Digital Twins in Process Engineering based on Measurements, Simulations and Neural Networks

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

von der KIT-Fakultät für
Chemieingenieurwesen und Verfahrenstechnik
des Karlsruher Instituts für Technologie (KIT)
genehmigte

Dissertation

von
M. Sc. Dennis Teutscher
aus Balingen

Tag der mündlichen Prüfung: 20. Oktober 2025

Erstgutachter:	PD Dr. rer. nat. Mathias J. Krause
Zweitgutachter:	Prof. Dr.-Ing. Gregor Wehinger

Zusammenfassung

Das rasante Voranschreiten der Digitalisierung und die daher zunehmende Integration von Künstlicher Intelligenz (artificial intelligence, AI) in industrielle Prozesse markieren den Übergang von Industrie 4.0 zu Industrie 5.0. Während Industrie 4.0 den Schwerpunkt auf Automatisierung, Datenaustausch und intelligente Systeme legt, erweitert Industrie 5.0 diese Vision, indem sie sich stärker auf die Zusammenarbeit zwischen Mensch und Maschine sowie Nachhaltigkeit, Widerstandsfähigkeit und Anpassungsfähigkeit konzentriert. Bei dieser Transformation spielt die Technologie des Digitalen Zwillings (Digital Twin, DT) eine wichtige Rolle und ermöglicht eine dynamische Kopplung zwischen physischen Systemen und ihren virtuellen Repräsentationen. Mithilfe Echtzeit-Datenaustausch ermöglichen DTs die kontinuierliche Überwachung, vorausschauende Analyse und Optimierung von industriellen Prozessen.

Obwohl DTs in Branchen wie der Fertigungsindustrie, der Automobilindustrie und dem Gesundheitswesen bereits weit verbreitet sind, existiert im Bereich der mechanischen Verfahrenstechnik noch eine signifikante Forschungslücke. Diese Lücke ist nicht auf mangelndes Potenzial zurückzuführen, sondern vielmehr auf die einzigartigen Herausforderungen, die solche Systeme mit sich bringen. Verfahrenstechnische Prozesse sind von Natur aus multiphysikalisch sowie multiskalisch und beinhalten oft nichtlineare Interaktionen, hohe Rechenanforderungen und einen Mangel an standardisierten digitalen Darstellungen. Im Gegensatz zu Fertigungsstraßen, die in der Regel repetitiv und stark strukturiert sind, umfasst die mechanische Verfahrenstechnik dynamische und nichtlineare Verhaltensweisen - von Mehrphasenströmungen bis hin zu variablen Randbedingungen - was die Modellierung und Vorhersage in Echtzeit erheblich komplexer macht. Hinzu kommt, dass viele industrielle Prozesse immer noch auf älterer Technologie beruhen, deren digitale Konnektivität begrenzt ist und die Integration moderner DT-Architekturen weiter erschwert. In dieser Arbeit wird diese Lücke adressiert und stellt ein modulares DT-Framework vor, das speziell für die Verfahrenstechnik entwickelt wurde. Ziel ist es, eine flexible, erweiterbare Plattform bereitzustellen, die sowohl eine realitätsnahe physikalische Modellierung als auch die Integration moderner datengesteuerter Methoden ermöglicht.

In dieser Arbeit wird der DT als eine Kombination von verschiedenen, lose gekoppelten Modulen verstanden, die flexibel kombiniert und an eine Vielzahl von Anwendungsszenarien angepasst werden können. Zu den primären Modulen, die in dieser Arbeit entwickelt wurden, gehört eine Visualisierungskomponente, ein CFD-basiertes Simulationsmodell für zwei- und dreidimensionale Strömungen sowie ein datengetriebenes Modell auf der Basis eines neuronalen Netzwerks (neural network, NN). Diese Module bilden die Grundlage für vier beispielhafte Anwendungen, die jeweils auf den spezifischen Modulen beruhen und die praktische Durchführbarkeit des Frameworks demonstrieren.

Die erste Anwendung ist ein Visualisierungstool namens OpenLBar, welches CFD-Strömungsdaten mit Hilfe von Augmented Reality über physikalische Umgebungen legen kann. Dies verbessert das räumliche Verständnis und die Interaktion mit komplexen Strömungsphänomenen. Als Teil dieser Entwicklung wurde die modulare Open-Source-Bibliothek OpenVisFlow erstellt, welche eine erweiterbare Integration von Visualisierungen und Geometrie-Overlays auf verschiedenen Plattformen ermöglicht. Studien zur Benutzerfreundlichkeit haben die Zugänglichkeit und Effektivität des Tools sowohl in der Forschung als auch in der Industrie bestätigt und sein Potenzial als menschenzentrierte Schnittstelle für DTs unterstrichen.

Die zweite Anwendung - das Virtual CFD Lab - erweitert das Visualisierungsmodul, indem es mit einem auf der Lattice-Boltzmann-Methode (LBM) basierenden CFD-Modell gekoppelt wird. Dieses Tool ermöglicht Echtzeitsimulationen direkt auf mobilen Geräten durch Scannen und Interpretieren von handgezeichneten 2D-Geometrien. Das System erkennt die Domäne automatisch, initiiert eine Just-in-Time-Simulation und präsentiert die Ergebnisse. Ein Benchmarking mit Standardtestfällen bestätigte sowohl die Genauigkeit der Simulationen als auch die praktische Nutzbarkeit der mobilen Plattform. Diese Anwendung veranschaulicht, wie interaktive und portable DTs den Zugang zu High-Fidelity-Simulationen demokratisieren können.

