort monitoring [47]. These studies reinforce the potential of using BIM as a dynamic data hub, not just for geometry and documentation, but for managing live environmental data. This thesis builds on this foundation by creating a system that parses an IFC model directly and links it with room-specific sensor observations via the SensorThings API, forming the basis for both visualization and prediction.

Focusing more closely on the application of IoT at the scale of individual rooms and buildings, several studies explore how sensor-driven intelligence can optimize indoor environmental conditions. Imran et al. [35] investigate IoT-based task scheduling to reduce energy usage in residential spaces, incorporating human behaviour and appliance usage patterns to fine-tune consumption. While their emphasis is on appliance-level control, the underlying logic that granular sensor data can be modelled to support indoor efficiency aligns with this thesis. By shifting the unit of analysis from tasks to architectural features such as room volume and window exposure, this work extends that logic to a spatial-informatics-driven approach for environmental modelling.

Complementing the premise of IoT integration with prediction models, Xin et al. [61] explore a more algorithmically advanced approach of combining Convolutional Neural Networks (CNNs) with Long Short-Term Memory (LSTM) models to forecast short-term power consumption. Trained on large datasets from major Chinese cities, their hybrid model demonstrates strong predictive capabilities across both spatial and temporal dimensions. This research supports the use of structured time-series data such as sensor observations over time to train predictive models. While their focus lies in urban scale energy demand forecasting, the methodological principles align closely with this thesis. Herein, instead of deep learning, an XGBoost regression model is used. It has been chosen for its ability to handle heterogeneous feature types (room volume, solar inflow, external temperature), its speed of training and inference, and its effectiveness on medium-sized structured datasets, which is typical in building-level simulations. Furthermore, XGBoost supports incremental retraining and explainability, both of which are essential in this application where the model is updated daily and its behaviour must remain interpretable. This deliberate selection reflects a balance between predictive performance and operational feasibility for real-time environmental simulation.

In a comparable line of inquiry, Allam et al. [3] present a lightweight IoT framework for temperature and humidity forecasting using linear regression on sensor data gathered via Message Queuing Telemetry Transport (MQTT) protocol and stored in Amazon Web Services (AWS) DynamoDB. Although their use case focuses on basic weather monitoring, their end-to-end cloud architecture and reliance on timestamped sensor data echoes the structure of this thesis. However, unlike their use of static regression, this thesis implements a modular pipeline with support for incremental learning and model retraining using XGBoost, enabling more flexible deployment for real-time predictions.

Similarly, Paul et al. [49] explore multivariate forecasting of indoor temperature in smart buildings using a suite of machine learning models including Random Forests, Support Vector Machines, and Neural Networks, evaluated on high-resolution sensor data collected from a residential building in Spain. Notably, they advocate for an *online learning* methodology that adapts the model incrementally to new data—a strategy aligned with this thesis, where daily updates improve model accuracy based on the latest environmental inputs. Their study also reinforces the significance of solar irradiance and outdoor temperature as key predictors, both of which are central to the feature engineering in this thesis.

Beyond identifying predictive features, several studies also address how spatial factors—such as surface orientation and geometric exposure can be quantitatively modelled within building environments. Mardaljevic and Roy [44] propose a matrix-based approach for estimating cumulative Solar Beam Irradiance (SBI) using discretized sun position data throughout the year. Their methodology, integrated into BIM-based simulation pipelines, captures the directional and temporal variability of solar exposure across different facade elements. While this thesis employs a simplified, zenith-based method for estimating inflow due to its computational efficiency, it shares the same goal: transforming spatial geometry into model-ready inputs. Incorporating more granular SBI-based techniques presents a logical next step to enhance inflow realism in future work.

Taken together, these studies illustrate how IoT, BIM, and machine learning can be leveraged in diverse ways to address energy efficiency and environmental modelling. Yet, each contribution typically emphasizes only part of this integration, either IoT for monitoring, BIM for visualization, or ML for forecasting, without combining them into a unified, standards-based framework. To clarify how the present thesis builds upon and extends this body of work, Table 2.1 compares key case studies across their BIM, IoT, and prediction aspects, highlighting their focus areas and limitations relative to this research.

Table 2.1: Comparison of related work across BIM, IoT, and prediction aspects relative to this thesis

Study	Focus	BIM Aspect	IoT Aspect	Prediction / Modelling	Limitations
This Thesis	Prototype for room-level indoor temperature prediction in uninstrumented spaces	IFC semantics (room volume, window areas, SHGC, orientation)	OGC SensorThings API via FROST; sensor observations	XGBoost regression; incremental retraining; served to web app	Combines STA + IFC + ML for room-level prediction in uninstrumented rooms; open, standards-based.
Paul et al. (2018) [49]	Indoor temperature prediction in a smart building (Spain dataset)	None	Sensor-based dataset ingestion	Multivariate ML (RF, SVM, NN); online learning advocated	No BIM/IFC; no STA; no predictions for uninstrumented rooms.
Allam et al. (2021) [3]	Lightweight IoT + ML pipeline for temp/humidity (AWS)	None	MQTT ingestion; AWS DynamoDB storage (cloud)	Linear regression (static) forecasting	No BIM/spatial data; no STA; limited modelling scope.
Imran et al. (2022) [35]	IoT-based task scheduling for residential energy savings	None	Appliance/task-level IoT control	Predictive optimization of tasks (not room-scale)	No spatial features; no BIM semantics; no STA.
Xin et al. (2022) [61]	City-scale power management forecasting (smart cities)	None	Large-scale city datasets	CNN+LSTM spatio-temporal deep learning	Not building/room scale; no IFC/STA integration.
Desogus et al. (2021) [16]	BIM–IoT integration for monitoring existing buildings	Revit BIM with Dynamo/API linkage	Low-cost sensors; data ingestion and visualization	No prediction; real-time monitoring only	No IFC semantics; no STA; no ML framework.
Natephra & Motamedi (2019) [47]	AR + BIM live visualization of sensor data	BIM–AR interface for visualization	IoT streaming for display	No prediction; visualization-focused	No IFC/STA pipeline; no modelling.
Mardaljevic & Roy (2021) [44]	Solar Beam Irradiance (SBI) modelling for BIM simulations	BIM daylight/solar exposure estimation	None	Physical simulation (no ML)	No IoT; no STA; no predictive ML system.
Metallidou et al. (2020) [45]	Review of IoT approaches for energy-efficient smart buildings	None	Survey of IoT technologies and use-cases	No prediction; literature review	No semantic BIM; no modelling; no STA.
Alavi et al. (2018) [2]	IoT-enabled smart cities; design principles (interoperability, feedback, awareness)	None	Conceptual architecture and principles	No prediction; conceptual only	Lacks building-scale case studies; no STA; no BIM.
Fragkos et al. (2017) [23]	EU-level energy policy and IoT adoption for sustainability	None	Macro-scale IoT adoption for energy policy	No prediction; policy analysis only	No building-scale prototype; no BIM/IFC; no ML.

Apart from case-specific applications, a broader stream of literature provides reviews and methodological contributions that establish the foundations for ML in building-related prediction tasks. These works synthesize findings across multiple studies or introduce algorithms that have since become standard in environmental data modelling. Table 2.2 summarizes key review articles and methodological studies, showing how they validate the suitability of ML approaches and provide the theoretical tools upon which this thesis builds.

Table 2.2: Review and methodological studies on ML for building prediction tasks

Study	**Key Findings and Relevance**
Liu et al. (2018) [43]	Comprehensive survey of data driven methods for prediction and classification of building energy consumption. Highlights that ML approaches consistently outperform traditional statistical methods in energy and environmental prediction tasks. Validates ML's suitability for structured datasets in building applications.
Ahmad et al. (2017) [1]	Reviews both physics-based and data-driven approaches to building energy prediction. Shows growing dominance of ML methods for short-term indoor environmental forecasting. Provides theoretical grounding for ML over conventional statistical approaches.
Breiman (2001) [12]	Introduced Random Forests as an ensemble method. Demonstrated robustness to noise and overfitting through tree aggregation. Now widely applied in building level energy and comfort prediction studies.
Chen & Guestrin (2016) [14]	Proposed XGBoost as a scalable and efficient gradient boosting framework, capable of handling heterogeneous features, missing values, and medium-sized structured datasets efficiently. Provides the methodological backbone for the prediction model used in this thesis.
IBM (2023) [34]	Provides an accessible overview of XGBoost and its applied use-cases. Although not academic, it illustrates how the algorithm is adopted in real-world ML deployments, including environmental and energy forecasting domains.

2.2. Ensemble Models for Prediction

Ensemble methods in machine learning combine the predictive power of multiple individual models to achieve higher accuracy and robustness compared to single learners. The principle is rooted in the idea of the “wisdom of the crowd,” where aggregating multiple perspectives tends to yield better decisions [28].

Decision Trees as Base Learners

Decision trees form the foundation of many ensemble models. They are non-parametric supervised learning algorithms that can be applied to both classification and regression problems. Trees split data recursively into homogenous subsets using a divide-and-conquer strategy [32]. While they are easy to interpret, single decision trees often suffer from overfitting and instability, particularly when the tree becomes very deep [11].

Bagging and Random Forests

Bagging (bootstrap aggregating) was introduced to reduce variance by training multiple trees on bootstrapped samples of the dataset and averaging their predictions [12]. Random Forests extend bagging by introducing feature randomness, where each split in a tree considers only a random subset of features, thereby reducing correlation among trees and improving generalization [32]. This approach is widely recognized for its ease of use and strong performance across classification and regression tasks.

Boosting

In contrast to bagging, boosting trains weak learners sequentially, with each new model focusing on correcting the errors of its predecessors. AdaBoost was one of the first popular boosting algorithms, while gradient boosting advanced the approach by using gradient descent to minimize errors at each stage [24]. XGBoost represents a highly optimized implementation of gradient boosting, designed for scalability and computational efficiency [14, 34]. It incorporates features such as built-in regularization, sparsity-aware learning, and parallelization, making it well-suited for large-scale structured data tasks.

Bias–Variance Trade-off and Regularization

A central consideration in ensemble learning is the balance between bias and variance. While simple models such as shallow trees have high bias and low variance, complex models can exhibit low bias but high variance. Ensembles seek to find a balance by combining multiple learners [28]. Overfitting, where a model captures noise rather than the underlying pattern, is a common challenge and can be addressed using pruning, cross-validation, or regularization techniques [32, 33]. XGBoost, for instance, incorporates L_1 and L_2 regularization directly into its objective function, improving generalization compared to traditional gradient boosting [14].

Other Gradient Boosting Frameworks

Beyond XGBoost, other gradient boosting libraries have been developed with specific advantages. LightGBM focuses on efficiency through a leaf-wise tree growth strategy, which can be faster but more prone to overfitting if not carefully tuned [37]. CatBoost introduces native support for categorical features and employs techniques to reduce prediction bias, making it particularly effective for tabular datasets with many categorical variables [51].

In summary, ensemble models enhance predictive accuracy by aggregating the strengths of multiple learners. While bagging methods like Random Forests reduce variance, boosting methods such as XGBoost reduce bias and improve accuracy through sequential learning. The choice of ensemble method therefore depends on the data characteristics, computational resources, and the desired trade-off between interpretability and predictive power.

3 Methodology

This chapter presents the methodology adopted in this thesis to **design, develop, and evaluate a system** that utilizes OGC standards for sensor communication, integrates semantic data extracted from IFC-based BIM models, and employs an open-source machine learning framework for predicting indoor environmental conditions, specifically indoor temperature. The chapter focuses on the overall design rationale and methodological choices, while implementation details such as parameters, hardware, software tools, and specific code components are described in chapter 4.

3.1. Conceptual Workflow Overview

The conceptual workflow of the prototype system is summarized in Figure 3.1. This system integrates three primary data sources that collectively contribute to the training of a machine learning model, which is then employed to predict indoor temperature in rooms where no physical sensor is installed. The design is interoperable, enabling the same framework to be applied for other environmental parameters such as CO_2 or noise within the same building, and transferable across buildings at larger scales. Importantly, as demonstrated in section 6.2, predictive accuracy improves as additional data become available, highlighting the progressive reliability of the approach.

Internal Temperature = f(External Temp, Solar Inflow, Room Volume, ...)

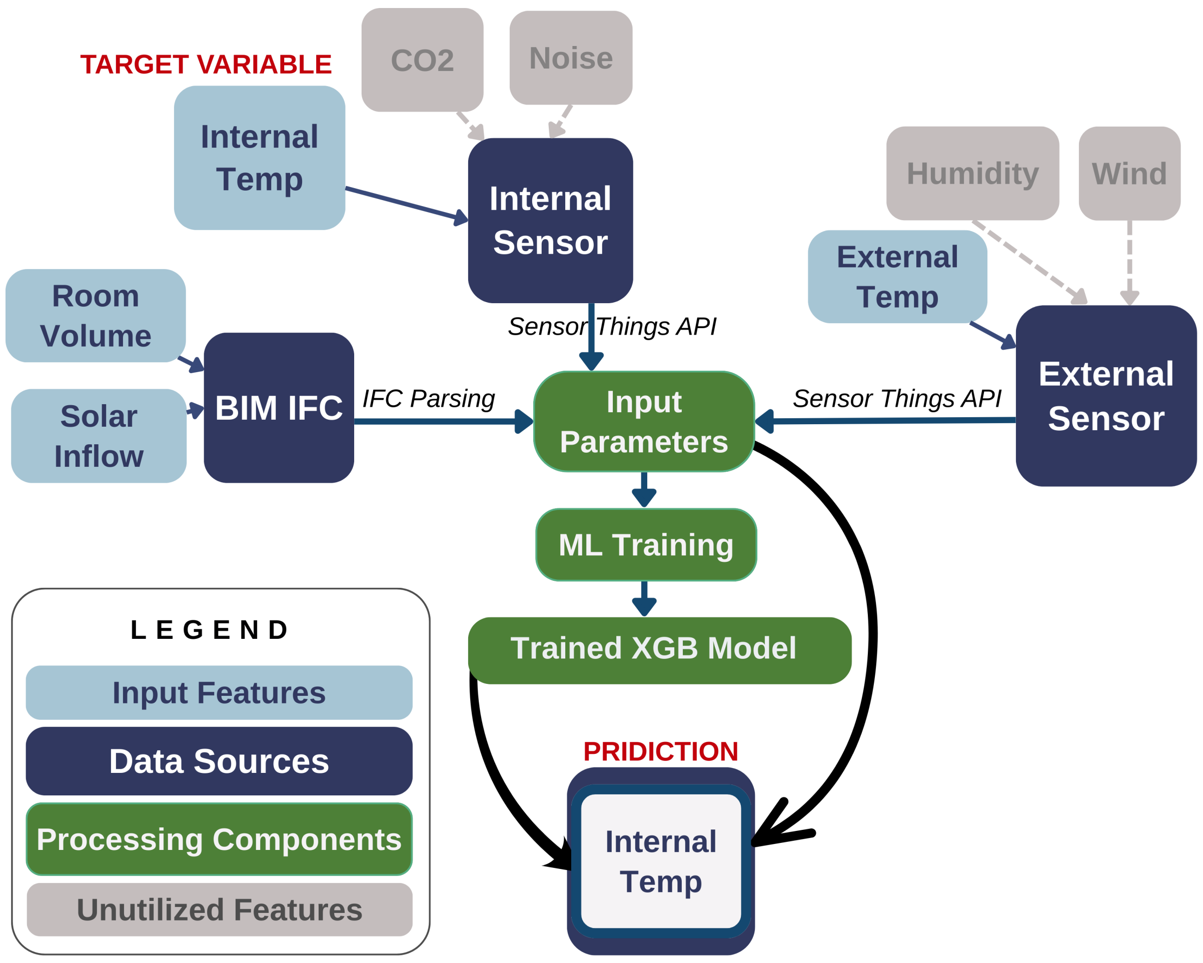


Figure 3.1: Conceptual workflow of the prototype system: integrating sensor observations, IFC-based building parameters, and machine learning for indoor temperature prediction

The methodology is structured into three broad sections:

- **Sensor Setup for Data Collection** – capturing indoor environmental conditions through IoT devices connected via OGC SensorThings API.
- **IFC BIM Model Processing and Visualisation** – extracting static spatial features such as room volume, orientation, and window geometry, and using these to derive solar inflow and provide visual interaction through a web interface.
- **ML Prediction Model Development** – applying XGBoost regression to predict indoor temperature using environmental and IFC-derived features.

Each component is introduced in the sections below.

3.2. Sensor Setup for Data Collection

Three indoor sensors were deployed in selected rooms of the Bouwkunde (BK) building at TU Delft to capture datastreams for indoor environmental parameters such as temperature, noise, and CO_2. Measurements were transmitted in real time through the FROST server, compliant with the Open Geospatial Consortium (OGC) SensorThings

API [41]. The ingested data were stored in a PostgreSQL database with PostGIS extensions, supporting both spatial queries and temporal data management. A practical advantage of STA is its support for server side querying via OData parameters such as `$filter`, `$orderby`, and `$top`. By delegating time-window filtering and ordering to the FROST server, only the necessary observations are returned to the application.

Rooms were selected by physical installation of sensors based on practical factors such as accessibility and diversity of baseline characteristics (e.g., different room volumes and solar inflow exposures). Rather than a strictly model-driven deployment, the emphasis was on feasibility and rapid setup, treating the collected data as exploratory. This reflects realistic implementation constraints while enabling the investigation of correlations between building attributes and environmental conditions.