Die dritte Anwendung demonstriert die Entwicklung eines groß angelegten, physikgesteuerten DT für städtische Umgebungen. Durch die kontinuierliche Einbeziehung von Daten meteorologischer Stationen, einschließlich Windge-

schwindigkeit, -richtung und Schadstoffgehalt, ist der DT in der Lage, die städtische Luftströmung und Schadstoffverteilung in Echtzeit zu simulieren. Ein dynamisch anpassbares CFD-Modell verarbeitet eingehende Messdaten und ermöglicht Langzeitsimulationen zur Ermittlung von Schadstoff-Hotspots und Gebieten mit hohen Windgeschwindigkeiten im gesamten Stadtgebiet. Das Modell umfasst eine automatisierte Schnittstelle für die Geometrieerzeugung, die OpenStreetMap (OSM)-Daten nutzt und einen einfachen Einsatz in verschiedenen Stadtgebieten gewährleistet. Diese Eigenschaft hat einen erheblichen Nutzen für die Stadtplanung und die Umweltüberwachung.

Die vierte Anwendung konzentriert sich auf eine Kammerfilterpresse, eine Kerntechnologie für Fest-Flüssig-Trennverfahren. Hier wird ein DT entwickelt, welcher auf einem NN basiert. Dieses nutzt Sensordaten, die über eine Internet of Things (IoT)-Anlage gesammelt werden. Parameter wie Druck, Durchflussmenge und die benutzerdefinierten Konfigurationen werden kontinuierlich an eine Cloud-Datenbank übertragen, mit dessen Daten das Modell kontinuierlich trainiert wird. Der NN-basierte DT ist in der Lage, die Prozesszeit und die Effizienz der verwendeten Komponenten (z. B. Filtermedium) vorherzusagen, da dieser auf historischen Nutzungs- und Echtzeitdaten basiert. Diese Vorhersagen werden über eine App-Schnittstelle zugänglich gemacht. Die Anwendung mit ihrem integrierten ML-basierten Modul zeigt, dass das Framework in der Lage ist, intelligente Prozessvorhersagen treffen zu können und das potential mit Prozessoptimierung zu unterstützen.

Jedes Modul und jede Anwendung wurde mit gezielten Validierungsmethoden bewertet: Die CFD-Simulationen wurden mit analytischen Lösungen und vorhandenen Solver-Daten verglichen, um numerische Genauigkeit zu gewährleisten, während die neuronalen Netzwerkmodelle anhand von Testdatensätzen validiert wurden, um die Zuverlässigkeit von Vorhersagen zu bewerten. Es wurden Studien zur Benutzerfreundlichkeit durchgeführt, um die Klarheit, die Reaktionsfähigkeit und den pädagogischen Wert der Visualisierungstools zu bewerten. Diese Bewertungen wiesen ein hohes Maß an praktischer Nutzbarkeit auf.

Ein wesentlicher Vorteil, des in dieser Arbeit entwickelten Frameworks, liegt in seiner Interoperabilität. Durch die konsequente modulare Architektur und die Verwendung von standardisiertern Schnittstellen, können die einzelnen Module einfach in bestehende Prozessleitsysteme integriert werden. Gleichzeitig ermöglicht diese Architektur eine hohe Skalierbarkeit, so dass das Framework auf spezifische Prozesse zugeschnitten und auf verschiedene industrielle Kontexte übertragen werden kann. Erwähnenswert ist auch die zukünftige Mög-

lichkeit, datengesteuerte und physikalische Modelle zu kombinieren, um eine adaptivere digitale Darstellung realer Prozesse zu erhalten.

Die Ergebnisse dieser Arbeit zeigen, dass ein modularer, adaptiver Ansatz für DTs eine leistungsfähige, flexible und übertragbare Lösung für die mechanische Verfahrenstechnik darstellt. Die entwickelten Konzepte, Modelle und Anwendungen leisten einen wesentlichen Beitrag zur Weiterentwicklung der DT-Technologie im Kontext industrieller Prozesse. Das vorgestellte Framework bietet nicht nur eine solide Grundlage für zukünftige Forschung, sondern ermöglicht auch die direkte Anwendung in der industriellen Praxis zur Steigerung von Effizienz und Nachhaltigkeit. Langfristig eröffnen DTs neue Perspektiven für die intelligente, ressourceneffiziente und ökonomisch sinnvolle Gestaltung industrieller Systeme.

Abstract

The rapid advancement of digitalization and the increasing integration of Artificial Intelligence (AI) into industrial processes mark the transition from Industry 4.0 to Industry 5.0. While Industry 4.0 emphasizes automation, data exchange, and smart systems, Industry 5.0 expands this vision by focusing more on human-machine collaboration, sustainability, resilience, and adaptability. In this transformation, the role of the technology Digital Twin (DT) plays an important role and enables a dynamic coupling between physical systems and their virtual representations. With the help of real-time data exchange, DTs facilitate continuous monitoring, predictive analysis, and optimization of industrial processes.

Although DTs are already widely used in industries such as manufacturing, automotive, and healthcare, a significant research gap remains in the field of mechanical process engineering. This gap is not due to a lack of potential, but rather the unique challenges posed by such systems. Mechanical process engineering processes are inherently multi-physical and multi-scale, often involving nonlinear interactions, high computational demands, and a lack of standardized digital representations. Unlike manufacturing lines, which are typically repetitive and highly structured, mechanical process engineering encompasses dynamic and nonlinear behaviors—ranging from multiphase flows to variable boundary conditions—making real-time modeling and prediction significantly more complex. In addition, many industrial processes are still relying on legacy equipment, which is limited in digital connectivity and further complicates the integration of modern DT architectures. By recognizing this gap, this work introduces a modular DT framework designed specifically for process engineering with the goal of providing a flexible, extensible platform that enables both high-fidelity physical modeling and the integration of modern data-driven

methods.

In this work, the DT is understood as a combination of various loosely coupled modules that can be flexibly combined and adapted to a wide range of application scenarios. The primary modules developed in this work include a visualization component, a CFD-based simulation model for two- and three-dimensional flow, and a neural network (NN)-based data-driven model. These modules are the foundation for four exemplary applications, each relying on specific modules and demonstrating the practical feasibility of the framework.

The first application is a visualization tool named OpenLBar, which overlays CFD flow data on physical environments using Augmented Reality. This enhances spatial comprehension and interaction with complex fluid phenomena. As part of this development, the modular open-source library OpenVisFlow was created, allowing for extensible integration of visualizations and geometry overlays into various platforms. Usability studies validated the tool's accessibility and effectiveness in both research and industrial settings, emphasizing its potential as a human-centric interface for DTs.

The second application—the Virtual CFD Lab—extends the visualization module by coupling it with a lattice Boltzmann method (LBM)-based CFD model. This tool enables real-time simulations directly on mobile devices by scanning and interpreting hand-drawn 2D geometries. The system automatically recognizes the domain, initiates a just-in-time simulation, and presents the results. Benchmarking against standard test cases confirmed both the accuracy of the simulations and the practical usability of the mobile platform. This application illustrates how interactive and portable DTs can democratize access to high-fidelity simulations.

The third application demonstrates the development of a large-scale, physics-driven DT for urban environments. By continuously incorporating data from meteorological stations, including wind speed, direction, and pollutant levels, the DT is able to simulate urban airflow and pollution distribution in real-time. A dynamically adjustable CFD model is able to process the incoming data and enables long-term simulations to identify pollution hotspots and high wind-speed areas across the urban area. The model includes an automated interface for geometry generation, which utilizes OpenStreetMap (OSM) data and ensures easy deployment across different urban areas. This capability has significant use for urban planning and environmental monitoring.

The fourth application focuses on a chamber filter press, a core technology in solid-liquid separation processes. Here, a neural network-based DT is developed using sensor data collected via an internet of things (IoT) setup. Parameters

such as pressure, flow rate, and user-defined configurations are continuously transmitted to a cloud database, which retrains the model with each new data entry. The NN-based DT is able to predict process time and efficiency of the used equipment (e.g., filter medium) due to it being based on historical usage and real-time inputs. These insights are made accessible through an app interface which integrates predictive analytics. The application, with its integrated ML-based module, demonstrates that the framework is able to support intelligent process prediction and optimization.

Each module and application was evaluated using targeted validation methods: the CFD simulations were benchmarked against analytical solutions and existing solver data to ensure numerical accuracy, while the neural network models were validated on test datasets to assess predictive reliability. Usability studies were conducted to assess the clarity, responsiveness, and educational value of the visualization tools. These assessments revealed not only strong alignment with physical reality but also a high degree of practical usability.

A key advantage of the framework developed in this work lies in its interoperability. Thanks to a consistently modular architecture and the use of standardized interfaces, the individual modules can be easily integrated into existing process control systems. At the same time, this architecture enables high scalability, allowing the framework to be tailored for specific processes and transferred to various industrial contexts. Noteworthy is the future possibility to combine data-driven and physics-based models to get a more adaptive digital representation of real-world processes.

The results of this work demonstrate that a modular, adaptive approach to DTs offers a powerful, flexible, and transferable solution for mechanical process engineering. The developed concepts, models, and applications make a substantial contribution to the advancement of DT technology in the context of industrial processes. The presented framework not only provides a solid foundation for future research but also enables direct application in industrial practice to enhance efficiency and sustainability. In the long term, DTs open up new perspectives for the intelligent, resource-efficient, and economically viable design of industrial systems.

Contents

1
Introduction

1.1 Motivation

The transition from Industry 4.0 to Industry 5.0 emphasizes intelligent, adaptive systems where digital twins (DTs) play a crucial role in predictive analytics and optimization. Despite increasing adoption, DT applications in process engineering remain underdeveloped. Unlike discrete manufacturing, process engineering involves continuous systems, complex fluid dynamics, and computationally intensive modeling, making real-time simulation and adaptive control particularly challenging. This research addresses these challenges by developing a modular DT framework tailored to process engineering, ensuring adaptability for filtration systems, fluid processing, and environmental monitoring. The framework is composed of three core components, visualization, modeling, and interface - designed to operate independently but in concert. Its architecture supports the seamless addition of new modules, enabling the continuous enhancement and integration of future models, data sources, and domain-specific applications. The proposed framework represents a step toward enabling more adaptable and efficient DT implementations in process engineering, particularly through its modular and extensible design.

1.2 State of the Art

The industrial sector is undergoing rapid transformation with advancements in digital technologies. Industry 4.0 introduced automation and data-driven decision-making through the internet of things (IoT), cloud computing, and big data to enhance operational efficiency [16]. Industry 5.0 extends this foundation by integrating artificial intelligence (AI) to foster a closer collaboration between humans and intelligent systems [17].

A key milestone in this transformation is the concept of the DT, first introduced by Grieves in 2002 [18]. This framework defines two primary components: the digital twin prototype (DTP), which facilitates the creation of a physical asset from its virtual counterpart, and the digital twin instance (DTI), which maintains a continuous link between the physical asset and its digital representation. Together, these elements operate within the digital twin environment (DTE), where multidomain physics applications enable DTs to simulate, monitor and predict system behavior.

Although this definition provides a foundational understanding, the specific requirements of DTs vary between disciplines. In process engineering, for example, real-time data exchange, high-fidelity simulations, and predictive modeling are crucial. Despite the growing number of DT definitions proposed in the literature, there is still no universally accepted framework, leading to inconsistencies in implementation [19–22]. This lack of standardization highlights the need for a structured and modular approach to ensure interoperability between different applications.