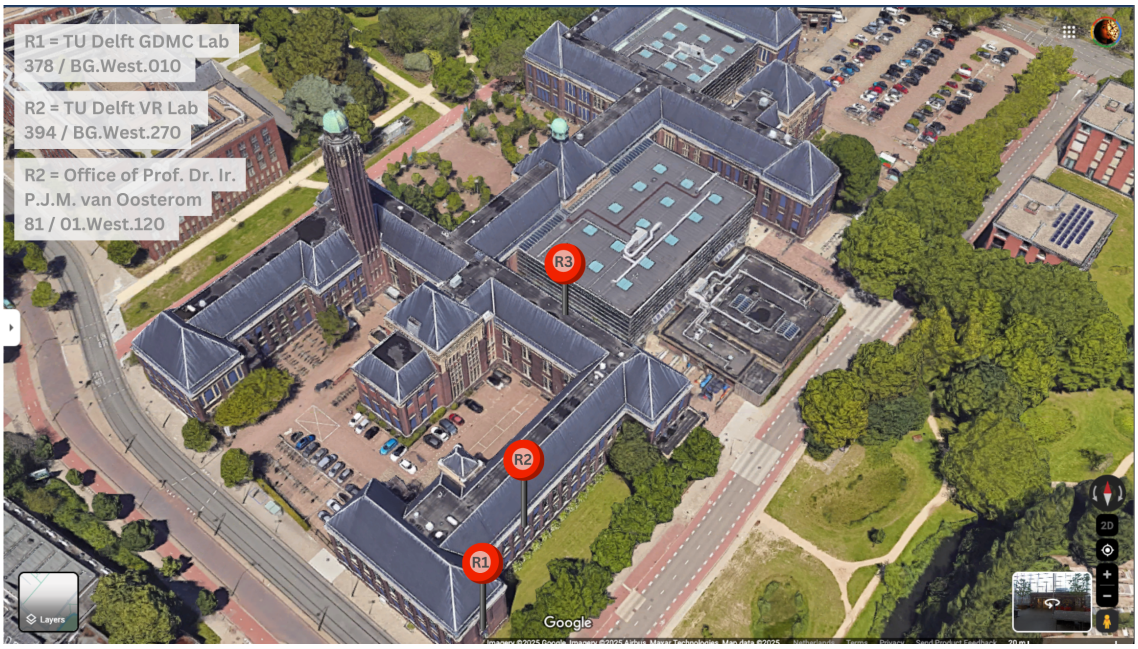
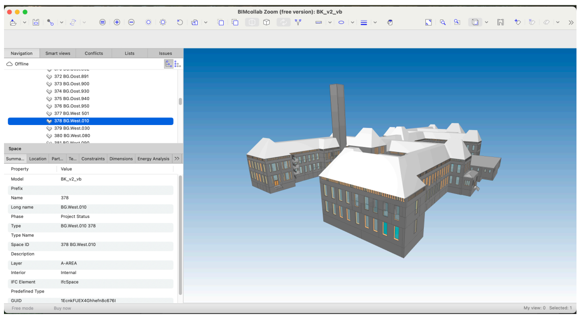
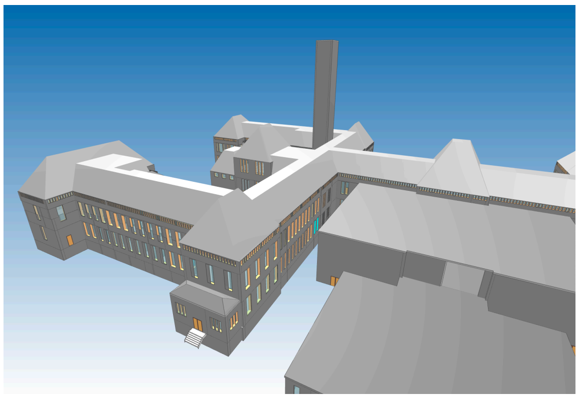
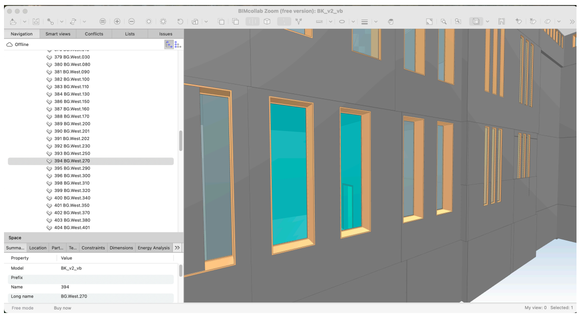


Figure 3.2: Position of indoor sensors in the BK building

Figure 3.3 illustrates the sensor deployment in the IFC-based BIM model of BK. These rooms were explored using BIMcollab Zoom and the IFCOpenShell library. An overview of the rooms with their IoT and datastream IDs is provided in Table 4.1. The technical details of FROST server configuration, STA mapping, and querying patterns are presented in chapter 4.


(a) Room 378: BG.West.010


(b) Room 394: BG.West.270 (Exterior)


(c) Room 394: BG.West.270 (Interior)

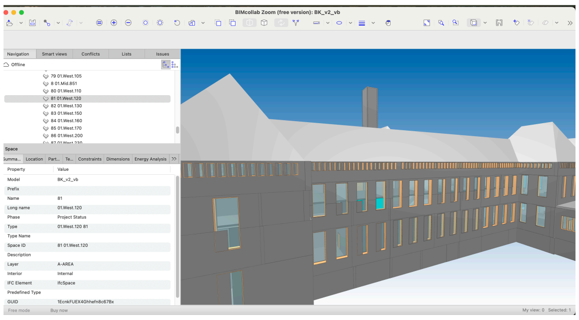

(d) Room 81: 01.West.120

Figure 3.3: IFC-based BIM visualization of rooms selected for sensor deployment and analysis

Table 3.1: Overview of Rooms with IoT Sensors and Corresponding Datastream IDs

Room Name	IFC Short / Long Name	Remarks	IoT ID	DS ID In. Temp	DS ID CO2	DS ID noise
TU Delft GDMC	378 / BG.West.010	GDMC lab at the Faculty of Architecture	1	1	3	6
TU Delft VR Lab	394 / BG.West.270	VR Lab at Faculty of Architecture [40]	2	7	9	12
Room 120	81 / 01.West.120	Office of Prof. P Oosterom [18]	3	13	15	18

3.3. IFC for Visualisation and Solar Inflow

The Industry Foundation Classes (IFC) standard was used as a neutral, non-proprietary format for representing the Bouwkunde (BK) building model. IFC supports reproducibility, semantic richness, and interoperability across BIM applications [13]. In this study, the IFC model was the source of several static parameters: room volumes, window geometries, Solar Heat Gain Coefficients (SHGC), and orientations. These parameters were linked with IoT sensor data to enrich both training datasets and prediction inputs.

Two primary uses of the IFC model were:

1. Extracting static building semantics such as room volume and site location.

2. Supporting dynamic calculations such as solar inflow, based on window area, orientation, and SHGC.

Some preprocessing was required: geographic coordinates were corrected, and a custom spatial routine was developed to associate windows with rooms where `IfcRelSpaceBoundary` relationships were absent. The implementation of these preprocessing routines is detailed in chapter 4.

3.3.1. Conceptual Basis of Solar Inflow Calculation

Solar inflow represents the portion of incident solar radiation transmitted into an indoor space through its external glazed surfaces. Within the context of this system, it serves as one of the key input features for the machine learning model that predicts indoor temperature dynamics. Specifically, solar inflow denotes the cumulative solar energy transmitted into a room through all *external* windows during a defined five-minute interval.

The estimation of solar inflow depends on both building geometry and environmental conditions. It **quantifies the combined influence of solar radiation and window geometry** including factors such as orientation, surface area, and material transmittance.

Conceptually, the computation of solar inflow is governed by geometric and physical relationships that describe how incident solar radiation interacts with building facades [20, 8]. The total inflow for a given window surface over a time interval Δt can be expressed as:

$$S_{\Delta t} = A_g \cdot \cos\theta \cdot I_{AM} \cdot \tau \cdot \Delta t \tag{3.1}$$

where:

- A_g glazed area of the window,
- θ solar incidence angle relative to the window normal,
- I_{AM} irradiance corrected for air mass and atmospheric attenuation, estimated using standard transposition and clear-sky models [50, 36, 54],
- τ transmittance or Solar Heat Gain Coefficient (SHGC) of the glazing material,
- Δt the time interval over which inflow is integrated.

Solar position parameters required to compute θ are determined using the Solar Position Algorithm (SPA) or its implementation in the `pvlib` library [53, 52].

This formulation highlights that solar inflow is not a purely geometric variable but a function of both geometry and environmental context. The angle θ varies temporally with the sun's position, which depends on geographic location, building orientation, and time of day. The parameter I_{AM} captures atmospheric effects, while τ represents the physical transmission characteristics of the window material.

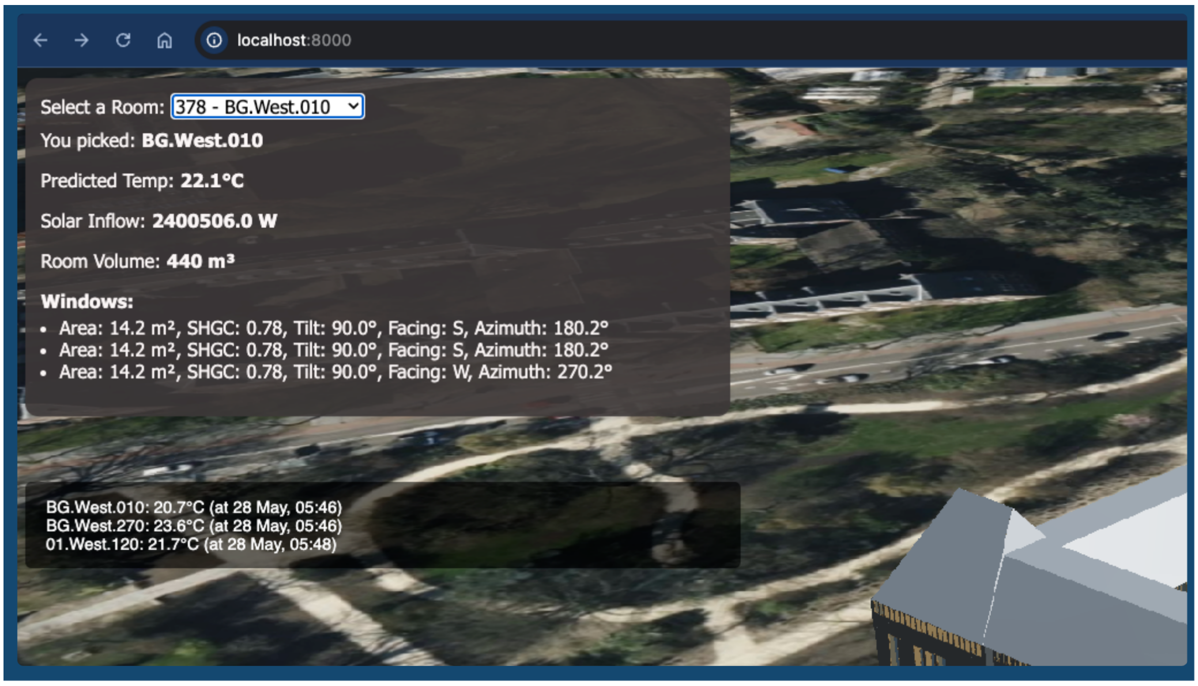
Conceptually, the method involves three sequential steps:

1. **Solar Position Determination:** calculating the sun's azimuth and elevation based on geographic coordinates and timestamp using standard solar geometry models.
2. **Window Orientation and Area Extraction:** obtaining window surface normals, window area, and externality flags from the IFC model.
3. **Solar Irradiance Computation:** estimating the direct and diffuse irradiance components on each surface and integrating them to obtain total inflow for the defined interval.

The above process forms the theoretical foundation for solar inflow computation. Implementation details, including parameter sources and equations used for irradiance correction, are described in chapter 4.

3.3.2. Visualization Interface

A web application was developed to visualize IFC-derived building geometry and display predictions. Built using CesiumJS and FastAPI, the interface enables users to select rooms and view predicted indoor temperatures alongside observed values when available (Figure 3.4). The visualisation enhances interpretability of model outputs while linking predictions to building semantics.



Select a Room: 378 - BG.West.010

You picked: **BG.West.010**

Predicted Temp: **22.1°C**

Solar Inflow: **2400506.0 W**

Room Volume: **440 m^3**

Windows:

- Area: 14.2 m^2, SHGC: 0.78, Tilt: 90.0°, Facing: S, Azimuth: 180.2°
- Area: 14.2 m^2, SHGC: 0.78, Tilt: 90.0°, Facing: S, Azimuth: 180.2°
- Area: 14.2 m^2, SHGC: 0.78, Tilt: 90.0°, Facing: W, Azimuth: 270.2°

```
BG.West.010: 20.7°C (at 28 May, 05:46)
BG.West.270: 23.6°C (at 28 May, 05:46)
01.West.120: 21.7°C (at 28 May, 05:48)
```

Figure 3.4: Web-based application interface for visualizing IFC-derived building model and displaying predicted indoor temperature results for user-selected rooms

3.4. ML Prediction Model Development

At the core of this system lies an XGBoost regression model trained to predict indoor temperature. Input features include:

- Room volume (from IFC)

- Solar inflow (calculated from IFC geometry and solar data)
- External temperature (from public APIs)

The target variable is the indoor temperature recorded by Netatmo sensors. Predictions are generated for rooms without sensors, extending monitoring coverage.

The model is designed for incremental retraining, whereby newly available sensor observations are incorporated into the training set. As shown in section 6.2, the inclusion of an additional month of training data improved accuracy by 7% (RMSE), validating the assumption that more data leads to more robust models. In deployment, when a user selects a room in the web interface, the most recent model is retrieved from the server, ensuring that predictions reflect the latest data and environmental dynamics. Full implementation details, including code, retraining scripts, and versioning strategies, are provided in chapter 4.

3.5. System Architecture

The system architecture integrates **Building Information Modeling (BIM)**, **Internet of Things (IoT)**, and **Machine Learning (ML)** components into a unified workflow for indoor temperature prediction (in the context of this thesis project). The data flow from external and internal inputs to prediction delivery is illustrated in Figure 3.5. This architecture establishes a continuous data and prediction loop in which BIM geometry, IoT observations, and environmental factors converge to produce real-time, room-level indoor temperature predictions.

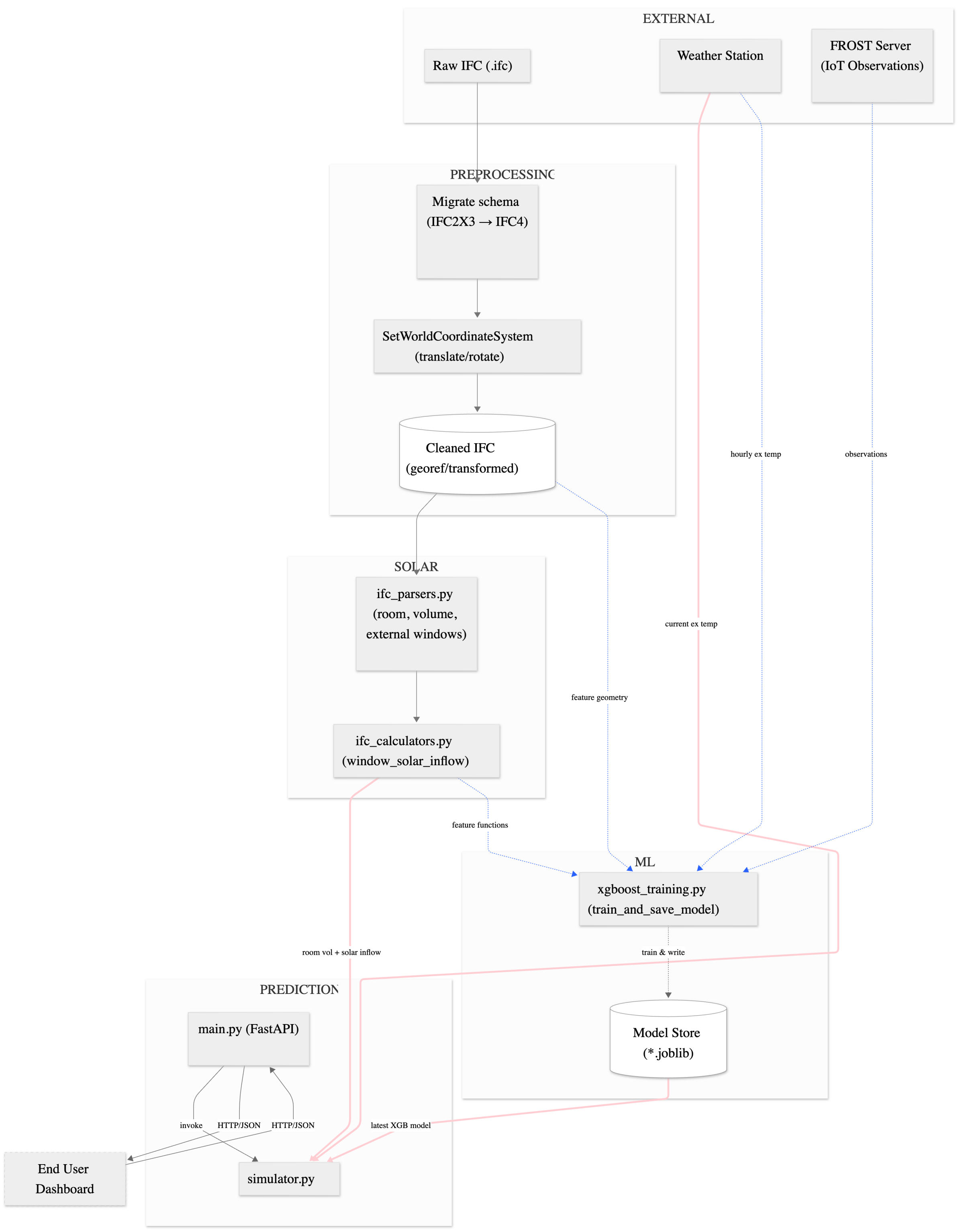


Figure 3.5: System architecture with training loop (pink) and prediction loop (blue)

3.5.1. Data Flow

The overall workflow comprises two loops: a **training loop** (blue lines in Figure 3.5) and a **prediction loop** (pink lines in Figure 3.5). The distinction between the two lies in their objectives. The purpose of the training loop is to iteratively improve the accuracy of the XGBoost model using input features, including room volume and solar inflow. In contrast, the prediction loop focuses on fetching the latest trained XGBoost model and, using the same input features, predicting the internal temperature of unsupervised rooms.

The key steps in the overall data flow are as follows:

1. External data are collected from the Weather Station.
2. The IFC model is preprocessed for schema migration and spatial alignment.
3. Solar and geometric features are computed from the IFC model.

Training-Specific Steps

- Room-specific internal temperature data are fetched for model training.
- ––*overall workflow*————
- The XGBoost model is trained and stored.

Prediction-Specific Steps

- The user selects a specific room as input.
- The FastAPI service loads the most recent XGBoost model for real-time prediction.
- ––*overall workflow*————
- The prediction is generated and displayed on the web application.

The data for an entire week is prepared to have hourly entries from each of the three rooms. the cleaned dataset in then used to train an XGBoost model which is stored on the server - this is a weekly process. When deployed in real time prediction, the solar inflow for last 5 minutes is calculated to act as input for the latest trained model in the server.

3.5.2. External Data Sources

Two external data sources are utilized:

1. **Weather Station:** Provides outdoor temperature values, which serve as the input with the highest feature importance. Data are extracted as hourly averages in the training loop and as a single live reading in the prediction loop.
2. **FROST Server:** Contains IoT observations, including indoor temperature and CO_2 levels, collected through deployed sensors. This data source is used exclusively in the training loop, as it provides the target feature for the prediction exercise.

These data streams form the observational basis for model training where both external data sources are used and for live prediction, which relies on weather station data for outdoor temperature values.

3.5.3. Framework for IFC Preprocessing

The system developed in this project utilizes the Industry Foundation Classes (IFC) format for two major purposes:

1. **Geometric properties:** These define the relative placement of building elements within a three-dimensional space. This geometric hierarchy forms the 3D model itself and is essential both for visualization and for determining the relative positions of `IfcSpaces` and `IfcWindows`. Through this structure, the developed system identifies all external windows associated with a selected room.
2. **Attribute information:** In addition to geometry, IFC entities store critical attributes that are required by the system, including:

   2.a. Coordinates of the `IfcSite` element (`RefLatitude`, `RefLongitude`).

   2.b. The overall building orientation within a defined Coordinate Reference System (CRS).

   2.c. **Room volume:** extracted from `IfcSpace` under the property set `BaseQuantities.GrossVolume` (and alternatively `Qto\_SpaceBaseQuantities.GrossVolume` for IFC4 models).

   2.d. **Window attributes:**

      2.d.1. Window area, obtained from `BaseQuantities.Area`.

      2.d.2. Solar Heat Gain Coefficient (SHGC) of the window material, retrieved from the vendor specific property set `Analytical Properties(Type)`.

      2.d.3. Externality flag, identified through `Pset_WindowCommon.IsExternal`, which specifies whether a window faces the external environment.

Preprocessing of IFC data may therefore be required to ensure that building model supplied to the system exposes all of the geometric and attribute information in a consistent, accessible manner. Depending on the source and schema of the IFC input, steps such as schema migration, coordinate transformation, or attribute normalization may be necessary to make the model compatible with downstream analytical components. The specifics of IFC preprocessing carried out in this thesis are described in detail in the Implementation chapter (see section 4.3), where the technical procedures, libraries, and code routines are elaborated.

3.5.4. Solar Geometry and Feature Extraction

Following IFC preprocessing, the system derives spatial and solar geometry features that serve as inputs to the prediction model. This process connects the static building model with dynamic environmental conditions, producing physically interpretable variables (input features) for model training.

Methodologically, this process integrates two key modules:

- `ifc_parsers.py`: extracts geometric parameters such as room boundaries, volumes, and window surfaces associated with each `IfcSpace`. The module identifies external windows through the attribute `Pset_WindowCommon.IsExternal` and retrieves their geometric attributes from `BaseQuantities.Area` and normal vectors derived from the window placement data.

- `ifc_calculators.py`: computes solar inflow values for each room by combining the geometric outputs of the parser with solar position and irradiance data from the weather station. The procedure accounts for the solar incidence angle on each window, its area, and SHGC value to estimate incoming solar energy during a given time interval (this was taken as a five minute window).

The solar inflow, computed for each five-minute interval, serves as one of the principal input features in the system, used both during model training and real-time prediction. In the **training loop**, solar inflow is computed for a five-minute interval corresponding to each hourly record in the training dataset. The data prepared for an entire week therefore consists of hourly records from each of the three monitored rooms. This cleaned and aggregated dataset is then used to train the XGBoost regression model, which is subsequently stored on the server as part of the weekly model update cycle. In **real-time deployment**, the solar inflow corresponding to the most recent five-minute interval is computed and supplied as an input to the latest trained model hosted on the server.

This formulation is grounded in the hypothesis (examined and corroborated in Section 5.2.1) that solar inflow exerts a significant influence on the indoor thermal environment, such that variations in solar gains are expected to correspond with measurable changes in internal temperature. As an integrated expression of window area, orientation, and material properties (SHGC), it reflects the extent to which these physical characteristics govern the transmission of solar energy into indoor spaces.

Consequently, solar inflow, in conjunction with room volume, functions as key physical parameters governing indoor thermal behaviour. These IFC-derived features are subsequently combined with external temperature data from the weather station and indoor temperature observations from the FROST server to form the complete input matrix for model training and prediction. Conceptually, this feature extraction pipeline ensures that each variable in the dataset reflects an interpretable physical relationship within the building environment system. In other words, solar inflow captures the dynamic thermal gain from the sun, room volume captures the space's thermal capacity, and external temperature represents the climatic forcing condition. Together, these features establish the methodological foundation of the system's predictive framework.

3.5.5. Machine Learning Pipeline

The ML module, implemented in `xgboost_training.py`, constitutes the analytical core of the system. It consumes:

- Geometric and solar inflow features,
- Current external temperature data, and
- Historical indoor observations from the FROST Server.

The script trains a gradient-boosted regression model (XGBoost) to predict indoor temperature. The trained model is serialized into the **Model Store** as a `.joblib` file, ensuring reproducibility and efficient inference.

Prediction and Deployment Layer

At runtime, the latest trained model is deployed using a **FastAPI-based prediction service** defined in `main.py`. The service exposes REST endpoints that receive feature data and return temperature predictions in JSON format. The `simulator.py` module acts as an intermediary between the model and the user interface, invoking the API and visualizing the predicted outcomes on the dashboard.

End users access the predictions through an interactive dashboard, enabling them to explore temperature dynamics under varying environmental conditions.

4 Implementation Details

This chapter presents the technical implementation of the methodology outlined in chapter 3. While the previous chapter introduced the design rationale and conceptual frameworks, here the focus is on the software setup, data processing routines, detailed application of standards, and coding strategies. The discussion covers (i) the local development environment, (ii) implementation of the OGC SensorThings API using FROST, (iii) solar inflow computation, and (iv) the machine learning pipeline built with XGBoost.

Overview of Implementation Workflow. To contextualize the subsequent module descriptions, the end-to-end implementation pipeline is first outlined. The system operates as a modular sequence linking the API interface, simulation controller, IFC geometry parsers, solar irradiance calculators, and the prediction model. When a user selects a room in the Cesium-based web client, the FastAPI endpoint `/api/simulate/{room_name}` is invoked. This triggers `simulator.py` to (i) parse the selected room from the IFC model via `ifc_parsers.py`, (ii) compute per-window five-minute solar inflow via `ifc_calculators.py`, (iii) retrieve the current external temperature from the weather station API, and (iv) load the latest trained XGBoost model. The feature vector comprising five-minute solar inflow, room volume, and external temperature is then passed to the predictor, and the resulting temperature prediction is returned to the client as structured JSON for visualization.

The overall sequence of these interactions, beginning from user input on the Cesium web interface and extending through data retrieval, feature computation, and temperature prediction, is illustrated in Figure 4.1. The diagram clarifies the flow of control between the FastAPI service layer, backend simulation modules, and analytical components, emphasizing how geometry extraction, solar irradiance computation, and model inference are orchestrated in the system. The overarching methodological loops corresponding to the training and prediction workflows are depicted in Figure 3.5

For offline training, the same components assemble a historical dataset. Indoor temperature observations are fetched from the FROST SensorThings endpoint for each monitored room and merged into a single table. For each observation timestamp, external temperature is retrieved at the site location and five-minute solar inflow is computed by summing window-level inflows for the room (see section 4.4). Helper routines (`fetch_all_sensor_data()`, `fetch_external_temp()`, `calculate_total_solar_inflow()`, `prepare_training_data()`) orchestrate these steps and return a DataFrame with columns `[timestamp,room_name,internal_temp,external_temp,volume,solar_inflow]`. Hourly downsampling is performed by flooring timestamps to the hour and averaging all observations within each hour, ensuring a consistent and smoothed training cadence across rooms.

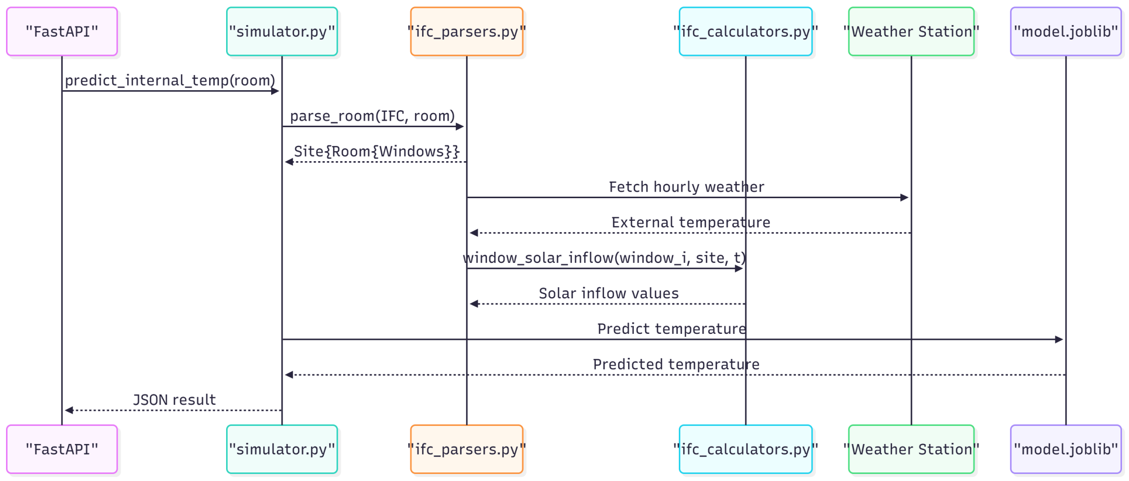


Figure 4.1: Sequence diagram illustrating the end-to-end inference workflow, showing data flow from user input to IFC parsing, irradiance computation, and model prediction.

The FastAPI layer invokes `simulator.py`, which parses IFC geometry (`ifc_parsers.py`), computes solar inflow (`ifc_calculators.py`), fetches external temperature from the weather station, and loads the serialized XGBoost model (`model.joblib`). The predicted indoor temperature is returned as a JSON response to the Cesium-based web interface.

4.1. Code Setup

All code development for this thesis was carried out locally using Visual Studio Code (VS Code) as the integrated development environment (IDE). The setup used the following build:

- **VS Code Version:** 1.103.2 (Universal) with **Python** `3.11.11`
- **Platform:** macOS (Darwin arm64 24.6.0)

The project folder linked to local Git repository has been synchronized with public GitHub repository accessible at: `https://github.com/vidushi711/BKviewer_FastAPI`.

Git setup allowed smooth collaboration with supervisors and ensured version tracking for reproducibility.

Python dependencies were managed with **uv**, a fast Rust-based package manager that provides lockfile-based dependency resolution. All dependencies were declared in `pyproject.toml` and resolved into `uv.lock`, ensuring reproducibility.

4.2. SensorThings API Implementation (STA Mapping)

The OGC SensorThings API (STA) specification [41] was applied consistently to structure the data. It provided a standard, queryable representation of the project's IoT data. STA core entities `Thing`, `Datastream`, `Observation`, `ObservedProperty`, and `FeatureOfInterest` map directly to the hardware (Netatmo devices), variables (temperature, CO_2, noise), and building context (room as FeatureOfInterest) used in this thesis. This alignment allows observations to be linked unambiguously to IFC rooms and subsequently combined with geometry derived attributes (e.g., volume, window orientation) in the downstream pipeline. A practical advantage of STA is its support for server-side querying via OData parameters such as `$filter`, `$orderby`, and `$top`. By delegating time-window filtering and ordering to the FROST server, only the necessary observations are returned to the application. In practice, this reduced retrieval time markedly: client-side filtering of one month of data required more than twenty minutes, whereas the same query executed on the server returned in approximately twelve seconds.

4.2.1. Querying Patterns

Practical querying patterns included:

- By datastream and time window: `/v1.0/Observations?$filter=Datastream/ideq1andphenomenonTimege'2024-01-01T00:00:00Z'andphenomenonTimelt'2024-02-01T00:00:00Z'&$orderby=resultTimeasc`
- Selecting minimal fields: `/v1.0/Observations?$select=result,resultTime`
- Expanding linked entities: `/v1.0/Datastreams(1)?$expand=Observations($select=result,resultTime)`
- Bulk reads: `/v1.0/Datastreams(1)/Observations?$resultFormat=dataArray`

4.2.2. Server Deployment

The FROST server was deployed using Docker and linked to a PostgreSQL database with PostGIS. All entities (Things, Locations, ObservedProperties, Sensors, and Observations) were stored in the schema, maintaining links between IFC-derived rooms and IoT datastreams.

System efficiency and API-level data management. A notable advantage of this deployment lies in how data is accessed through the OGC SensorThings API. The system performs server-side filtering, ordering, and limiting directly within the FROST endpoint by passing parameters such as:

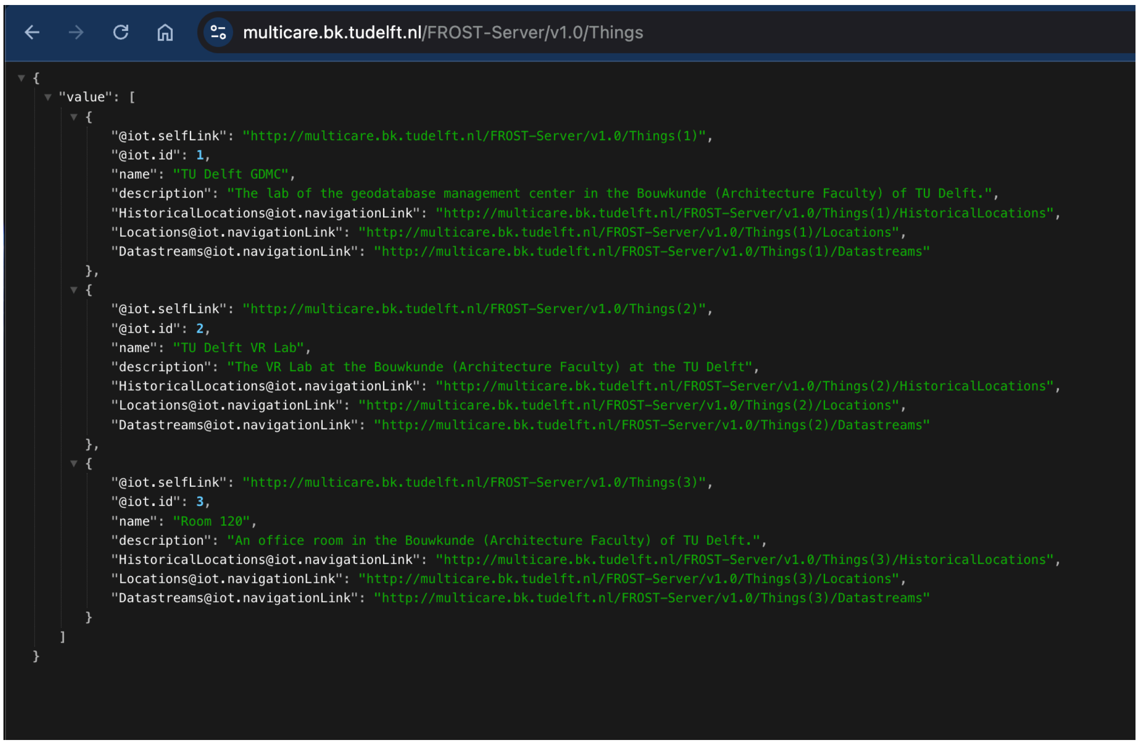
```
?$orderby=phenomenonTime desc&$top=...
?$filter=phenomenonTime ge ... and phenomenonTime le ...
```

This approach delegates heavy operations such as time filtering, sorting, and pagination to the database tier inside FROST rather than the client. As a result, only small, structured JSON payloads containing the relevant time window of observations are transferred to the FastAPI layer or Jupyter notebooks. In practice, this optimization reduced retrieval time dramatically: when the filtering and aggregation were executed on the client side, fetching one month of data required more than twenty minutes, whereas delegating the same operation to the FROST server returned the results in approximately twelve seconds.

This architecture highlights an important design strength of the developed system. By leveraging the API's server-side computation, the workflow achieves:

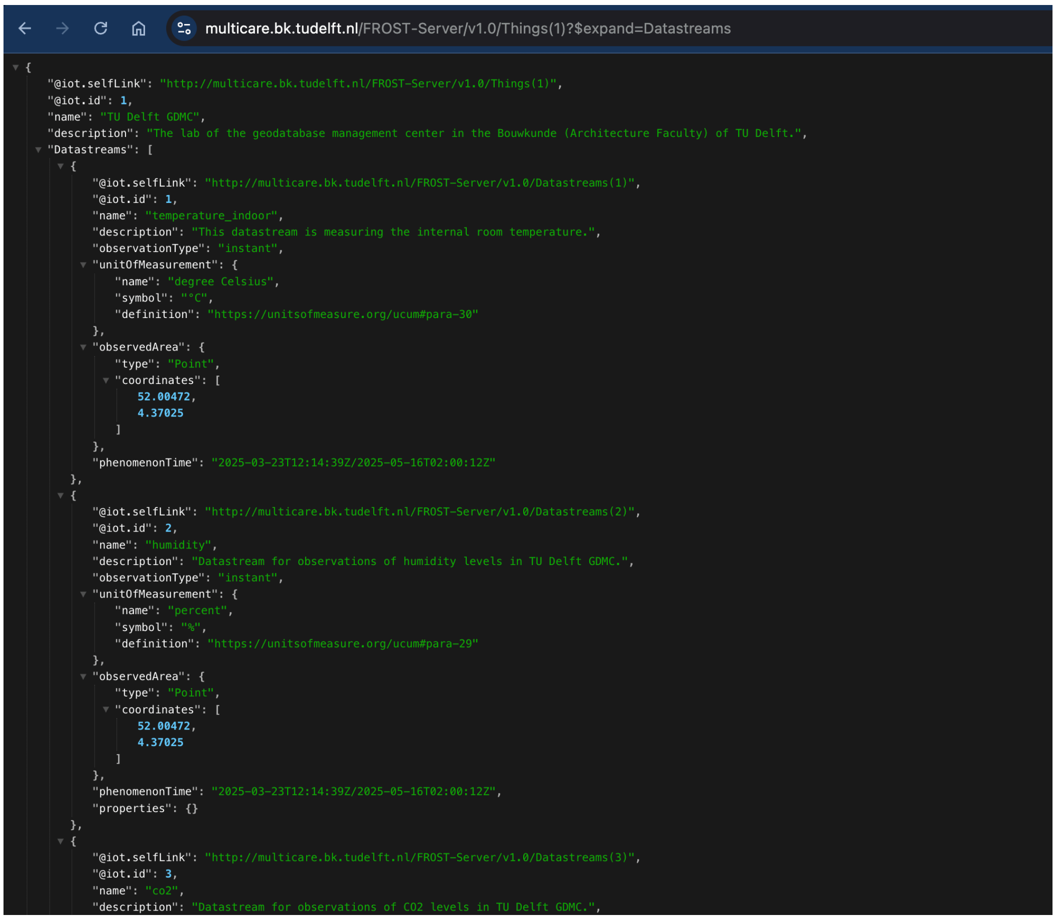
- **Efficiency and scalability** — large IoT datastreams are filtered and aggregated near the data source, reducing bandwidth and local processing load;
- **Smart edge processing** — FROST's optimized query engine handles indexing and time-based retrieval, so the client only receives ready-to-use subsets of data;
- **Responsiveness for real-time analysis** — the system can serve updated readings or aggregated hourly data with minimal latency, supporting interactive visualization and model retraining cycles.