1.2.1 Definitions of Digital Twins

The concept of DTs has evolved across various domains, leading to multiple definitions and interpretations. This section explores key perspectives, highlighting commonalities and distinctive features. Aerospace applications define a DT as an ultra-high-fidelity simulation that mirrors the life of its physical twin, incorporating real-world data, health management, and fleet maintenance history [23]. A broader view sees the DT as a versatile tool that has emerged over the past decade, applicable to manufacturing healthcare and infrastructure [24]. Other definitions emphasize the real-time evolution of a DT, where continuous data exchange enables dynamic decision making [19]. Some works stress the necessity of a physical counterpart, arguing that without one, a DT is merely a model [20]. Additionally, DTs are described as digital representations of unique assets, integrating properties, conditions, and behaviors through data-driven models [25, 26]. Table 1.1 summarizes key DT definitions and their core characteristics.

Publication	Year	Application	Definition Keywords
[18]	2017	Lifecycle management	Representation of a physical system, data exchange, connection along the entire lifecycle
[23]	2012	Aerospace	Simulation with real-world data, data exchange, connection along the entire lifecycle
[24]	2022	General	Real – time monitoring, simulation, predictions
[19]	2021	Manufacturing	Real – time monitoring, simulation and modeling, predictions, evolution, domain-specific applications
[27]	2018	Manufacturing	Real – time monitoring, interaction and Convergence, self evolution
[20]	2020	General	Representation of a physical system, evolving, data exchange, predictions
[22]	2021	General	Virtual representation, data exchange, update
[28]	2020	Manufacturing	Physical entity/twin, virtual entity/twin, physical environment, virtual environment, state, realisation, metrology, twinning, twinning rate, physical-to-Virtual connection/twinning, virtual-to-physical connection/twinning, physical processes
[25]	2017	Manufacturing	Master model, digital shadow
[26]	2019	Business model	Data exchange, evolving

Table 1.1: *Definitions of a digital twin from literature*

The following aspects frequently appear in literature:
Data Exchange: Various authors [18–20, 22, 23, 26], consistently highlight the importance of data exchange. This involves the dynamic flow of information between the digital and physical counterparts.
Connection along the entire lifecycle: [18, 23] stresses the importance of maintaining a continuous link between the digital and physical entities throughout the lifecycle. This ensures a seamless integration that extends beyond specific phases.
Real-time monitoring: [19, 24, 27], emphasize the concept of real-time monitoring. This involves continuous observation and updating of the digital model based on real-world data received from the physical counterpart.
Simulation: [19, 20, 23, 24] focus on simulation. DTs enable simulations that go beyond static models, allowing actionable insights into physical forces and

behaviors over time.
Predictions: [19, 20, 24] highlight the capability of DTs for making predictions. This involves leveraging real-time data to foresee future behaviors and outcomes.
Virtual representation: [22] discuss the concept of virtual representation. DTs serve as dynamic digital models that intelligently represent physical objects or processes in a virtual space.
Evolving: [20, 26, 27], contribute to the understanding of DTs as evolving entities. The digital representation intelligently evolves in response to real-time data, maintaining synchronization with the physical counterpart.

1.2.2 Digital Twin in Process Engineering

Process engineering, particularly in the mechanical domain, plays a crucial role in industrial advancements by optimizing the transformation of raw materials into refined products. This field relies on precise control of fluid dynamics, particle behavior, and thermodynamic processes to enhance efficiency, safety, and product quality. Computational fluid dynamics (CFD) is a key enabler, providing simulation-based insights for performance optimization. Despite the widespread adoption of DTs in manufacturing and healthcare (as summarized in Table 1.1 and discussed in [21, 28]), their application in mechanical process engineering remains limited. This gap presents an opportunity to leverage CFD-enhanced DTs for real-time process optimization, predictive maintenance, and improved decision making. Using both data-driven models, such as neural network (NN)-based approaches and physics-based simulations, DTs in process engineering can significantly improve the efficiency of the industrial system, providing a scalable and adaptive framework for complex fluid- and particle-based processes.

1.3 Conceptualizing Digital Twins for Process Engineering

Given the challenges and varying definitions discussed, a structured but flexible approach to defining DTs in process engineering is necessary. Table 1.1 compiles various definitions of DTs from the literature, extracting key elements that distinguish a DT from a conventional model. Although these defining elements provide essential characteristics, their necessity varies depending on the specific application. A general consensus in the literature suggests that for a system to qualify as a DT, it must encompass data exchange, simulation, and prediction—all of which are highly relevant in mechanical process engineering. Rather than imposing these elements as rigid requirements, we propose a modular approach, allowing DTs to be constructed with flexibility while maintaining essential components. Certain core modules must be present for a system to be classified as a DT, while others remain adaptable to specific use cases. A fundamental aspect of a DT is data exchange between the physical and virtual domains. This interaction is part of a broader modular architecture, which includes an interface (for data exchange), a computational model, and a visualization component. Figure 1.1 illustrates this modular composition, showing how these elements interact to form a coherent DT system tailored to process engineering. Wright *et al.* [20] argue that a system without data interaction is