Technically, this means that the FROST–FastAPI–script pipeline is designed around API-level efficiency rather than bulk data transfer. It acts as a lightweight, queryable feature store that provides quick inputs for prediction and training, enabling scalable integration of additional sensors and datastreams with minimal modification.


multicare.bk.tudelft.nl/FROST-Server/v1.0/Things

```
{
  "value": [
    {
      "@iot.selfLink": "http://multicare.bk.tudelft.nl/FROST-Server/v1.0/Things(1)",
      "@iot.id": 1,
      "name": "TU Delft GDMC",
      "description": "The lab of the geodatabase management center in the Bouwkunde (Architecture Faculty) of TU Delft.",
      "HistoricalLocations@iot.navigationLink": "http://multicare.bk.tudelft.nl/FROST-Server/v1.0/Things(1)/HistoricalLocations",
      "Locations@iot.navigationLink": "http://multicare.bk.tudelft.nl/FROST-Server/v1.0/Things(1)/Locations",
      "Datastreams@iot.navigationLink": "http://multicare.bk.tudelft.nl/FROST-Server/v1.0/Things(1)/Datastreams"
    },
    {
      "@iot.selfLink": "http://multicare.bk.tudelft.nl/FROST-Server/v1.0/Things(2)",
      "@iot.id": 2,
      "name": "TU Delft VR Lab",
      "description": "The VR Lab at the Bouwkunde (Architecture Faculty) at the TU Delft",
      "HistoricalLocations@iot.navigationLink": "http://multicare.bk.tudelft.nl/FROST-Server/v1.0/Things(2)/HistoricalLocations",
      "Locations@iot.navigationLink": "http://multicare.bk.tudelft.nl/FROST-Server/v1.0/Things(2)/Locations",
      "Datastreams@iot.navigationLink": "http://multicare.bk.tudelft.nl/FROST-Server/v1.0/Things(2)/Datastreams"
    },
    {
      "@iot.selfLink": "http://multicare.bk.tudelft.nl/FROST-Server/v1.0/Things(3)",
      "@iot.id": 3,
      "name": "Room 120",
      "description": "An office room in the Bouwkunde (Architecture Faculty) of TU Delft.",
      "HistoricalLocations@iot.navigationLink": "http://multicare.bk.tudelft.nl/FROST-Server/v1.0/Things(3)/HistoricalLocations",
      "Locations@iot.navigationLink": "http://multicare.bk.tudelft.nl/FROST-Server/v1.0/Things(3)/Locations",
      "Datastreams@iot.navigationLink": "http://multicare.bk.tudelft.nl/FROST-Server/v1.0/Things(3)/Datastreams"
    }
  ]
}
```

Figure 4.2: Three registered Things on the FROST server, each representing a Netatmo sensor located in different rooms.


multicare.bk.tudelft.nl/FROST-Server/v1.0/Things(1)?$expand=Datastreams

```
{
  "@iot.selfLink": "http://multicare.bk.tudelft.nl/FROST-Server/v1.0/Things(1)",
  "@iot.id": 1,
  "name": "TU Delft GDMC",
  "description": "The lab of the geodatabase management center in the Bouwkunde (Architecture Faculty) of TU Delft.",
  "Datastreams": [
    {
      "@iot.selfLink": "http://multicare.bk.tudelft.nl/FROST-Server/v1.0/Datastreams(1)",
      "@iot.id": 1,
      "name": "temperature_indoor",
      "description": "This datastream is measuring the internal room temperature.",
      "observationType": "instant",
      "unitOfMeasurement": {
        "name": "degree Celsius",
        "symbol": "°C",
        "definition": "https://unitsofmeasure.org/ucum#para-30"
      },
      "observedArea": {
        "type": "Point",
        "coordinates": [
          52.00472,
          4.37025
        ]
      },
      "phenomenonTime": "2025-03-23T12:14:39Z/2025-05-16T02:00:12Z"
    },
    {
      "@iot.selfLink": "http://multicare.bk.tudelft.nl/FROST-Server/v1.0/Datastreams(2)",
      "@iot.id": 2,
      "name": "humidity",
      "description": "Datastream for observations of humidity levels in TU Delft GDMC.",
      "observationType": "instant",
      "unitOfMeasurement": {
        "name": "percent",
        "symbol": "%",
        "definition": "https://unitsofmeasure.org/ucum#para-29"
      },
      "observedArea": {
        "type": "Point",
        "coordinates": [
          52.00472,
          4.37025
        ]
      },
      "phenomenonTime": "2025-03-23T12:14:39Z/2025-05-16T02:00:12Z",
      "properties": {}
    },
    {
      "@iot.selfLink": "http://multicare.bk.tudelft.nl/FROST-Server/v1.0/Datastreams(3)",
      "@iot.id": 3,
      "name": "co2",
      "description": "Datastream for observations of CO2 levels in TU Delft GDMC.",
```

Figure 4.3: Datastreams linked to Thing with @iot.id:1, showing observed properties such as temperature, humidity, and CO_2.

Table 4.1: Summary of registered Things and their temperature Datastreams

IoT ID	Room Name	IFC Short / Long Name	DS ID	Temperature URL	Remarks
1	TU Delft GDMC	378 / BG.West.010	1	`https://multicare.bk.tudelft.nl/FROST-Server/v1.0/Datastreams(1)`	GDMC lab located in BG.West.010, Faculty of Architecture [25].
2	TU Delft VR Lab	394 / BG.West.270	7	`https://multicare.bk.tudelft.nl/FROST-Server/v1.0/Datastreams(7)`	VR Lab situated in BG.West.270 [40].
3	Room 120	81 / 01.West.120	13	`https://multicare.bk.tudelft.nl/FROST-Server/v1.0/Datastreams(13)`	Office of Prof. P J M van Oosterom, in 01.West.120 [18].

Overall, the STA-based design benefits all components of the system. The FastAPI layer remains thin, since the FROST endpoint delivers time filtered and ordered observations that can be used directly for feature construction and inference. The training scripts can reproduce large historical pulls efficiently using stable, human-readable

URLs and consistent query parameters, while the Cesium interface can request only the latest values for responsive visualisation. Compared with alternatives such as bespoke REST/GraphQL over a time-series database, vendor-specific cloud APIs, or raw MQTT/CSV workflows, STA provides a standard vocabulary and mature query semantics that minimise client-side code, reduce bandwidth, and scale cleanly as additional rooms and datastreams are added. In this project, STA thus acts as a lightweight, queryable feature store that underpins both real-time prediction and offline retraining with low latency and high reproducibility.

4.3. Practical Implementation of IFC Preprocessing

This section elaborates on the specific preprocessing steps implemented to prepare the IFC model for use within the developed system. As discussed in the Methodology chapter (subsection 3.5.3), preprocessing ensures that the IFC data are consistent in schema, spatial reference, and attribute structure so that geometric and semantic information can be accurately extracted. In this thesis, preprocessing was carried out using a combination of `IfcOpenShell`, `IfcPatch`, and Python-based utility scripts developed for georeferencing and attribute harmonization.

4.3.1. Overview of the Input IFC Model

The original model was provided in the `IFC2X3` schema and represented a multi-room building geometry without explicit geospatial referencing. Although `IFC2X3` stores approximate latitude and longitude metadata in `IfcSite.RefLatitude` and `IfcSite.RefLongitude`, it lacks formal definitions for the coordinate reference system (CRS) or map conversion parameters. Consequently, the model could not be directly integrated with spatial datasets or used for location-sensitive computations such as solar inflow analysis. To overcome these limitations, several sequential preprocessing operations were applied, as described below.

4.3.2. Schema Migration

The first stage involved upgrading the IFC schema from `IFC2X3` to `IFC4`. This migration was not required for tool compatibility, since both schemas are supported by `IfcOpenShell`, but was performed to leverage the enhanced geospatial and property definitions available in `IFC4`. The conversion provided the following advantages:

- Support for explicit CRS definitions through the entities `IfcProjectedCRS` and `IfcMapConversion`, which enable true georeferencing of building models.
- Access to standardized property sets such as `Qto_SpaceBaseQuantities.GrossVolume`, improving the consistency of quantitative attribute extraction.
- Better interoperability with downstream processes, including visualization, solar analysis, and integration with geospatial data sources (e.g. converting IFC4 to GeoJSON for visualisation and validation within QGIS)

The migration was executed using `IfcConvert` (part of the `IfcOpenShell` toolkit) with default parameters.

4.3.3. Coordinate Transformation and Georeferencing

Following schema migration, the model was georeferenced to the Dutch national coordinate system (Rijksdriehoekscoördinaten, EPSG:28992). This was achieved using the `SetWorldCoordinateSystem` recipe from `IfcPatch`, which applies translation and rotation transformations to align the IFC model with its real-world geographic location.

The transformation parameters were defined using an auxiliary script `update_site_info.py`, which extracted geographic coordinates from the `IfcSite` entity and computed the translation offsets. The script then inserted an `IfcMapConversion` entity referencing a local CRS definition (`IfcProjectedCRS`). The resulting georeferenced model, `BK_v6_ifc4_georef.ifc`, was converted to GeoJSON using the `ifc2gis` web converter developed by CityGeometrix[1]. This tool enables browser-based conversion of IFC files to GIS-compatible formats such as GeoJSON while allowing users to specify the target coordinate reference system. The exported GeoJSON file was subsequently verified in QGIS against the RD New (EPSG:28992) grid to confirm correct spatial alignment.

4.3.4. Orientation Correction

Although the model was now georeferenced, its local coordinate system was not aligned with true north. To ensure accurate calculation of solar azimuths and irradiance values, the model was rotated by 135° clockwise based on orientation analysis conducted in QGIS. This correction was incorporated into the `IfcPatch` transformation recipe and validated by comparing window azimuths before and after rotation. The corrected model, `BK_v6_ifc4_georef_transformed.ifc`, served as the baseline for all subsequent spatial computations.

4.3.5. Attribute Standardization and Validation

After spatial adjustments, property sets were standardized to maintain consistency across schema versions and to ensure that required attributes were properly populated and retrievable by parsing scripts. The following checks were performed:

- Verification that each `IfcSpace` contained a volume property under either `BaseQuantities.GrossVolume` or `Qto_SpaceBaseQuantities.GrossVolume`.
- Confirmation that every `IfcWindow` included surface area data under `BaseQuantities.Area`.
- Validation of the boolean flag `Pset_WindowCommon.IsExternal`, which identifies whether a window faces the exterior environment.
- Normalization of property-set naming conventions where vendor-specific IFC exports deviated from standard IFC nomenclature.

Custom validation functions written in Python parsed the IFC file and produced logs identifying missing or inconsistent attributes. This quality check process ensured that the final dataset conformed to the expectations of the feature extraction workflow described in subsection 3.5.3.

[1]`https://citygeometrix.com/ifc2gis/`

4.3.6. Output and Integration

The final output, referred to as the **Cleaned IFC**, preserved both the geometric hierarchy and the complete set of relevant semantic attributes while being fully georeferenced and properly oriented. This model was subsequently used as input for:

1. Feature-extraction routines implemented in `ifc_parsers.py` and `ifc_calculators.py`.
2. Solar inflow computation based on window orientation, surface area, and material properties.
3. Visualization within the Cesium-based dashboard for interactive exploration of the model and results.

4.4. Solar Inflow Calculation

4.4.1. Inputs, Data Structures, and Dependencies

The computation is implemented in Python using `pvlib` for solar geometry and irradiance, and `IfcOpenShell` for IFC parsing. Custom data structures are defined in `ifc_parsers.py` as follows:

- `BoundingBox` class: defines axis-aligned bounding limits for rooms and windows, with fields `x_min`, `x_max`, `y_min`, `y_max`, `z_min`, and `z_max`.
- `Window` class: represents an individual window instance with attributes `global_id`, `area`, `SHGC`, `tilt`, `azimuth`, `is_external`, and an associated `bounding_box`.
- `Room` class: represents an individual room instance with attributes `global_id`, `short_name`, `long_name`, `volume`, and a `bounding_box`; it also maintains a list of associated `Window` objects.
- `Site` class: represents the overall building site, storing site-level metadata including `latitude`, `longitude`, `elevation`, and `timezone` (set to `Europe/Amsterdam`); it maintains a dictionary mapping room identifiers to their corresponding `Room` objects.

Key helper functions used throughout:

- `extract_site_details(ifc_path)` — reads `IfcSite` (`RefLatitude`, `RefLongitude`, `RefElevation`) and returns a `Site`.
- `compute_bounding_box(shape)` — builds a `BoundingBox` from `ifcopenshell.geom.create\_shape` vertices.
- `true_north_deg(model)` — obtains the building's true-north rotation (used to align azimuths).
- `window_tilt_az_from_placement(win, tn_deg)` — derives per-window *tilt* and *azimuth* from the placement chain, corrected to true north.

4.4.2. Summary of Functions and Script Responsibilities

The codebase is structured around four primary scripts that together enable streaming of solar inflow to the prediction workflow. The execution begins at the FastAPI interface where user selection (room LongName) is passed as an input, proceeds through the simulation controller, and finally invokes the geometry and irradiance computation modules.

- `main.py`: **FastAPI Interface**

  This script serves as the entry point for user interaction.

  - `/api/simulate/{room_name}` — a FastAPI endpoint that accepts the long name of a room as a path parameter. Upon invocation, it calls the `predic t\_internal\_temp()` function from `simulator.py`. The endpoint returns a structured JSON response containing the predicted indoor temperature, aggregated room features (volume, solar inflow, external temperature), and window level solar inflow diagnostics. This response is consumed by the Cesium based web interface, where it is parsed and visualized for the user.

- `simulator.py`: **Prediction Orchestrator**

  This module acts as the middleware between the API layer and the analytical scripts.

  - `get_current_external_temp(site, timestamp)` — retrieves the latest external temperature reading from the weather station API corresponding to the given timestamp and site coordinates. Returns a single floating-point temperature value in °C.
  - `get_latest_model()` — loads the most recently trained XGBoost model from the server's `model_store` directory. Returns a deserialized `XGBRegressor` object for inference.
  - `predict_internal_temp(room_name)` — serves as the primary controller for XGBoost-based indoor temperature prediction. It accepts the long name of a room (`room_name`) as input, which is used to identify the corresponding `IfcSpace` within the building's IFC model. The function internally references the preprocessed IFC file through a globally defined file path constant, `IFC_PATH`, which points to the georeferenced and orientation corrected model prepared during the preprocessing stage. Access to this model is delegated to the parsing routines wherein the function then perform the following sequence:

    1. Invokes `parse_room()` from `ifc_parsers.py`, which loads the IFC model using `ifcopenshell.open(IFC_PATH)`, locates the specified room, and extracts all associated external windows using bounding-box intersection.
    2. For each identified window, calls `window\_solar\_inflow()` from `ifc\_calculators.py` to compute the five-minute solar inflow based on geometric orientation, solar position, and material transmittance parameters.
    3. Aggregates the window-level inflows for all external windows to obtain the total solar inflow for the selected room.
    4. Retrieves the most recent trained XGBoost model and current external temperature.

5. Constructs the feature vector (`[solar\_inflow,room\_volume,external\_temp]`) and performs the indoor temperature prediction.

The function outputs a JSON compatible dictionary containing the predicted indoor temperature, along with diagnostic metadata that includes per-window inflow characteristics and feature-level statistics. These statistics include the actual values of each input feature (solar inflow, room volume, and external temperature) used by the XGBoost model at prediction time.

- `ifc_parsers.py`: **IFC Geometry and Metadata Extraction**

  This script provides the core routines for reading, interpreting, and structuring IFC geometry into usable Python objects.

  – `extract_site_details(ifc_path)` — parses the `IfcSite` entity to extract latitude, longitude, elevation, and sets timezone (`Europe/Amsterdam`). Returns a `Site` object with these fields.

  – `compute_bounding_box(shape)` — constructs an axis-aligned bounding box (AABB) from tessellated geometry produced by `ifcopenshell.geom.create\_shape()`. Returns a `BoundingBox` instance used for spatial intersection tests.

  – `true_north_deg(model)` — reads the building's true-north vector from `IfcGeometricRepresentationContext.TrueNorth`, converts it to a numeric bearing in degrees, and returns the rotation offset used to correct all azimuth calculations.

  – `window_tilt_az_from_placement(win, tn_deg)` — derives per-window tilt and azimuth by resolving the local placement hierarchy (`IfcLocalPlacement` → `IfcAxis2Placement3D`). Returns a tuple `(tilt, azimuth)` corrected to true north.

  – `parse_room(ifc_path, room_name)` — identifies the specified `IfcSpace` using its long name, computes its bounding box, and matches it with window bounding boxes to identify associated external windows. For each matched window, it extracts geometry, area, SHGC, and externality. Returns a `Site` object containing the selected `Room` instance with attached `Window` objects.

- `ifc_calculators.py`: **Solar Geometry and Irradiance Computation**

  This module integrates BIM geometry with solar models implemented using the `pvlib` library.

  – `window_solar_inflow(window, site, timestamp)` — performs the complete solar inflow calculation pipeline for an individual window, integrating geometric parameters from the IFC model with physical models of solar radiation implemented in `pvlib`. The function returns the instantaneous transmitted solar energy through each external glazed surface, expressed in watt-minutes or equivalent units. The major computational steps are as follows:

    1. **Solar position calculation:** The solar azimuth and elevation angles are computed using `pvlib.solarposition.get_solarposition()`,

which internally implements the NREL Solar Position Algorithm (SPA) [53]. Inputs include the site's geolocation (`latitude`, `longitude`, `elevation`) and the timestamp (localized to `Europe/Amsterdam`). Atmospheric pressure is implicitly derived from elevation by the `pvlib` algorithm unless specified otherwise.

2. **Clear-sky irradiance estimation:** Using the `pvlib.location.Location.get_clearsky()` method, clear-sky irradiance components: Global Horizontal Irradiance (GHI), Direct Normal Irradiance (DNI), and Diffuse Horizontal Irradiance (DHI) are computed for the same timestamp. The function employs the Ineichen/Perez clear-sky model [36, 50], which accounts for atmospheric turbidity, air mass, and solar zenith angle, and provides baseline radiation values under cloud free conditions.
3. **Plane-of-array (POA) irradiance computation:** The irradiance incident on the tilted and oriented window surface is then derived using `pvlib.irradiance.get_total_irradiance()`. This step transposes the horizontal irradiance components (GHI, DNI, DHI) onto the window plane based on its `tilt` and `azimuth` angles extracted from IFC geometry (see `ifc_parsers.py`). The resulting POA irradiance (`poa_global`) captures the combined effect of direct, diffuse, and ground-reflected radiation specific to that window surface.
4. **Transmitted solar energy integration:** The total energy transmitted through the window glazing over a five-minute interval is computed by multiplying the POA irradiance by the window area and its Solar Heat Gain Coefficient (SHGC). Mathematically, this corresponds to:

   $$S_{\Delta t} = I_{\text{POA}} \cdot A_g \cdot \tau \cdot \Delta t$$

   where I_{POA} is the plane-of-array irradiance, A_g is the window area, τ is the transmittance (SHGC), and $\Delta t = 300\ \text{s}$ represents the 5-minute integration period. The output represents the instantaneous solar inflow in watt-minutes per window.
5. **Result packaging:** The function returns a structured tuple `(area, SHGC, tilt, azimuth, inflow)`, which encapsulates both geometric descriptors and the computed inflow value. These results are later aggregated at the room level by `predict_internal_temp()` in `simulator.py`.

4.4.3. Room Selection and Window Association

Given a user-selected room, the endpoint `/api/simulate/{room_name}` in `main.py` calls `simulator.predict_internal_temp(room_name)`, which builds the geometric context via `ifc_parsers.parse_room(ifc_path, room_name)`.

(i) Room lookup In `parse_room`, the IFC model is opened, `true_north_deg(model)` is enforced, and all `IfcSpace` entities are scanned. The target space is found by case insensitive match on either `Space.LongName` ("long name") or `Space.Name` ("short name") to the selected identifier. The room's `GrossVolume` is retrieved from `BaseQuantities.GrossVolume` (or `Qto_SpaceBaseQuantities.GrossVolume` in IFC4).

(ii) Room bounding box Using `ifcopenshell.geom.create_shape`, a tessellated shape is created for the `IfcSpace` and passed to `compute_bounding_box` to derive axis-aligned bounds.

(iii) Candidate windows. All `IfcWindow` entities are enumerated. For each window:

1. A shape is created and a `BoundingBox` computed.
2. Orientation is derived via `window_tilt_az_from_placement`, yielding *tilt* (0° up, 90° vertical) and *azimuth* clockwise from true north.
3. Attributes are read from property sets: `BaseQuantities.Area` for *area*; `Pset\_WindowCommon.IsExternal` for the *external* flag; glazing *SHGC* from the vendor-specific `AnalyticalProperties(Type)` set (when present).

(iv) Window–room association (bounding-box test) To associate windows to the selected room in IFC models lacking explicit `IfcRelSpaceBoundary` (like in the case of this project), the room's bounding box is compared to each window's bounding box. Windows whose bounding boxes intersect/overlap the room's bounding box (within a small tolerance) are attached to the room object. Only windows marked `IsExternal = TRUE` are retained for solar inflow.

4.4.4. Solar Geometry and Irradiance per Window

Per window inflow is computed in `ifc_calculators.py` via the routine `window\_solar\_inflow(window,site,timestamp)`, which takes a single timestamp (the end of a fixed five-minute interval) and returns the tuple `(area, SHGC, tilt, azimuth, inflow)`. The steps are:

1. **Solar position (SPA).** With `pvlib.solarposition.get_solarposition`, solar zenith/azimuth are computed from `site.latitude`, `site.longitude`, and `site.elevation` at the given timestamp (timezone from `site.timezone`).
2. **Clear-sky and transposition.** Clear-sky components (GHI/DNI/DHI) are obtained using the Ineichen/Perez family of models; `pvlib.irradiance.get\_total\_irradiance` then projects these to the window's plane-of-array using its *tilt* and *azimuth* (already corrected to true north).
3. **Plane-of-array irradiance to inflow.** Plane-of-array irradiance ($\mathrm{W/m^2}$) is multiplied by `window.area` ($\mathrm{m^2}$), the glazing `SHGC` (dimensionless), and the fixed duration (300 s) to obtain energy in joules over the five-minute interval:

$$S_{\Delta t} \;=\; I_{\mathrm{POA}} \times A_g \times \mathrm{SHGC} \times \Delta t$$

If a window's `SHGC` is missing, a conservative default is applied.

Room-Level Aggregation and Time Binning

In `simulator.predict_internal_temp`, once the `Site` and its single `Room` are constructed:

1. The current timestamp (for prediction) or scheduled batch timestamps (for training) are generated in the site timezone.
2. For each external window in the room, `window_solar_inflow` is called; per-window inflows are accumulated into a room-level sum for the five-minute interval.
3. For training, one five-minute inflow value is associated with each *hourly* record (i.e., a five-minute representative within the hour), producing a weekly table of hourly rows per room.

The routine returns:

- total solar inflow for the room over the last 5 minutes,
- the room's `GrossVolume`,
- per-window diagnostics (area, SHGC, tilt, azimuth, inflow),
- and the latest external temperature (see below).

4.4.5. Edge Cases, Fallbacks, and Validation

- **True North** `true_north_deg(model)` must be available; the code enforces its presence before deriving azimuths. The `true_north_deg(model)` function retrieves the building's rotation relative to geographic north. Its enforcement guarantees that computed window azimuths are globally aligned, preventing directional bias in solar inflow estimation.
- **Property set variability** Volume is read from `BaseQuantities.GrossVolume` (IFC2X3) or `Qto_SpaceBaseQuantities.GrossVolume` (IFC4). Window area is read from `BaseQuantities.Area`. Externality is checked via `Pset\_WindowCommon.IsExternal`. SHGC is read from the vendor set `Analytical Properties(Type)` when present.
- **Window association** Owing to the absence of `IfcRelSpaceBoundary`, a bounding-box intersection is used. This is robust for typical envelope windows; corner cases (e.g., curtain walls spanning multiple rooms) are flagged for manual review.
- **Temporal alignment.** Five-minute inflow is computed at the timestamp used for the hourly record; for real-time prediction, the most recent five-minute interval is used, consistent with the deployment described in section 3.5.

4.4.6. Reproducibility and Scheduling

For training, weekly datasets are prepared with one five-minute inflow value aligned to each hourly record per room, yielding three room-wise hourly series over the week. The cleaned table is used to train and serialize the XGBoost model; the latest model is loaded at prediction time. This scheduling mirrors the data flow described in section 3.5 and ensures that five-minute solar dynamics are captured consistently across training and deployment.

4.5. Machine Learning Model Implementation

This section describes how training data are assembled, how the model is fit and versioned, and how prediction is performed at runtime.

Problem Formulation

The prediction task is formulated as learning a mapping from environmental and geometric factors to indoor temperature. Let $y_{r,t}$ denote the indoor temperature for room r at time t. The system models

$$y_{r,t} \;=\; f\big(T_t^{\text{ext}},\, S_{r,t},\, V_r\big) \;+\; \varepsilon_{r,t},$$

where T_t^{ext} is the external temperature at time t, $S_{r,t}$ is the five-minute solar inflow aggregated to the hour for room r, V_r is the room volume, and $\varepsilon_{r,t}$ is an error term. The function $f(\cdot)$ is learned using gradient-boosted decision trees (XGBoost) on rooms with sensors. Once f is estimated, it can be applied to *non-instrumented* rooms by supplying their IFC-derived volume, computed solar inflow, and the outdoor temperature, thereby producing a prediction for the indoor temperature of the room.

4.5.1. Data Assembly and Feature Construction

For the creation of successive model versions within the training loop, historical indoor temperature observations were fetched from the FROST SensorThings endpoint for each monitored room and merged into a unified dataframe. For each observation timestamp, external temperature is retrieved at the site location using the TU Delft Green Village Davis weather station Kafka client; five-minute solar inflow is computed by summing window-level inflows for the selected room (see section 4.4). The helper routines `fetch_all_sensor_data()`, `fetch_external_temp()`, `calculate_total_solar_inflow()`, and `prepare_training_data()` orchestrate these steps and return a DataFrame with columns `[timestamp,room_name,internal_temp,external_temp,volume,solar_inflow]`.

4.5.2. Model Specification and Training

A scikit-learn `Pipeline` wraps a `StandardScaler` and an `XGBRegressor` (`xgboost==3.0.5`). The input feature set comprises `[external_temp,volume,solar_inflow]`, representing the external environmental condition, the geometric characteristic of the room, and the dynamic solar gain derived from window irradiance, respectively. These variables collectively capture the principal physical drivers of indoor thermal behavior. The target variable, `internal_temp`, corresponds to the observed indoor temperature obtained from the Netatmo sensor datastreams. The data are split 80/20 for train/test, the pipeline is fit, and the mean squared error (MSE) is reported on the hold-out set. A date-stamped model artifact (e.g., `xgb_pipeline_YYYYMMDD.joblib`) is persisted to `xgboost_models/` for reproducibility and rollbacks [14].

4.5.3. Model Versioning and Persistence

Each training run saves a new, immutable pipeline file; the latest file is loaded for inference by the runtime service. This simple, file-based model store enables weekly (or ad-hoc) updates without changing inference code.

4.5.4. Runtime Inference Path

During runtime inference, the FastAPI endpoint `/api/simulate/{room_name}` triggers the machine learning prediction workflow. Upon receiving the request, the simulator retrieves the geometric and environmental context required by the model: the room volume is extracted from the IFC model, the most recent five-minute solar inflow is computed using `ifc_calculators.py`, and the current external temperature is obtained from the weather API. These parameters are then assembled into the standardized feature vector `solar_inflow`, `room_volume`, `external_temp` , matching the schema used during model training.

The regression pipeline comprises a `StandardScaler` and `XGBRegressor`. The model is loaded from the model store, which resides on the same server instance that hosts the FastAPI application. This ensures that inference runs on the most recent trained model without requiring external dependencies or manual updates. The model files are stored as serialized artifacts (`.joblib`) in a designated directory, versioned by training date, and automatically loaded during runtime for prediction. This ensures consistency between the preprocessing transformations applied during training and those executed during prediction. In both training and prediction cycles, an optimal XGBoost configuration, identified through extensive model tuning, is applied. The selection of input features and XGBoost hyperparameters was derived from the best-performing configuration obtained through systematic experimentation in the standalone optimization script `xgboost_training.py`, available in the project's GitHub repository (`https://github.com/vidushi711/BKviewer_FastAPI`). This tuning process established the final learning rate, tree depth, and regularization parameters used in the deployed model. The final training and prediction configuration employed this optimized regressor within a unified scikit-learn pipeline, ensuring that the same preprocessing, scaling, and feature ordering were preserved across both stages. The pipeline was serialized using `joblib` and stored alongside the live application code on the FastAPI server, enabling seamless version alignment between the deployed model and the inference workflow.

The model output is returned as a structured JSON containing the predicted value (`predicted_temp`) and the corresponding feature diagnostics (`solar_inflow`, `room_volume`, `external_temp`), which facilitate interpretability and allow visualization in the web interface. Static assets such as bounding boxes and IFC geometry metadata are served separately under `/IFC_BB` to support contextual display of the model results.

4.5.5. Retraining Schedule

Training jobs generate weekly tables with one entry per hour per room and, upon completion, produce a newly trained XGBoost model version that reflects the additional week of data. Empirical evaluations presented in section 6.2 confirmed that expanding the training dataset by incorporating successive weeks of sensor observations consistently enhanced model performance and stability. This iterative retraining approach enables the regression model to capture evolving environmental patterns and seasonal variations more effectively. The inference service automatically loads the latest model version, ensuring that the deployed model remains aligned with the most recent data.

5
Data Analysis

This study worked with data across two major segments:

1. **Sensor Data:** Collected from the three Netatmo Weather Stations (NWS03) with indoor modules. The focus was primarily on internal temperature readings, which serve as the target variable for the machine learning model. However, supplementary data such as noise levels were also examined to better understand sensor behavior and room conditions.
2. **Input Features for the Prediction Model:** In addition to the cleaned sensor data, the XGBoost-based prediction model relies on input features that are calculated within the setup. These include dynamically computed solar inflow (based on timestamp and geometry) and static room volume.

Each of these stages is analyzed in this chapter to uncover patterns, identify limitations, and evaluate the predictive reliability of the workflow.

5.1. Sensor Data Analysis

Sensor data analysis was conducted on around 34000 data points belonging to the three rooms (with unique timestamps ranging from 23 March 2025 to 31 July 2025).

5.1.1. Initial Cleaning and Generalizations

The indoor sensor records environmental data at 10-minute intervals; however, for the purposes of this analysis, the data was downsampled to retain only one reading per hour per sensor from the three rooms and includes the following six fields: `timestamp`, `room_name`, `internal_temp`, `external_temp`, `volume`, and `solar_inflow`.

The internal temperature varies between 17.4°C and 30.3°C, with a mean of 23.6°C. External temperatures during this period range from as low as 3.11°C to as high as 36.7°C, reflecting seasonal changes and day-night cycles. The room volume (intentionally) captures variation with 123 m^3, 220 m^3 and 440 m^3, and solar inflow spans from 0 (at night or low-sunlight conditions) up to 8.78 million (unit-less simulated metric based on window geometry and orientation).

5.1.2. Relationship between Variables

To explore relationships between the input features and the target variable (internal temperature), line charts and scatter plots were generated for initial understanding of the three input features.

- **External Temperature vs Internal Temperature:** This relationship is visibly positive, showing that indoor temperature tends to rise with outdoor conditions. Each room displays a distinct linear trend, suggesting that indoor temperature responds differently to outdoor conditions depending on room-specific factors such as location, volume and exposure. Plots below illustrates thermal behavior of each room relative to outdoor conditions through a sampled diurnal trend for a day in June and through monthly averaged values (per hour) for the month of June.

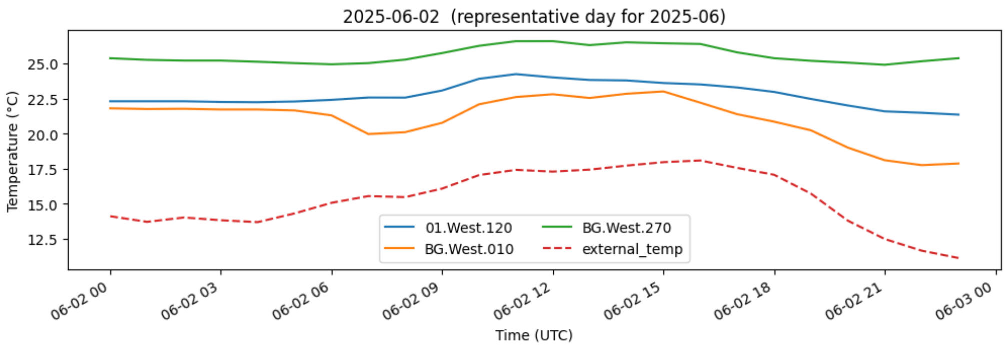


Figure 5.1: Random sample - diurnal temperature profile for June 2025, comparing external temperature with internal temperatures recorded in three rooms of the BK building (01.West.120, BG.West.010, BG.West.270) on 02 June 2025.

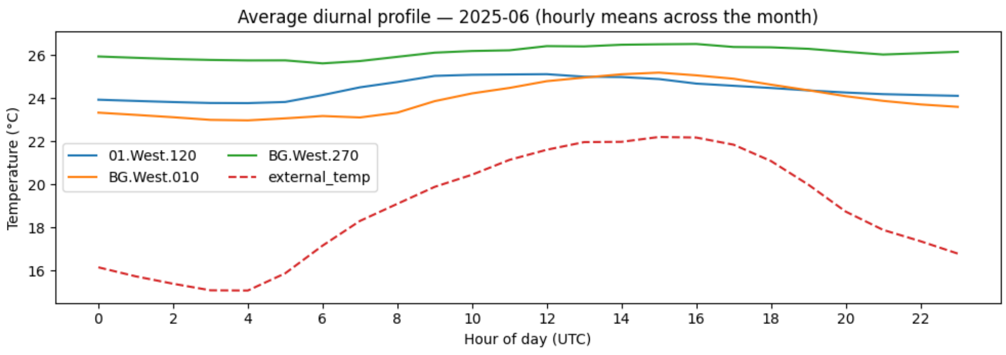


Figure 5.2: Monthly Average (for each hour) - representing daily temperature profile for June 2025

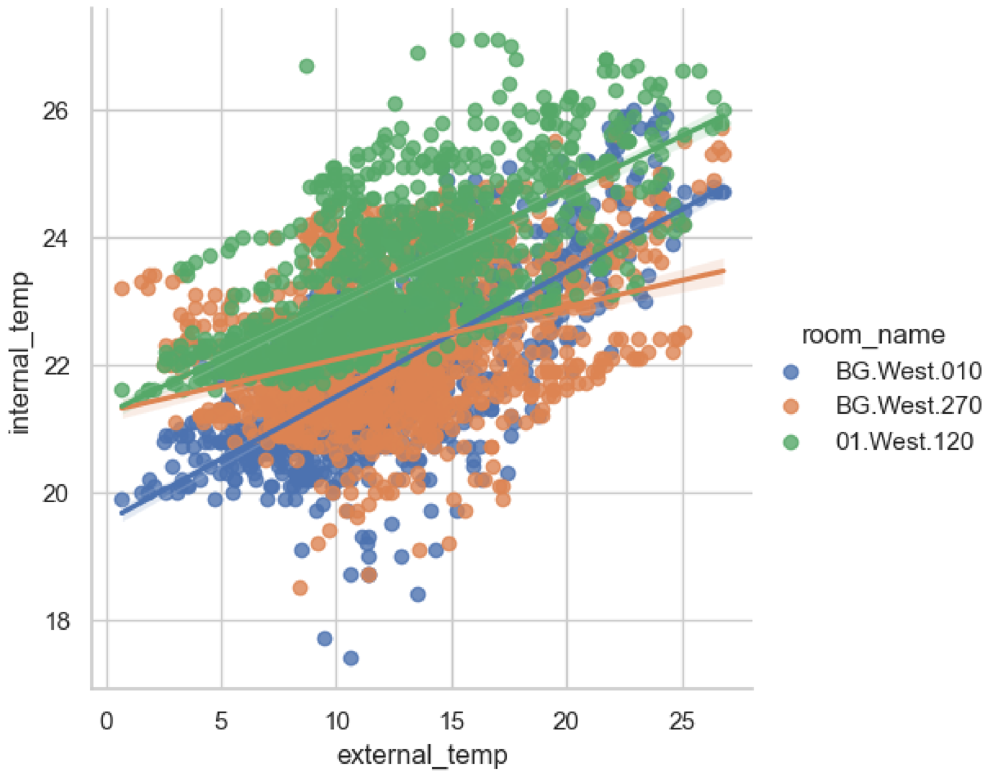


Figure 5.3: Scatter plot showing the relationship between external and internal temperature across three rooms

- **Room Volume vs Internal Temperature:** The scatter plot in Figure 5.3 illustrates a clear positive relationship between external and internal temperature across the three rooms, indicating that indoor conditions rise broadly in tandem with outdoor variations. However, the chart alone does not directly reveal how room volume influences these dynamics. As shown in subsection 5.2.1, linear correlation analysis confirms that external temperature is the strongest predictor of internal conditions, with room volume appearing secondary in comparison. Yet, this perspective changes when examined through the lens of machine learning. subsection 5.2.2 demonstrates that room volume emerges as the most influential feature in the predictive model, underscoring its role as a critical driver in explaining variance once feature interactions and non-linearities are taken into account.

  An additional comparison at 5 a.m., shown in Figure 5.4, highlights that indoor temperatures remain consistently above the external baseline even when solar inflow effects are minimised overnight. This indicates the influence of thermal buffering and possibly HVAC regulation in maintaining higher indoor stability. Moreover, differences between the rooms are evident: BG.West.010 consistently shows lower internal values compared to 01.West.120, hinting at the role of architectural and operational factors in shaping early-morning indoor conditions. These observations reinforce the need for feature-level analysis to disentangle the specific contribution of room volume relative to external drivers.

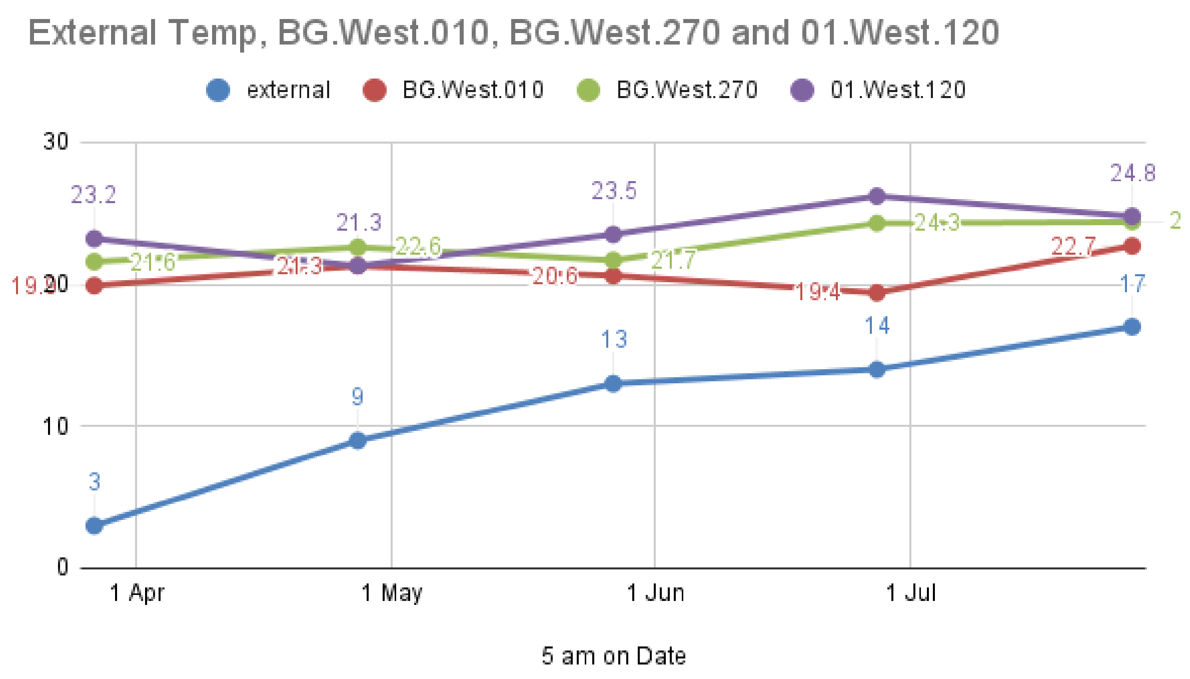


Figure 5.4: External and internal temperatures at 5 a.m. from March to July, showing stable indoor conditions compared to rising outdoor temperatures

- **Solar Inflow vs Internal Temperature:** This relationship is non-linear and scattered. At lower solar inflow levels, indoor temperature shows minimal variation, but as inflow increases, so does temperature—though not uniformly. This supports the hypothesis that solar inflow contributes to heating, but its influence varies with other contextual factors such as time of day and room orientation.