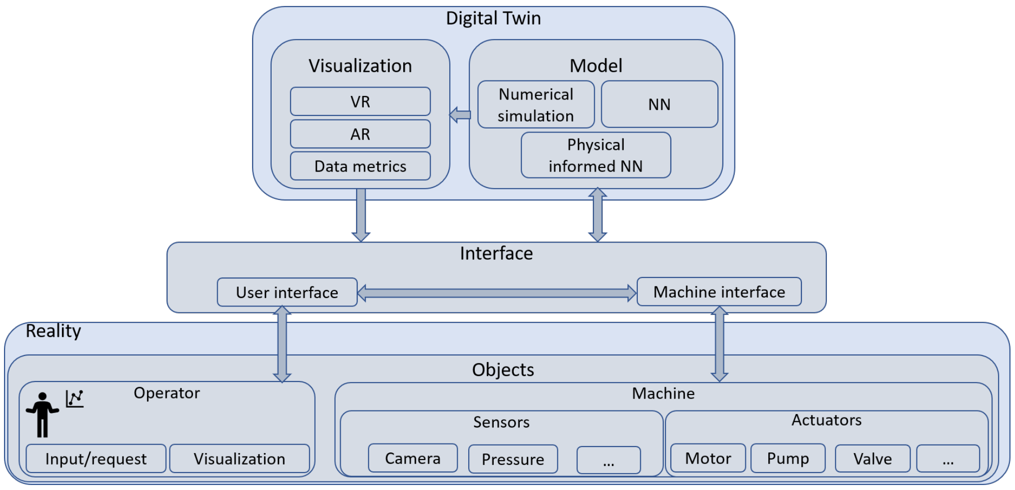


Figure 1.1: *Modular architecture of a DT in process engineering. The DT consists of three core modules: the interface, which acts as a cornerstone connecting the DT with the physical world and enabling bidirectional data exchange; the model module, which may include numerical simulations, physics-informed neural networkphysics-informed neural networks (PINNs), or standard data-driven NN; and the visualization module, which provides insight through metrics, virtual reality (VR), or augmented reality (AR). The user interface facilitates interaction with the system, while real-world data is acquired via sensors and actuators connected to machines and operators.*

merely a model rather than a DT. The author aligns with this view, identifying data exchange as a cornerstone module: an interface that ensures connectivity between DT and reality. However, this exchange does not necessarily need to be bidirectional in all cases. Another essential component is the user interface, which facilitates data input and output. This becomes particularly critical when numerical simulations serve as the model, requiring user interaction to modify boundary conditions, control parameters, or visualize results. The visualization module provides meaningful representations of system data, ranging from simple metrics to immersive formats such as augmented reality (AR)—which blends virtual data with real-world context—or virtual reality (VR), offering a fully virtual experience of the system. Finally, the model module defines the computational logic of the DT. It can be implemented using data-driven NN, physics-informed physics-informed neural networkphysics-informed neural networks (PINNs), or high-fidelity numerical simulations, depending on the application's fidelity requirements and computational constraints. This modular framework provides a structured yet flexible foundation for DTs in process engineering, ensuring adaptability while preserving their essential characteristics.

1.4 Aims and Challenges

Current research indicates a significant gap in the adoption of DTs within process engineering, particularly in the areas of simulation-based DT and efficient data exchange. As discussed in Section 1.3, a modular definition of DTs for process engineering has been proposed based on the existing literature. However, challenges remain in terms of interoperability, scalability, real-time data integration, and the ability to generalize DT frameworks across different process engineering applications. Building upon this foundation, the primary objective of this thesis is:

The development of a digital twin framework for process engineering that can be transferred to a wide range of applications, particularly in industrial process optimization and smart infrastructure, with a focus on simulation-driven and neural network-based DTs, ensuring adaptability, scalability and seamless data integration.

To achieve this, the modular components outlined in Figure 1.1 are developed, forming the foundation of a flexible and scalable DT framework. These modules are designed to ensure seamless integration, adaptability, and efficient data exchange within process engineering applications. The development of these modules is guided by the following subgoals:

- Subgoal 1: Development of a **visualization module** that enables effective representation of DT data through performance metrics, AR, or VR, thereby enhancing interpretation and analysis in process engineering.

- Subgoal 2: Implementation of a DT that integrates the **visualization module** and a **CFD-based simulation module**, enabling real-time fluid dynamics analysis with improved data representation and user interaction.
- Subgoal 3: Creation of an updatable **CFD-based simulation module** capable of adapting to real-time changes, connected through the same **interface module** to ensure synchronization with current system states and continuous data feedback.
- Subgoal 4: Development of a **NN-based module** for real-time monitoring, predictive analytics, and system optimization in process engineering, supported by an **interface module** that bridges the DT with real-world data through the integration of IoT sensor inputs.

By addressing these subgoals, this research aims to develop a transferable simulation- and neural network-based DT framework applicable to similar use cases in process engineering.

1.5 General Approach

To address the challenges and objectives outlined in Section 1.4, this research adopts the modular definition of DT from Section 1.3 to develop a **flexible and scalable DT framework** for process engineering. The framework is developed incrementally through three distinct **DT applications**:

- A **Virtual CFD Lab for Mobile Devices**, enabling real-time, interactive fluid simulations.
- A **DT for a Chamber Filter Press**, designed to predict filter performance and optimize operations using machine learning.
- A **DT for Urban Environments**, integrating CFD-based models with meteorological data to dynamically track pollution and wind distribution over extended periods.

Each DT is constructed by developing and integrating specific modules, with a focus on preprocessing, visualization, and modeling. The contributions of this work are primarily expressed through the development process of the respective modules.

Data Preprocessing To ensure adaptability, interactivity, and usability, classes are implemented in the software OPENLB that enable flexible parameter adjustments, validation, and seamless integration of external data sources. The focus here is to establish the foundation for user interfaces that allow users to adjust numerical simulation-based DTs. This step is particularly significant for the application DT for Urban Environments. Information is stored in XML or CSV files, which are accessed by the classes during runtime and can be edited externally without requiring recompilation.

Contribution I: Data Preprocessing

- Adjustable dynamics from XML or third-party sources.
- Parameter validation to enhance reliability.
- Large-scale, interactive geometries processed from OpenStreetMap (OSM).

Visualization Module A modular, extensible code framework based on UNITY is developed for the visualization module. This framework allows for the generation of application-specific user interfaces for visualizing data. It integrates NN and lattice Boltzmann method (LBM) shared libraries to interact with and display simulation data. The framework supports overlaying simulations and corresponding geometries in AR on physical objects and can be ported to VR for a fully virtual environment. Leveraging UNITY's cross-compilation capability, the application can be built for various platforms, including mobile devices and Windows executables. A visualization tool for precompiled CFD simulations was also developed, with usability validated through user testing with students.