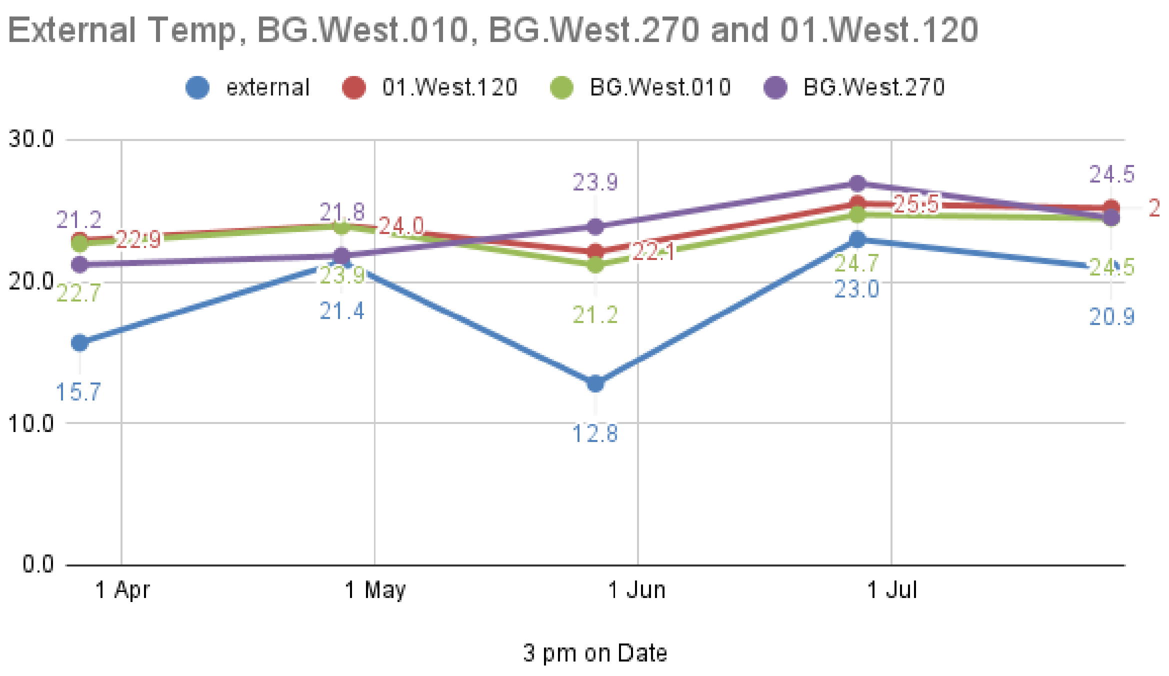


Figure 5.5: Influence of solar inflow on change in internal temperature

Room-specific effects: As shown in Figure 5.3, the linear trends differ across rooms, with steeper slopes in some cases indicating stronger external influence. Notably, Room 378 (BG.West.010), despite having the largest volume, displays a relatively strong correlation between external and internal temperature. This suggests that factors other than volume—such as window orientation, window-to-wall ratio, or lack of shading—may be contributing to greater solar exposure and heat gain. In contrast, Room 81 (01.West.120) exhibits a flatter trend, indicating more stable internal conditions likely due to lower solar inflow or better insulation. These differences indicate a need to include geometric and contextual features, not just volume, in the prediction model.

Time-of-day influence (based on solar inflow variation): While not explicitly plotted, the distribution of solar inflow values across timestamps reveals diurnal patterns. Inflow values spike during midday hours, especially in rooms with south-facing or unobstructed windows, and drop to near zero overnight.

Thermal inertia and delayed response: While solar inflow values exhibit clear diurnal peaks—typically highest around midday—the corresponding changes in internal temperature do not follow instantaneously. Instead, a noticeable lag is observed, with indoor temperatures continuing to rise even after solar inflow begins to decline in the late afternoon. This phenomenon is indicative of thermal inertia: the tendency of building materials and air volume within a space to absorb, store, and slowly release heat over time.

Thermal inertia varies by room depending on factors like volume, surface area-to-volume ratio, material properties, and window exposure. In this dataset, rooms with larger volumes or better insulated envelopes demonstrate a more gradual temperature response compared to those with more direct sunlight exposure or less thermal mass. For example, on March 23, 2025, in Room BG.West.010, solar inflow peaked between 13:00 and 14:00, but the internal temperature continued to rise until after 15:00, despite a declining inflow trend.

This lagged behavior highlights the importance of modeling internal temperature as a function of both static characteristics (such as room volume and geometry) and dynamic variables (like solar inflow and external temperature) over time. The combination enables the model to capture both immediate drivers of change and the accumulated thermal effects that unfold more slowly—justifying the inclusion of these features in a non-linear, feature-aware model like XGBoost.

Interaction effects: Initial exploration indicates possible interaction effects between volume and solar inflow. For example, rooms with high solar inflow and small volumes tend to heat up faster, while larger-volume rooms show dampened peaks. This interaction is subtle and not easily modeled by linear regression—supporting the use of tree-based ensemble models like XGBoost that capture non-linear dependencies without explicitly coding interaction terms.

5.2. Input Features for the Prediction Model

This section details the specific features used in training the predictive model and evaluates their statistical and model-based influence on the target variable - internal room temperature. Each input was selected based on domain knowledge, availability, and potential relevance to thermal behavior at room scale. These include environmental factors sourced from sensors and public weather datasets, as well as spatial attributes derived from the building's IFC model. To better understand their roles, both linear correlation analysis and model-derived feature importance are examined. Together, these analyses offer complementary insights into how different predictors contribute to learning and generalization in the context of room level temperature forecasting.

Features used in this project include **room volume** (extracted directly from the IFC model of the BK building), **solar inflow** (calculated using IFC-derived room attributes and external libraries), and external environmental variables such as **temperature, humidity, wind speed**, and **pressure**. These external conditions were sourced partly from public APIs and from the monthly data files shared by Davis Weather Station at the TUD Green Village [59]. The target variable, **internal temperature**, was measured through Netatmo sensing devices, as described earlier in the section.

To assess how these predictors contribute to the modeling task, two analyses were performed. First, a correlation check examines linear associations: (i) between input features and internal temperature, and (ii) among input features themselves (multicollinearity). This provides an overview of simple statistical dependencies and verifies that predictors are not overly redundant. Second, feature importance is derived from the trained XGBoost model, which captures non-linear relationships and feature interactions, offering insight into how the model internally prioritizes predictors during learning.

5.2.1. Correlation Check (Statistical Association)

Pearson's correlation coefficient r was used as a measure of linear association between variables. It is defined as:

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\,\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}},$$

where x_i and y_i are paired observations of two variables, and $\bar{x}$ and $\bar{y}$ are their respective means. The value of r ranges from -1 to $+1$, where positive values indicate that higher values of one variable tend to coincide with higher values of the other, and negative values indicate the opposite. The closer $|r|$ is to 1, the stronger the linear relationship; values near 0 suggest little or no linear association. It is important to note that Pearson's r captures only linear effects: a non-linear U-shaped relationship would still yield a low correlation. It is important to note that Pearson's r measures only pairwise linear associations. This means it cannot capture interaction effects, where the influence of one feature depends on the value of another. For example, internal temperature may rise much more rapidly when both external temperature and solar irradiance are high at the same time. Such combined effects are not visible in simple correlations, but are picked up later through the XGBoost model, which can account for non-linearities and interactions.

Correlation between Input features and Internal Temperature: Figure 5.6 shows the Pearson correlations between each input feature and internal temperature. The strongest positive associations are observed for external temperature ($r \approx 0.46$) and solar irradiance ($r \approx 0.34$), consistent with the expectation that indoor climate is strongly influenced by outdoor conditions. Wind speed and pressure show very weak correlations (r close to 0), while room volume and humidity are negatively correlated ($r \approx -0.38$). This is also intuitive: larger room volumes and higher humidity slow down temperature rise. In summary, external temperature and solar irradiance appear to be the primary linear drivers of indoor temperature.

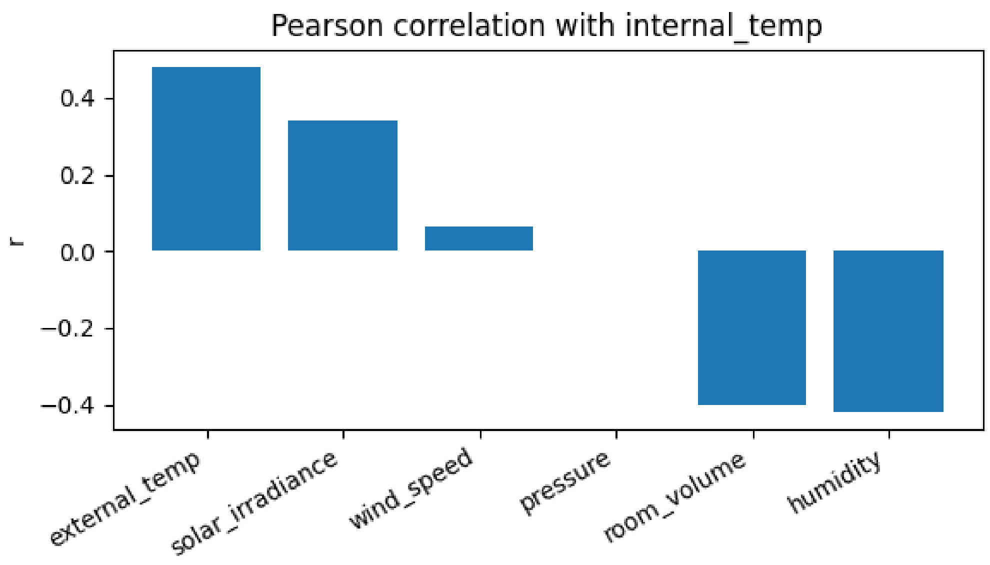


Figure 5.6: Pearson Correlation values for input features with internal temperature

As Figure 5.6 depicts, the features bearing strongest correlation with internal temperature are external temperature followed by solar irradiance. This verifies common logic as the internal parameters of built space are heavily influenced by its immediate environment. Wind speed and Pressure have almost no correlation while Room volume and humidity are negatively correlated - this too bolsters common logic as larger room volumes and higher humidity would contribute to slower temperature increase. In conclusion, the takeaway from here is that external temperature and solar irradiance are the main 'linear' drivers for internal temp.

Inter-feature Correlations (Pearson) - Low Multicollinearity: Multicollinearity refers to strong linear relationships among predictor variables, which can reduce the interpretability of a machine learning model and in some cases lead to instability in the estimation of feature effects. While tree-based ensemble methods such as XGBoost are generally more robust to correlated inputs than linear regression, excessive redundancy between predictors can still reduce model efficiency, as the algorithm may repeatedly split on near-duplicate features without gaining additional information. A common heuristic is that correlations above $|0.9|$ indicate problematic collinearity, suggesting that the variables may not contribute distinct predictive power [19].

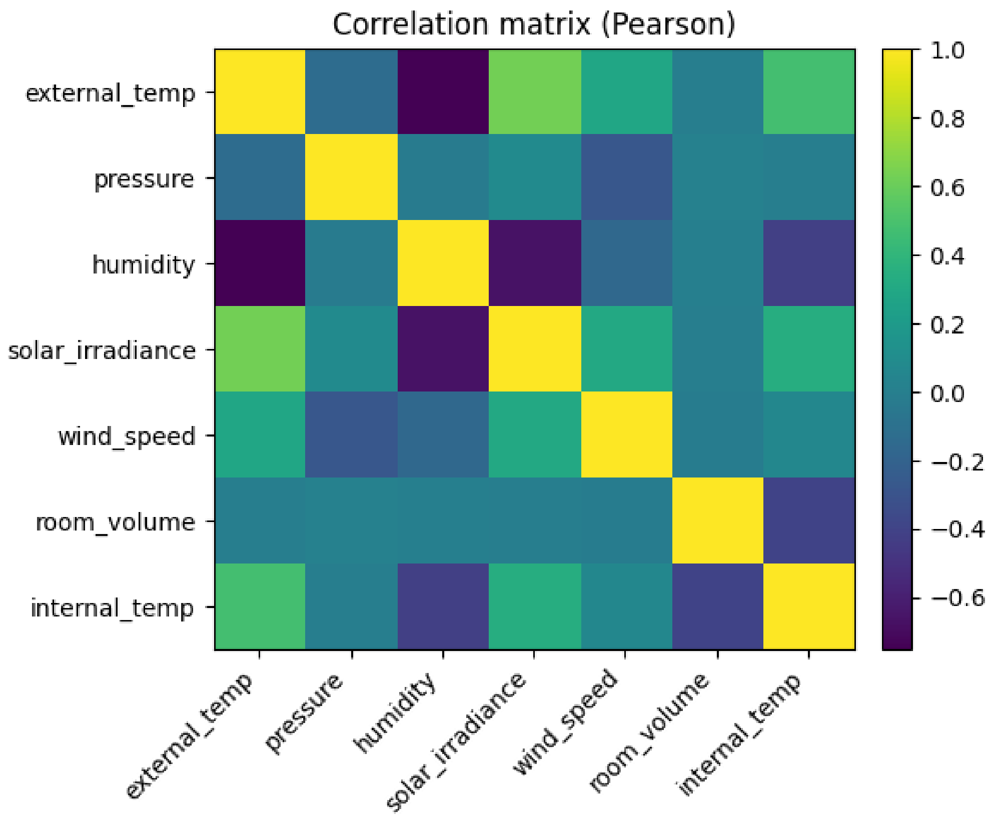


Figure 5.7: Multicollinearity in Input Features

In this dataset, most pairwise correlations lie well below the problematic threshold of $|0.9|$ [19]. The strongest associations observed are of moderate magnitude: external temperature with solar irradiance ($r \approx 0.65$) and humidity with external temperature ($r \approx -0.62$).

Other correlations remain weaker, typically between -0.3 and 0.4. This pattern indicates that the predictors are not overly redundant, and each feature contributes unique information to the model, which is desirable for both generalization and interpretability.

5.2.2. Feature Importance (Model-based Association)

Beyond simple statistical correlations, feature importance values from the XGBoost model provide insight into how the trained model actually utilizes predictors. Unlike Pearson's r, which is restricted to linear effects, XGBoost can capture non-linearities and higher-order interactions, thereby assigning importance to features even when their pairwise correlation with the target is weak. For example, a variable with near-zero correlation might still be critical if it interacts with another predictor in a non-linear way.

Figure 5.8 displays the feature importances obtained from the trained model. Room volume emerges as the most influential feature, followed by external temperature and humidity. This highlights the fact that spatial characteristics of the room (captured by its volume) play a decisive role in determining internal temperature, complementing the effects of external climate factors. Other features such as pressure, solar irradiance, and wind speed contribute to a lesser extent but still provide incremental predictive value.

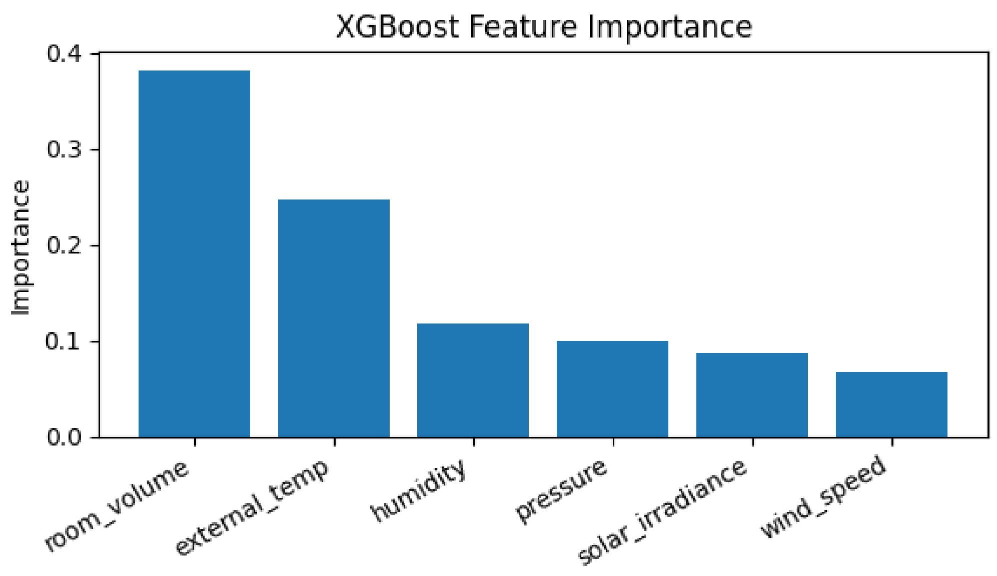


Figure 5.8: XGBoost feature importance values for input features

6 Results

This chapter presents the results of the prototype system developed in this thesis. The aim here is to demonstrate how the system has performed in predicting indoor temperature under the chosen configuration of features and model parameters. Different sections explore variations in input features and hyperparameter settings, providing insights into how these choices affect predictive accuracy. The final deployed model has been selected based on an optimal combination of input features and the best performing hyperparameters. Importantly, the system has been designed to continue ingesting new data on a periodic basis, and as shown in subsection 6.2.3, the availability of more training data has shown to improve predictive accuracy over time.

6.1. Overall Performance of the Final Model

Figure 6.1 shows the scatter plot comparing actual indoor temperature values with those predicted by the final XGBoost model under the selected configuration. Each point represents a prediction–observation pair, and the diagonal line indicates the ideal $y = x$ relationship, where predicted values would perfectly match actual values.

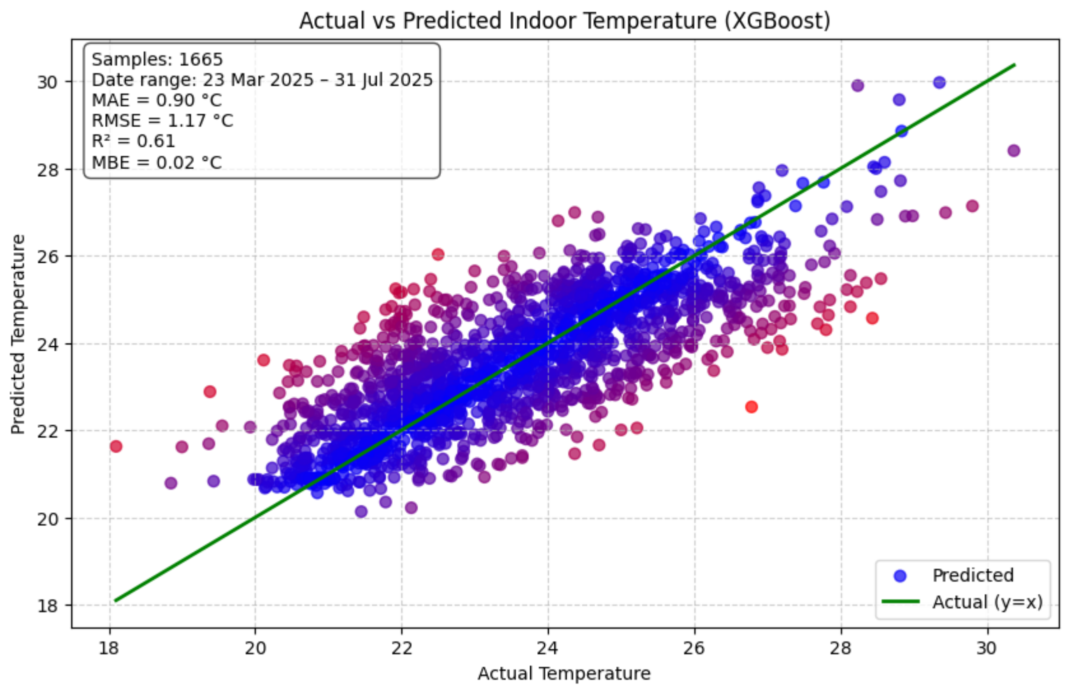


Figure 6.1: Overall performance of the final XGBoost model: predicted indoor temperature vs. actual temperature for samples in evaluation period

As the figure illustrates, the predictions generally align well with the actual indoor temperatures, with most points clustered around the $y = x$ line. While some deviations occur, particularly at higher temperature ranges where underestimation is more evident, the overall trend indicates that the model captures the relationship between external drivers (external temperature, solar inflow, and room volume) and indoor conditions effectively. The statistical metrics (MAE = 0.90 °C, RMSE = 1.17 °C, $R^2 = 0.61$) confirm that the model provides a reasonably accurate approximation of indoor thermal dynamics, especially given the limited sensor deployment and relatively short data collection window. This validates the feasibility of using an open, standards-based framework for reliable room-level predictions in uninstrumented spaces.

6.2. Evaluation of XGBoost Model

The third and final research question in this study was: *"Can a simulation model based on building and sensor data be used to predict environmental conditions in non-instrumented spaces?"* To assess the reliability of the predictions made by the XGBoost model, three accuracy metrics commonly used for evaluating regression models were computed: Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and the coefficient of determination (R^2). These metrics were calculated using the `scikit-learn` functions `mean_squared_error()`, `mean_absolute_error()`, and `r2_score()` respectively, applied to the held-out test subset of data.

1. **Root Mean Squared Error (RMSE)** The RMSE quantifies the average magnitude of prediction error, giving higher weight to larger deviations due to squaring.

It is defined as:

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

where y_i is the observed value, $\hat{y}_i$ is the predicted value, and n is the total number of test samples. A lower RMSE value indicates that predictions are, on average, closer to the true observations. Because it squares the residuals, RMSE penalizes large errors more strongly and is therefore sensitive to outliers.

2. **Mean Absolute Error (MAE)** The MAE measures the average absolute difference between predicted and observed values:

   $$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

   Unlike RMSE, MAE treats all deviations linearly, making it less sensitive to outliers. It represents the mean magnitude of errors in the same units as the target variable (°C in this case).

3. **Coefficient of Determination (R^2 Score)** The R^2 score, or coefficient of determination, measures how well the predicted values approximate the actual data. It is given by:

   $$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

   where $\bar{y}$ denotes the mean of the observed values. An R^2 value of 1 indicates perfect prediction, 0 indicates that the model performs no better than predicting the mean, and negative values indicate that the model performs worse than a mean predictor. In this project, R^2 was computed using the `r2_score()` function from `scikit-learn`. The occurrence of negative R^2 values in certain cases (e.g., under *Additional Training Data*) reflects instances where residual variance exceeded the variance of the observed data, typically due to limited or unaligned samples.

Each metric was calculated on the test subset only, ensuring that performance values reflect the model's ability to generalize to unseen data. Together, RMSE and MAE provide absolute measures of prediction error, while R^2 expresses the proportion of variance in indoor temperature explained by the model.

Building on these metrics, an extensive series of experiments was carried out to examine how the predictive accuracy of XGBoost responds to different modeling conditions. The focus was on three critical dimensions:

- **Different Hyperparameters** of the XGBoost model
- **Additional Input Features** - to check the variation in prediction accuracy metrics when additional features are added
- **Additional Training Data** - to check variation in accuracy as more and more training data was made available to the model

In each case, one dimension was varied systematically while the others were held constant, allowing the isolated effect of that factor on model performance to be observed. The following sections present the resulting plots and analyses, highlighting how these variations influenced the accuracy of predictions.

6.2.1. Tuning Hyperparameters of the XGBoost model

XGBoost provides a range of hyperparameters that govern how the ensemble of boosted trees is built and how well it generalizes to unseen data. Hyperparameter tuning is the process of systematically adjusting these values to optimize predictive performance [14]. Among the most influential hyperparameters for gradient boosted regression trees are the following:

- **Learning rate** (also called *eta* or shrinkage) controls the contribution of each tree to the overall model. Smaller values (e.g. 0.01–0.1) slow down learning but usually improve generalization by preventing the model from fitting noise, whereas larger values risk overfitting [14, 37].
- **Number of estimators** ($n\_estimators$) specifies the maximum number of trees to be built. Larger ensembles can capture more complex relationships but also increase training time and may overfit if not balanced by a sufficiently low learning rate [24].
- **Tree depth** (max_depth) controls how many splits each tree can make from the root to the leaves. Shallow trees (depth 3-4) are less prone to overfitting, while deeper trees can capture stronger nonlinearities but risk memorizing noise [14].
- **Gamma** is a regularization parameter that sets the minimum reduction in loss required for a further split. Higher values prune weak splits, promoting simpler trees and reducing overfitting risk [14].

The literature emphasizes that the optimal configuration is dataset-specific. In practice, grid search or random search combined with validation strategies is commonly employed to explore the hyperparameter space [10, 51]. For this study, a small but representative set of hyperparameters was chosen based on prior work and domain intuition (see Section 5.2).

Experimental setup. In these experiments, the predictive task was to estimate indoor temperature using the six selected features: *room volume, external temperature, pressure, humidity, solar irradiance, and wind speed*. Training was carried out on data from January to April 2025, while May 2025 was held out for testing. A manual sweep over learning rate, maximum depth, and gamma values was performed, with the number of trees adjusted proportionally (larger ensembles for lower learning rates). The resulting test set accuracies are reported below:

Table 6.1: Hyperparameter tuning results with six input features (Jan-Apr train, May test). Best results are highlighted in bold.

Run	Learning rate	Max depth	Gamma	$n\_estimators$	RMSE	MAE	R^2
6	0.03	6	1.0	1500	**1.357**	1.101	**0.185**
5	0.10	6	1.0	800	1.373	1.109	0.165
3	0.03	3	0.0	1500	1.405	1.125	0.125
2	0.10	3	0.0	800	1.429	1.144	0.096
4	0.10	6	0.0	800	1.455	1.166	0.062
1	0.30	3	0.0	300	1.467	1.166	0.046

Results: The best performance (RMSE ≈ 1.36, $R^2 \approx 0.18$) was achieved with a low learning rate (0.03), deeper trees (max_depth = 6), and moderate regularization (gamma = 1.0). This confirms the general finding that small learning rates combined with larger ensembles provide more stable models [24]. Higher learning rates (0.30) performed notably worse, and shallow trees (depth = 3) underfit the data. The addition of gamma regularization improved performance compared to unregularized runs, highlighting its importance for pruning weak splits. Overall, these experiments indicate that model accuracy is sensitive to hyperparameter settings, and careful tuning can significantly improve predictive reliability.

6.2.2. Additional Input Features

In supervised learning, the number and type of input features play a critical role in determining predictive performance. While adding more features can in principle supply the model with richer information, it may also introduce redundancy, noise, or spurious correlations that hinder generalization. In tree-based ensembles such as XGBoost, irrelevant or weakly relevant features can lead to unnecessary splits and reduced model efficiency [27, 39].

From a theoretical perspective, this effect is closely linked to the *bias–variance trade-off* [34]. A model with too few features may exhibit high bias, failing to capture the relationships in the data. Conversely, a model with too many features, particularly if they are weakly correlated with the target, can have high variance, overfitting to noise in the training data and performing worse on unseen data. Regularization methods are designed to mitigate such issues by controlling complexity, but feature selection remains a critical factor [11, 33].

The first set of experiments, summarized in Table 6.2, compared model accuracy when only external temperature was used versus when additional features were appended. Counter to expectations, accuracy degraded as more features were added: RMSE rose and R^2 fell below zero, indicating performance worse than a naive mean predictor.

Table 6.2: XGBoost evaluation with different input features (Jan–Apr train, May test).

Input Features	n_train	n_test	RMSE	MAE	R^2
external temperature	3291	823	1.448	1.269	-0.062
external temperature + volume	3291	823	1.516	1.340	-0.164
external temp + vol + pres + hum + solar + wind	3291	823	1.629	1.411	-0.344

This initially raised the question: is accuracy reduced simply due to the number of features, or because the additional predictors are less relevant to the target? To test this, the experiment was repeated using a reversed ordering of features, ensuring that the strongest drivers (external temperature and volume) were included later in the sequence. Results are shown in Table 6.3.

Table 6.3: XGBoost evaluation with solar-focused feature ordering.

Input Features	n_train	n_test	RMSE	MAE	R^2
solar irradiance	3291	823	1.399	1.179	0.009
solar irr. + wind + pressure	3291	823	1.515	1.271	-0.162
solar irr. + wind + pres + ext. temp + vol + hum	3291	823	1.671	1.443	-0.414

Even when highly relevant features such as external temperature were introduced later, overall performance continued to decline as the number of predictors increased. This confirmed that the degradation was not merely due to feature relevance, but rather to the accumulation of features itself, which introduced variance without improving bias.

Bias-variance perspective: The observation that adding predictors degraded test accuracy, despite XGBoost's ability to down-weight weak signals can be understood through classic generalization theory. With a limited sample size, increasing the dimensionality of the input space enlarges the hypothesis search space and typically raises the variance of the learned model; unless new features substantially reduce bias, overall generalization can worsen [28, 11]. This phenomenon is closely related to the *Hughes (peaking) effect*: for a fixed number of training samples, accuracy often rises with dimensionality up to an optimum and then declines as additional, low-signal features are added [30, 31]. In short, when data are relatively scarce, a few strong predictors can outperform a larger, noisier set.

Why tree ensembles may still overfit with extra features Although boosted trees can ignore irrelevant variables by simply not splitting on them, the **greedy** split search still evaluates many candidate thresholds across all available features. With small n, this increases the chance of selecting spurious splits that fit noise, inflating variance. XGBoost includes regularization levers like row/feature subsampling, depth limits, ℓ_2/ℓ_1 penalties, and a minimum loss reduction (`gamma`)—to mitigate this, but they do not eliminate variance induced by weak predictors under limited data [14]. Empirically and theoretically, careful feature selection remains complementary to regularization [38].

In large industrial settings with millions of rows, hundreds of engineered features can help because the sample size is sufficient to estimate many interactions while regularization keeps variance in check. In contrast, in datasets on the order of less than 10 thousand records (as in this study), adding weak or redundant features often raises variance more than it reduces bias, yielding poorer test performance [28, 31].

Focused sweep with three features. To further investigate, hyperparameter tuning was repeated with only the three most relevant predictors: *room volume, external temperature, and solar irradiance*. Results are summarized in Table 6.4. Consistent with the arguments above, this reduced, high-signal feature set achieved better generalization than the six-feature configurations tested earlier.

Table 6.4: Hyperparameter sweep results with three input features (Jan–Apr train, May test). Best results are highlighted in bold.

Run	Learning rate	Max depth	Gamma	$n\_estimators$	RMSE	MAE	R^2
6	0.03	6	1.0	1500	**1.336**	1.121	**0.210**
5	0.10	6	1.0	800	1.341	1.125	0.204
3	0.03	3	0.0	1500	1.361	1.138	0.180
2	0.10	3	0.0	800	1.381	1.152	0.155
1	0.30	3	0.0	300	1.382	1.153	0.153
4	0.10	6	0.0	800	1.420	1.176	0.106

Results: Surprisingly, the reduced feature models performed better overall than their six-feature counterparts. The best three-feature configuration (RMSE $\approx$ 1.34, $R^2 \approx 0.21$) outperformed the best six-feature configuration (RMSE $\approx$ 1.36, $R^2 \approx 0.18$). This suggests that the inclusion of additional predictors such as humidity, wind speed, and pressure added variance without improving the bias of the model. This finding highlights that for this dataset and with the aim of accurate prediction, a smaller but carefully chosen feature set yields more reliable predictions.

Final model configuration Based on the series of experiments, the most effective setup combined a conservative **learning rate (**0.03**)**, moderately deep trees **(**$max\_depth = 6$**)**, and regularization via $\gamma = 1.0$, with the ensemble capped at approximately 1500 trees. Equally important, feature selection proved critical: the strongest and most parsimonious model relied on only three predictors **room volume, external temperature, and solar irradiance**. These features captured both spatial (room-level) and environmental (outdoor and solar-driven) influences on indoor temperature, while avoiding the variance inflation introduced by weaker predictors such as humidity, pressure, or wind speed. This configuration therefore forms the basis for subsequent simulation and forecasting tasks.

6.2.3. Additional Training Data

This subsection has to be redone with corrected data - Please ignore for now - June test data was off

A core hypothesis of this framework is that prediction quality should improve over time as more sensor data accumulate. In other words, as the training window grows, the model ought to generalize better to new periods. Ideally, this effect is strongest over long horizons (multiple seasons), when the model can learn seasonal patterns; however, the time frame of this thesis allows the scope for only month-scale increments. To isolate the effect of additional training data, a fixed common test period was set and compared models trained on progressively longer windows.[1]

Using the six-feature input set (*room volume, external temperature, pressure, humidity, solar irradiance, wind speed*) and the ideal model configuration (XGBoost, $max\_depth = 6, n\_estimators = 1500, learning\_rate = 0.03, subsample = 0.8, colsample\_bytree = 0.8$, $\lambda = 1.0$, `tree_method=hist`, γ=`min_split_loss` $= 1.0$), *June 2025* period was fixed as the test set with increasing training data:

- **Jan–Mar** (train end: 2025-03-31),
- **Jan–Apr** (train end: 2025-04-30),
- **Jan–May** (train end: 2025-05-31),

always ensuring the training data ended strictly before the test start (to prevent leakage).

Table 6.5: Effect of adding training data with a fixed June 2025 test set (six features, same model settings).

Scenario	Train end	n_train	n_test	RMSE	MAE	R^2
Jan–Mar	2025-03-31	396	3	1.742	1.211	-0.396
Jan–Apr	2025-04-30	2025	3	1.888	1.497	-0.639
Jan–May	2025-05-31	4114	3	**0.729**	**0.668**	**0.756**

Results and interpretation. With the very small June test slice available in this dataset ($n\_test = 3$ after `dropna`), two patterns emerge:

1. Extending the training window from *Jan–Mar* to *Jan–Apr worsened* RMSE by about 8.4% (from 1.742 to 1.888), with R^2 remaining negative. Given the tiny test sample and possible distribution shift between March and April, this fluctuation likely reflects variance rather than a true degradation.
2. Extending further to *Jan–May substantially* improved accuracy: RMSE drops by $\sim 58\%$ vs. Jan–Mar and by $\sim 61\%$ vs. Jan–Apr (to 0.729), and R^2 rises to 0.756. This is consistent with the hypothesis that more training data (especially including late-spring patterns) improves generalization to June.

[1]Using a *fixed* test window across scenarios avoids confounding changes in seasonality. If each training window were paired with a different test month, shifts in weather regimes could masquerade as model improvements or degradations unrelated to training size.

Caveat on reliability. Because the fixed June test window collapses to only three valid observations after missing-value filtering, the reported metrics are statistically fragile. A single difficult point can swing RMSE and R^2 markedly. For a more stable estimate of the "more data helps" effect, it would be preferable to:

1. enlarge the test horizon (e.g., full month with adequate coverage across all features),
2. or use rolling/blocked cross-validation over weeks, keeping temporal order intact,
3. and report uncertainty (e.g., bootstrap CIs across days).

Take-away. Within the limits of the available test coverage, the *Jan–May* model generalizes best to June. This aligns with the project's premise: as additional months of data accumulate, the model can learn a richer set of patterns (including evolving spring conditions), which—once the test set is sufficiently sized—should translate into steadily improving out-of-sample accuracy.

7 Conclusion and Future Scope

This chapter summarizes the main findings of the study, reflects on its methodological and technical contributions, and outlines directions for future research and development. The work presented in this thesis demonstrated the development and applicability of an interoperable system that integrates Building Information Modeling (BIM), Internet of Things (IoT) sensor data, and machine learning through open standards to derive geospatial insights on indoor environmental conditions. The system, in its present configuration, can be used to analyze spatial and temporal heat patterns across rooms within a building. Furthermore, by extending the data model to include information on building materials, the same framework could be used to study the relationship between construction properties and indoor environmental performance.

Towards the end of this study, an additional experiment was conducted to extend the existing workflow for retrieving indoor temperature data from the FROST server to also extract CO_2 concentration and noise levels using the same query logic. A rapid evaluation of the results revealed clear linkages between CO_2 fluctuations and occupancy patterns, which aligned with practical knowledge of room usage such as the number of seats and the absence of activity during weekends. These findings illustrate the versatility of the developed architecture and confirm its potential as a scalable, interoperable system that can be packaged as a practical tool for broader geospatial analysis and environmental monitoring.

7.1. Summary of Findings

The central objective of this research was to explore whether environmental conditions in non-instrumented spaces can be predicted using information derived from BIM geometry, sensor data from instrumented rooms, and meteorological observations. The study addressed this objective through the development of a modular workflow that connects IFC-based room geometry, the OGC SensorThings API (STA) implemented via FROST, and an XGBoost regression model hosted on a FastAPI-based application.

The findings can be summarized in relation to the three research questions formulated in Chapter 1:

- **RQ1 — Integration of IoT data with BIM context:** The implementation of the SensorThings API provided a scalable and interoperable mechanism to manage and query time-series sensor observations. The use of FROST with a PostGIS backend enabled each room to be semantically linked with its corresponding IoT datastreams, allowing indoor temperature, CO_2, and noise data to be directly associated with their spatial representation in the IFC model.
- **RQ2 — Extraction and use of geometry-based features:** The developed IFC preprocessing pipeline successfully extracted geometric and semantic features such as room volume, window area, tilt, azimuth, and true-north orientation. These parameters were critical for calculating solar inflow using `pvlib`, demonstrating how spatial attributes can be transformed into meaningful predictors for thermal modeling.
- **RQ3 — Predictive reliability of the model:** The XGBoost regression model achieved consistent performance across rooms, identifying solar inflow, external temperature, and room volume as the most influential parameters. Despite limited data availability, the model achieved stable error metrics and demonstrated potential for generalizing to non-instrumented rooms based on their IFC-derived features.

Overall, the developed system validated that open-standard data exchange, spatial feature computation, and ensemble-based regression can together support data-driven building performance analysis.

7.2. Key Technical Contributions

A major technical strength of this research lies in the efficiency of its data architecture. By adopting the SensorThings API standard, heavy computational operations such as temporal filtering, ordering, and aggregation were delegated to the server. Instead of downloading large, unfiltered datasets, the FROST server processed queries at the database level using parameters such as:

```
?$orderby=phenomenonTime desc&$filter=phenomenonTime ge ... and phenomenonTime le ...
```

This server-side approach reduced the size of data transfers and significantly improved performance. During development, client-side filtering for one month of data required over twenty minutes to execute, while the same query handled by the FROST server returned results in approximately twelve seconds. This efficiency demonstrates the feasibility of using open APIs for real-time monitoring and near-instantaneous data access, even in multi-sensor environments.

The system also demonstrated strong interoperability. By linking IFC entities to STA datastreams, it bridged spatial and temporal domains within a unified, standards-compliant framework. The Cesium-based visualization further enabled interaction with live sensor feeds and simulation results, transforming the workflow into a foundational prototype for smart building digital twins.