Contribution II: Visualization Framework (Visualization)

- Geometry overlay for better spatial representation.
- CFD data integration for enhanced insights.
- Usability validation to ensure accessibility and functionality.

Virtual CFD Lab on Mobile Devices The visualization module is combined with an LBM model that can be interacted with through user input, marking the first DT application. This allows users to scan hand-drawn simulation domains and simulate them in real-time on mobile devices. The results were compared with benchmark tests to validate usability.

Contribution III: Virtual CFD Lab on Mobile Devices

- LBM-based just-in-time simulation on mobile devices.
- 2D simulation from scanned hand-drawn domains.
- Validation of simulation accuracy.

Development of a CFD-Based DT for Urban Environments The third application involves the development of a CFD-based DT for urban environments. The DT is continuously updated with data from meteorological stations, such as pollution concentrations, wind speed, and direction. For this purpose, a dynamically adjustable CFD-based model module was developed.

Contribution IV: Development of a CFD-Based DT (Input, Physical Driven)

- Large-scale DT for urban environments.
- Interface for urban geometry adjustments.
- Continuous updates for pollution tracking.

DT for a Chamber Filter Press In the foruth application, a neural network-based model is trained using sensor data within an IoT framework. Data from the machine are collected via sensors and can be accessed by users through an app. The data is automatically stored in a cloud database, which continuously updates the neural network model with new training data. The DT predicts pressure, flow rate and processing times based on the number of cycles that the filter medium has been used, along with other process parameters. This integrates the NN-based model with the visualization module, enabling users to access performance predictions through a connected database on mobile devices or computers.

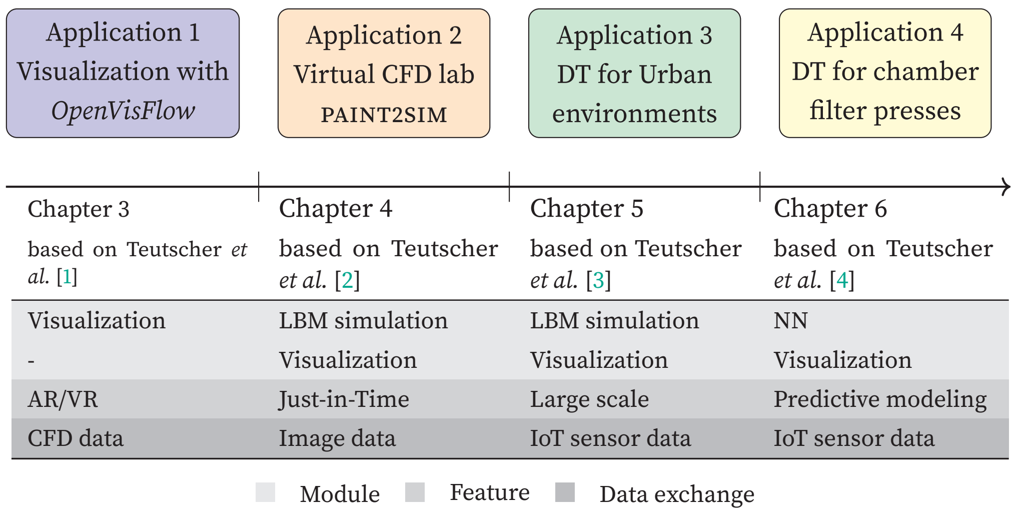
Contribution V: Development of the NN-based Model Module

- machine learning (ML)-based DT framework for a chamber filter press.
- Predictive modeling of filter medium efficiency.
- Continuous data exchange and model updates.

By developing these modules within the context of the three applications, the research ensures the transferability of the framework to other applications that require either NN-based or CFD-based models. The NN approach is applicable to various scenarios where measurable parameters correlate with a performance evaluation output variable, such as in industrial processes or predictive maintenance. The CFD-based module is suited for applications that require both quantitative and qualitative data analysis, such as in hospitals to track the distribution of bacteria influenced by ventilation systems and the number of people present in different areas.

1.6 Thesis Structure

The thesis is structured according to the four previously introduced applications, each of which integrates modules for visualization, modeling, and data exchange. Figure 1.2 illustrates the chronological development of these applications and their associated modules.



Application 1 Visualization with *OpenVisFlow*	Application 2 Virtual CFD lab PAINT2SIM	Application 3 DT for Urban environments	Application 4 DT for chamber filter presses
Chapter 3 based on Teutscher *et al.* [1]	Chapter 4 based on Teutscher *et al.* [2]	Chapter 5 based on Teutscher *et al.* [3]	Chapter 6 based on Teutscher *et al.* [4]
Visualization	LBM simulation	LBM simulation	NN
-	Visualization	Visualization	Visualization
AR/VR	Just-in-Time	Large scale	Predictive modeling
CFD data	Image data	IoT sensor data	IoT sensor data

▢ Module ▨ Feature ■ Data exchange

Figure 1.2: *Structure of thesis with four applications that were developed, integrating the modules for the DT. Each application represents different aspects, including visualization tools, CFD-based modeling, large-scale DT simulations, and data-driven modeling.*

Chapter 2 discusses the preprocessing steps necessary for preparing data for parsing, along with the core concepts and implementation details. This chapter contains **Contribution I: Data Preprocessing**, focusing on dynamic simulation setup adjustments, parameter validation, and the implementation of a parsing class for OpenStreetMap (OSM).

Chapter 3 presents **Contribution II: Visualization Framework**, based on the work of Teutscher *et al.* [1]. This chapter discusses the development of a modular visualization library, including interfaces for CFD data integration and functionality to overlay simulated geometries onto physical objects. A visualization application was developed to display pre-run simulations stored in a cloud database, which was tested by students and received positive feedback.