7.3. Limitations

While the developed workflow proved effective for a pilot-scale deployment, several limitations were observed. First, the IFC model lacked explicit `IfcRelSpaceBoundary` relationships, requiring window–room associations to be inferred through bounding-box intersection, which may be less accurate in complex geometries. Second, the training dataset covered a limited time span, capturing short-term variations but not full seasonal cycles, which may limit the model's ability to capture long-term trends. Third, the regression model was trained on a narrow set of environmental variables (temperature, solar inflow, and volume), without accounting for internal gains, occupancy, or material properties, which could further improve predictive robustness. Finally, data sparsity from IoT devices and occasional sensor outages introduced small temporal gaps that required interpolation.

Despite these constraints, the results demonstrate the proof-of-concept viability of the workflow and its capacity for extension.

7.4. Future Scope

The developed system provides a flexible foundation for further research and scaling. Several opportunities exist to advance the current framework:

(i) Expansion of the data and sensor network

Future deployments could extend the SensorThings database to include additional observed properties such as humidity, occupancy, and energy consumption. Automating the registration of new IoT sensors in FROST through API scripts would enable continuous integration of new data streams without manual configuration.

(ii) Model enhancement and retraining pipeline

The XGBoost model can be extended into an online or incremental learning pipeline where the model automatically retrains as new data are received through the STA endpoint. Incorporating multi-sensor inputs (e.g., CO_2, noise, humidity) and static IFC-derived features (e.g., wall material, glazing ratio) would improve predictive performance and enable multi-variable forecasting of comfort indicators.

(iii) System scaling and interoperability

The current integration between IFC, STA, and FastAPI can be expanded into a generalizable digital twin architecture applicable to other buildings or campuses. Aligning the workflow with emerging standards such as CityJSON or the Digital Twin Definition Language (DTDL) could enable interoperability with broader urban-scale modeling environments.

(iv) Interactive visualization and decision support

The Cesium-based interface can evolve into a dynamic feedback system that visualizes real-time conditions while allowing users to test hypothetical scenarios, such as altered glazing configurations or HVAC settings. Such extensions could transform the system from a predictive analytics tool into a decision-support environment for facility managers and urban planners.

7.5. Closing Remarks

This thesis contributes a reproducible and standards-based framework that integrates BIM, IoT, and machine learning for predictive environmental analysis. The study demonstrates that open data protocols such as the SensorThings API, when combined with spatially rich IFC data and efficient model architectures, can yield scalable and responsive digital workflows.

Beyond its immediate findings, the research establishes a methodological foundation for future work on interoperable, data-driven digital twins for buildings and campuses. With further expansion, the presented workflow can serve as a basis for adaptive control, sustainability assessment, and simulation-based decision-making in the built environment.

References

[1] Tariq Ahmad, Haibo Chen, and Yulong Guo. “A comprehensive overview on the data-driven and physics-based approaches for building energy prediction”. In: *Energy and Buildings* 144 (2017), pp. 152–173.

[2] Amir H. Alavi et al. “Internet of Things-enabled smart cities: State-of-the-art and future trends”. In: *Measurement* 129 (2018), pp. 589–606. DOI: 10.1016/j.measurement.2018.07.067. URL: https://www.sciencedirect.com/science/article/pii/S0263224118306912.

[3] Zaid Allam, A Kassem, and A Elagouz. “A Lightweight IoT Architecture for Indoor Temperature and Humidity Forecasting Using Machine Learning Algorithms”. In: *International Journal of Computer Applications* 174.15 (2021), pp. 16–21.

[4] Zaid Allam, Ahmed Kassem, and Ahmed Elagouz. “A Lightweight IoT Architecture for Indoor Temperature and Humidity Forecasting Using Machine Learning Algorithms”. In: *International Journal of Computer Applications* 174.15 (2021), pp. 16–21.

[5] Anonymous. “Analysis of factors influencing indoor thermal environment in passive residential buildings”. In: *Building and Environment* (2023).

[6] Anonymous. “Impact of solar radiation on indoor thermal comfort near highly glazed façades”. In: *Building and Environment* (2023).

[7] Authors of that article. “BIM-based semantic enrichment and knowledge graph generation via ...” In: *ScienceDirect / [Journal name]* (2025). Contains detail about geometry, spatial relationships and semantic data in BIM.

[8] ASHRAE. *ASHRAE Handbook—Fundamentals*. American Society of Heating, Refrigerating and Air-Conditioning Engineers, 2021.

[9] Autodesk. *What is BIM? Building Information Modeling – Autodesk*. Accessed: 2025-07-02. 2025. URL: https://www.autodesk.com/solutions/aec/bim.

[10] James Bergstra and Yoshua Bengio. “Random search for hyper-parameter optimization”. In: *Journal of Machine Learning Research*. Vol. 13. Feb. 2012, pp. 281–305.

[11] Christopher M Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[12] Leo Breiman. “Random forests”. In: *Machine learning* 45 (2001), pp. 5–32.

[13] buildingSMART International. *Industry Foundation Classes (IFC) Overview*. https://technical.buildingsmart.org/standards/ifc/. Accessed: 2025-09-14. 2020.

[14] Tianqi Chen and Carlos Guestrin. “XGBoost: A scalable tree boosting system”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2016), pp. 785–794. DOI: 10.1145/2939672.2939785.

[15] Drury B Crawley et al. “EnergyPlus: creating a new-generation building energy simulation program”. In: *Energy and Buildings* 33.4 (2001), pp. 319–331.

[16] Giovanni Desogus et al. “BIM and IoT Sensors Integration: A Framework for Consumption and Indoor Conditions Data Monitoring of Existing Buildings”. In: *Sustainability* (). DOI: 10.3390/su13084496. URL: https://doi.org/10.3390/su13084496.

[17] Giuseppe Desogus et al. “BIM and IoT Sensors Integration: A Framework for Consumption and Indoor Conditions Data Monitoring of Existing Buildings”. In: *Sustainability* 13.8 (2021), p. 4496.

[18] TU Delft Staff Directory. *Prof. Dr. Ir. P.J.M. van Oosterom*. Accessed May 2025. URL: https://www.tudelft.nl/staff/p.j.m.vanoosterom/.

[19] Carsten F. Dormann et al. “Collinearity: a review of methods to deal with it and a simulation study evaluating their performance”. In: *Ecography* 36.1 (2013), pp. 27–46. DOI: 10.1111/j.1600-0587.2012.07348.x. URL: https://doi.org/10.1111/j.1600-0587.2012.07348.x.

[20] John A. Duffie and William A. Beckman. *Solar Engineering of Thermal Processes*. 4th ed. John Wiley & Sons, 2013.

[21] Chuck Eastman et al. *BIM Handbook: A Guide to Building Information Modeling for Owners, Managers, Designers, Engineers and Contractors*. 2nd. John Wiley & Sons, 2011. ISBN: 9780470541371.

[22] ESRI. *GIS in Smart Buildings*. Accessed: 2025-09-12. 2024. URL: https://www.esri.com/en-us/industries/buildings/overview.

[23] Panagiotis Fragkos et al. “Energy system impacts and policy implications of the European Intended Nationally Determined Contribution and low-carbon pathway to 2050”. In: *Energy Policy* 100 (2017), pp. 216–226. DOI: 10.1016/j.enpol.2016.10.023. URL: https://doi.org/10.1016/j.enpol.2016.10.023.

[24] Jerome H Friedman. “Greedy function approximation: A gradient boosting machine”. In: *Annals of Statistics*. Vol. 29. 5. 2001, pp. 1189–1232.

[25] GDMC Group. *Geospatial Data Management Centre (GDMC)*. Accessed May 2025. URL: https://www.gdmc.nl/.

[26] Michael F. Goodchild. “Citizens as sensors: the world of volunteered geography”. In: *GeoJournal* 69 (2007). Seminal article on spatial data integration and user-contributed spatial information, pp. 211–221.

[27] Isabelle Guyon and André Elisseeff. “An introduction to variable and feature selection”. In: *Journal of Machine Learning Research*. Vol. 3. Mar. 2003, pp. 1157–1182.

[28] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2nd ed. Springer, 2009. URL: https://hastie.su.domains/ElemStatLearn/ (visited on 09/02/2025).

[29] Tianzhen Hong et al. “Advances in research and applications of energy-related occupant behavior in buildings”. In: *Energy and Buildings* 116 (2018), pp. 694–702.

[30] G.F. Hughes. “On the Mean Accuracy of Statistical Pattern Recognizers”. In: *IEEE Transactions on Information Theory* 14.1 (1968), pp. 55–63. DOI: 10.1109/TIT.1968.1054102.

[31] Giovanni Iacca et al. “Implications of the Curse of Dimensionality for Supervised Learning Classifier Systems”. In: *Pattern Analysis and Applications* 21.3 (2018), pp. 1193–1205. DOI: 10.1007/s10044-017-0649-0. URL: https://link.springer.com/article/10.1007/s10044-017-0649-0 (visited on 09/02/2025).

[32] IBM Research. *IBM Machine Learning Topics: Decision Trees, Random Forest, Boosting, Overfitting*. https://www.ibm.com/think/topics/. Accessed: 2025-08-02. Individual articles: Decision Trees, Random Forest, Boosting, Overfitting. 2023.

[33] IBM Research. *What is Regularization?* https://www.ibm.com/think/topics/regularization. Accessed: 2025-09-02. 2023.

[34] IBM Research. *What is XGBoost?* https://www.ibm.com/think/topics/xgboost. Accessed: 2025-09-02. 2023.

[35] Imran, Nadeem Iqbal, and Hyoung-Kyu Kim. “IoT Task Management Mechanism Based on Predictive Optimization for Efficient Energy Consumption in Smart Residential Buildings”. In: *Energy and Buildings* 257 (2022), p. 111762. DOI: 10.1016/j.enbuild.2021.111762. URL: https://www.sciencedirect.com/science/article/pii/S037877882101046X.

[36] Pierre Ineichen and Richard Perez. “A new airmass independent formulation for the Linke turbidity coefficient”. In: *Solar Energy* 73.3 (2002), pp. 151–157.

[37] Guolin Ke et al. “LightGBM: A highly efficient gradient boosting decision tree”. In: *Advances in Neural Information Processing Systems*. Vol. 30. 2017.

[38] Ron Kohavi and George H. John. “Wrappers for Feature Subset Selection”. In: *Artificial Intelligence* 97.1–2 (1997), pp. 273–324. DOI: 10.1016/S0004-3702(97)00043-X. URL: https://machine-learning.martinsewell.com/feature-selection/KohaviJohn1997.pdf (visited on 09/02/2025).

[39] Max Kuhn and Kjell Johnson. *Applied predictive modeling*. Springer, 2013.

[40] TU Delft BK Labs. *VR Lab — TU Delft Faculty of Architecture*. Accessed May 2025. URL: https://www.tudelft.nl/bk/onderzoek/bk-labs/vr-lab.

[41] Steve Liang, Chih-Yuan Huang, and Tania Khalafbeigi, eds. *OGC SensorThings API — Part 1: Sensing*. OGC Implementation Standard. Version 1.0. Open Geospatial Consortium (OGC), Aug. 4, 2016. URL: https://docs.ogc.org/is/15-078r6/15-078r6.html (visited on 08/29/2025).

[42] Steve Liang, Chih-Yuan Huang, and Tania Khalafbeigi. *OGC SensorThings API Part 1: Sensing*. Tech. rep. OGC 15-078r6. Open Geospatial Consortium, 2016. URL: `http://docs.opengeospatial.org/is/15-078r6/15-078r6.html`.

[43] Zhiyong Liu et al. "A review of data-driven approaches for prediction and classification of building energy consumption". In: *Renewable and Sustainable Energy Reviews* 82 (2018), pp. 1027–1047.

[44] John Mardaljevic and Arnab Roy. "Daylight Modeling and SBI Matrices in Parametric Building Simulation". In: *Building Simulation Conference Proceedings* (2021).

[45] C. K. Metallidou, K. E. Psannis, and E. A. Egyptiadou. "Energy Efficiency in Smart Buildings: IoT Approaches". In: *IEEE Access* 8 (2020), pp. 63679–63699. DOI: `10.1109/ACCESS.2020.2984461`. URL: `https://doi.org/10.1109/ACCESS.2020.2984461`.

[46] Genci Mustafaraj, Jiakang Chen, and Grant Lowry. "Forecasting building energy consumption using autoregressive models". In: *Energy and Buildings* 42.10 (2010), pp. 2059–2069.

[47] W Natephra and A Motamedi. "Live Data Visualization of IoT Sensors Using Augmented Reality (AR) and BIM". In: *Proceedings of the 36th International Symposium on Automation and Robotics in Construction (ISARC)*. Banff, Canada, 2019.

[48] Debayan Paul et al. "IoT and Machine Learning Based Prediction of Smart Building Indoor Temperature". In: *2018 4th International Conference on Computer and Information Sciences (ICCOINS)*. IEEE. 2018, pp. 1–6.

[49] Debayan Paul et al. "IoT and Machine Learning Based Prediction of Smart Building Indoor Temperature". In: *2018 4th International Conference on Computer and Information Sciences (ICCOINS)*. IEEE. 2018, pp. 1–6.

[50] Richard Perez et al. "Modeling daylight availability and irradiance components from direct and global irradiance". In: *Solar Energy* 44.5 (1990), pp. 271–289.

[51] Liudmila Prokhorenkova et al. "CatBoost: unbiased boosting with categorical features". In: *Advances in Neural Information Processing Systems* 31 (2018).

[52] pvlib developers. *pvlib.solarposition.get_solarposition Documentation (v0.4.2)*. Accessed 2025-05-03. 2019. URL: `https://pvlib-python.readthedocs.io/en/v0.4.2/generated/pvlib.solarposition.get_solarposition.html`.

[53] Ibrahim Reda and Afshin Andreas. *Solar Position Algorithm for Solar Radiation Applications*. Tech. rep. NREL/TP-560-34302. Accessed 2025-04-27. National Renewable Energy Laboratory (NREL), 2004. URL: `https://research-hub.nrel.gov/en/publications/solar-position-algorithm-for-solar-radiation-applications-2`.

[54] Matthew Reno, Clifford Hansen, and Joshua Stein. *Global Horizontal Irradiance Clear Sky Models: Implementation and Analysis*. Tech. rep. SAND2012-2389. Sandia National Laboratories, 2012.

[55] Per Sahlin and Lennart Eriksson. "TRNSYS—A transient system simulation program". In: *Simulation Series* 36.3 (2004), pp. 113–118.

[56] Trimble Solutions. *What Is BIM? Building Information Modeling*. Accessed: 2025-09-XX. 2024. URL: `https://www.trimble.com/blog/construction/en-US/article/what-is-bim-building-information-modeling`.

[57] Serena Summa et al. “Impact on thermal energy needs caused by the use of different solar irradiance decomposition and transposition models”. In: *Energies* 15 (2022), p. 8904.

[58] *Temperature in buildings—How thermal mass affects indoor temperature stability*. Swegon Knowledge Hub. 2023. URL: `https://www.swegon.com/knowledge-hub/the-indoor-climate-guide/factors-affecting-indoor-climate`.

[59] *TU Delft Green Village Davis Weather Station Data (Monthly NetCDF)*. Accessed via the Ruisdael Observatory data portal. TU Delft, Faculty of Civil Engineering and Geosciences. 2025. URL: `https://ruisdael.citg.tudelft.nl/davis_weather_station/TUD_Green_Village/Monthly_NetCDF/` (visited on 09/02/2025).

[60] Weiming Wang, Steve Liang, and Jihong Zhang. “Using the OGC SensorThings API in the Internet of Things”. In: *Sensors* 19.20 (2019), p. 4487. DOI: `10.3390/s19204487`.

[61] Qiang Xin et al. “A Deep Learning Architecture for Power Management in Smart Cities”. In: *Energy Reports* 8 (2022), pp. 1568–1577. DOI: `10.1016/j.egyr.2022.01.157`. URL: `https://doi.org/10.1016/j.egyr.2022.01.157`.

A

XGBoost Parameters

Booster Parameters: Role and Relevance

This appendix offers a deeper look at the key XGBoost hyperparameters that played a central role in the room temperature prediction task. While the results of hyperparameter tuning are discussed in Section 6.2.1, the focus here is to explain the purpose of these parameters, how they function internally within the XGBoost framework, and how their tuning impacted the overall modeling process.

Parameter Overview

XGBoost parameters can be grouped into three categories:

- **General parameters**: specify booster type and compute settings (e.g., CPU/GPU).
- **Booster parameters**: control the tree structure, depth, and regularization.
- **Task parameters**: define learning objectives and evaluation metrics.

This appendix focuses on **booster parameters**, which were the most actively tuned during experimentation.

Key Booster Parameters

`learning_rate`

(also referred to as eta) This controls how much the model weights are updated at each boosting step. Internally, it's a shrinkage factor applied to the contribution of each new tree. Lower values like 0.03 mean each tree has less influence, so more trees (e.g., 1500) are needed to converge — but this can lead to better generalization. In this thesis, learning rates of 0.30, 0.10, and 0.03 were tested, with 0.03 giving the best results in terms of RMSE and R^2.

`max_depth`

Defines how deep each tree can grow. Deeper trees can capture more complex relationships, but also increase the risk of overfitting. A depth of 3 was sufficient for some configurations, but increasing to 6 improved performance in combination with other controls like gamma.

`gamma (min_split_loss)`

This controls the minimum loss reduction required to make a further partition on a leaf node. Internally, XGBoost computes the gain from splitting a node and compares it to gamma — splits are allowed only if gain > gamma. A value of 0.0 means no regularization by gamma, while a value of 1.0 introduces moderate pruning. In experimentations done as part of Section 6.2.1, gamma=1.0 helped reduce overfitting in deeper trees.

`n_estimators`

Number of boosting rounds or trees. This is not a learned parameter but directly related to learning rate: lower rates need more trees. Here values from 300 to 1500 were used depending on the learning rate.

`subsample`

Fraction of training samples used per boosting round. Setting this < 1 introduces randomness, acting like bagging. A default value of 0.8 was used to help with generalization.

`colsample_bytree`

Fraction of features (columns) randomly selected for each tree. Also set to 0.8 in all runs, it serves a similar purpose to subsample but operates at the feature level.

`reg_lambda`

L2 regularization term on weights. XGBoost penalizes overly confident leaf scores, improving robustness. This wasfixed at 1.0

`tree_method`

Specifies how trees are built. `"hist"` was used, which uses histogram-based splits — this speeds up training and reduces memory usage. It's also recommended when using larger datasets.

These parameters work together behind the scenes to balance model complexity and generalization. XGBoost automatically computes gains, applies regularization, and selects splits based on these parameters. Understanding their logic helped tailor the model to projects's use case — predicting room temperature using external variables like solar irradiance, outside temperature, and room volume — with relatively limited but clean input data.

Experimental Context (for Reference)

The model was trained on data from January–April 2025 and tested on May 2025. Input features used were:

```
['room_volume', 'external_temp', 'solar_irradiance']
```

The limited feature set ensured interpretability and reduced noise. As explained in Section 6.2, the performance impact of the above parameters was discussed in the main report chapters using RMSE, MAE, and R^2 metrics.

XGBoost's flexibility and design made it a powerful choice for this project. While it is a high-performance library capable of handling complex workflows, its clear parameter structure allow effective use even with a basic understanding of regression and machine learning principles. In this project, the ability to directly configure core hyperparameters—such as `learning_rate`, `max_depth`, and `reg_lambda` enabled targeted experimentation.

The specific context of this thesis, including the limited but semantically relevant input features (`room_volume`, `external_temp`, and `solar_irradiance`) and the structured time-series nature of the data, meant that model performance was especially sensitive to the balance between complexity and generalization. Systematic tuning made it possible to identify configurations that delivered accurate predictions without overfitting. Thus, even with a basic setup, XGBoost enabled informed control over the training process and supported reliable model development aligned with the project's goals.