Chapter 4, based on Teutscher *et al.* [2], introduces **Contribution III: Virtual CFD Lab on Mobile Devices**, demonstrating the integration of the visualization framework with an LBM model. The application allows users to scan hand-drawn simulation domains for real-time simulations, supports numerical parameter adjustments (such as the Reynolds number), and functions as a DT for 2D fluid flow simulations.

Chapter 5 introduces **Contribution IV: Development of a CFD-Based DT**, describing a large-scale DT for urban environments based on Teutscher *et al.* [3]. This chapter features a model utilizing the homogenized lattice Boltzmann method (HLBM) to simulate the distribution of pollutants, such as particulate matter, in urban areas. The input data is based on OSM data and meteorological information from wind and pollution measurement stations. By continuously updating the model over time, pollution hotspots and high wind speed areas can be identified. This chapter also includes a validation study using experimental wind tunnel data.

Chapter 6 introduces **Contribution V: Development of a Data-Driven DT**, based on the work of Teutscher *et al.* [4]. This chapter presents an ML-based DT framework for a chamber filter press, which connects machine sensors to a cloud-based database. Additionally, it covers the implementation and training of a NN model for predictive modeling of filter medium efficiency, along with the validation process.

Finally, Chapter 7 provides a comprehensive summary of the key findings from previous chapters and outlines potential directions for future research.

2
Preprocessing for Digital Twin Integration

Before starting a numerical simulation, it is essential to meticulously prepare all necessary data, such as parameters, initial conditions, and boundary conditions. This preparation phase is critical because the data must meet the stringent requirements set by the numerical methods used. In the context of integrating user interfaces and external tools, preprocessing plays a pivotal role in ensuring accurate and efficient simulations.

Consider the example of a digital twin for a city, where the model simulates various urban phenomena, such as wind flow or pollution distribution. In such complex scenarios, preprocessing techniques are vital for adapting the simulation to different conditions and requirements. These techniques ensure that the data are compatible with the simulation framework and allow dynamic adjustments and validation of parameters, such as adjusting the characteristics of the wind or the current pollution state based on real-time data or simulating different environmental conditions.

Thus, preprocessing is not an isolated task, but a foundational component of the entire numerical simulation workflow. By incorporating robust communication strategies, parameter validation, and dynamic configuration, the user experience can be enhanced and ensure that simulations, such as those of the digital twin of a city, are both realistic and reliable.

2.1 Concept

Preprocessing is the foundation of an interactive simulation environment, ensuring that the parameters, models, and configurations are correctly set before the simulation begins. In this system, preprocessing can either be driven by a User Interface or automatically by a third-party software, depending on the application. The UML sequence diagram shown in Figure 2.1 illustrates both scenarios, showing how different components, such as the user interface, third-party software, for example, meterological stations, the OpenLB back-end interface and the simulation environment, interact to facilitate real-time and manual updates.

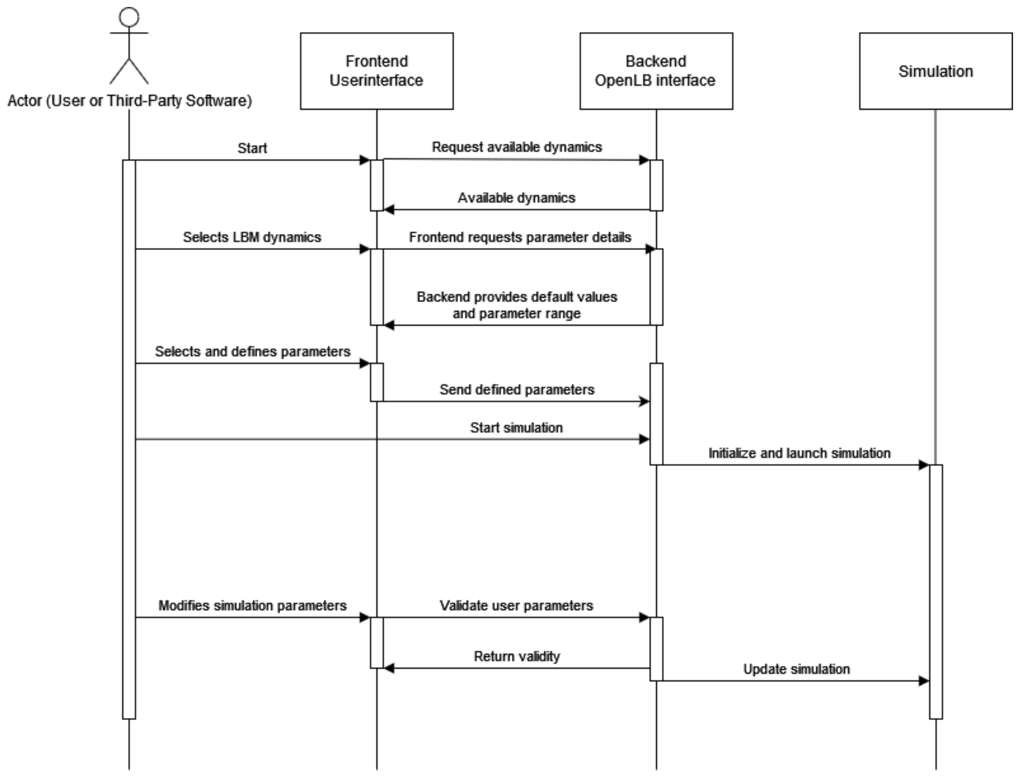

Actor (User or Third-Party Software)
Frontend Userinterface
Backend OpenLB interface
Simulation
Start
Request available dynamics
Available dynamics
Selects LBM dynamics
Frontend requests parameter details
Backend provides default values and parameter range
Selects and defines parameters
Send defined parameters
Start simulation
Initialize and launch simulation
Modifies simulation parameters
Validate user parameters
Return validity
Update simulation

Figure 2.1: *UML sequence diagram illustrating the preprocessing workflow in the simulation environment. The process can be initiated either by a user or third-party software, interacting with the frontend interface to define simulation parameters. These parameters are validated by the backend* OpenLB *interface before initializing or updating the simulation environment.*

2.1.1 User-Driven Preprocessing

When the preprocessing is manually driven through the user interface, the process begins by requesting and selecting the dynamics governing the simulation. The user initiates the process by selecting appropriate dynamics, such as fluid or wind flow models, in the case of simulating urban airflow or pollution dispersion.

After selecting the dynamics, the system moves to the parameter definition and validation phase. Here, the user interface prompts the backend for parameter suggestions, providing default values and advised ranges. Users can adjust parameters, the backend then validates these inputs to ensure that they are within acceptable ranges, preventing errors or instabilities during the simulation.

Once the parameters are validated, the system prepares the simulation environment by configuring the computational domain, the size of the grid, the boundary conditions, and the meshing. After completion of the setup, the simulation starts.

2.1.2 Automated Preprocessing via Third-Party Software

In contrast, preprocessing can also be automated when third-party software such as a meterological station is used to update the simulation with real-time data. Instead of manual user input, the system continuously receives updated environmental data, such as real-time wind speeds, temperature readings, or pollution levels, directly from the measuring station.

In this case, the measuring station serves the same role as the user, providing real-time data that the backend integrates into the simulation. The backend validates this incoming data in the same way that it validates user input, ensuring that the real-time measurements are physically meaningful and suitable for the simulation.

Once the data are validated, the Backend OpenLB interface updates the simulation with the new parameters. This process enables the simulation to dynamically adjust to real-world conditions, maintaining accuracy even as environmental factors change. For example, a sudden change in wind speed detected by the measuring station would prompt the backend to update the wind dynamics of the simulation accordingly.

2.1.3 Shared Processing Flow

Both the manual user-driven process and the automated real-time process share several key stages. Regardless of whether the input comes from a user or a measuring station, the system follows the following steps:

- Dynamics selection (either manually by the user or based on predefined models for the real-time data).
- Parameter validation to ensure that the data is accurate and within valid ranges.
- Simulation setup and execution after the parameters are validated.
- Real-time updates, which allow for continuous simulation adjustments based on changing conditions (either by user interaction or by input of real-time data).

This dual approach provides flexibility. For scenarios like urban planning, where simulations must reflect real-world conditions (e.g. air quality monitoring), using a measuring station for real-time data ensures that the simulation stays relevant and up-to-date. On the other hand, for controlled simulations or what-if analyses, a user interface provides manual control, allowing users to experiment with different parameters and observe outcomes.

2.2 Dynamic Code Adaptation for User Interface Integration

To facilitate the editing of simulation data from external tools, such as user interfaces or other code bases, a robust communication strategy is essential. This strategy must incorporate mechanisms for parameter validation to enhance the user experience by ensuring that inputs are accurate and meaningful. Key considerations include:

- **Communication strategy**: The system must support bidirectional communication between OpenLB and external tools, enabling seamless data exchange and real-time parameter updates.
- **Parameter validation**: Integrating parameter validation ensures the accuracy of the simulation setup, allowing the user interface to provide immediate feedback on the validity of user input.

- **Dynamic configuration**:XML-based configuration enables dynamic adaptation, allowing users to customize simulation parameters without directly modifying the underlying code. This flexibility is crucial for complex simulations, where different scenarios may require different dynamic setups.

2.2.1 Communication Strategy

An XML file serves as an intermediary between OpenLB, the user interface, and any third-party software, as illustrated in Figure 2.2. This XML file can be read and modified by OpenLB, the user interface, and other software applications, enabling seamless data exchange. This approach not only facilitates communication between OpenLB and the developed user interface but also allows easy integration with other external tools, extending the usability of the system beyond the initially developed interface.

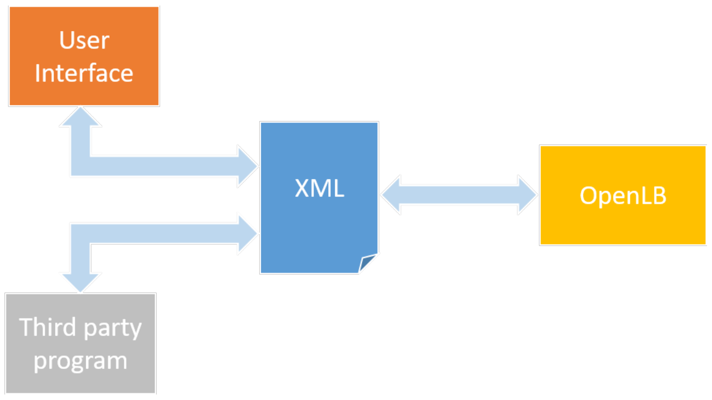


Figure 2.2: *Communication between OpenLB, user interface and other software with XML as an intermediary,*

2.2.2 Dynamics Configuration in OpenLB

In OpenLB, dynamics are defined using Tuples that encapsulate various components, such as momenta, equilibria, and the collision operator. For instance, a porous dynamic system with second-order equilibria and a Smagorinsky collision operator might be defined as follows:

```
template <typename T, typename DESCRIPTOR>
using BulkDynamics =
    dynamics::Tuple<T, DESCRIPTOR, Porous<BulkTuple>, equilibria::
        SecondOrder,
                    collision::SmagorinskyEffectiveOmega<collision
                        ::BGK>>;
```

Listing 2.1: *Point-in-Polygon Check*

The structure of this Tuple can be effectively represented in XML format as shown in Listing 2.2, allowing the inclusion of additional information, such as necessary parameters. This XML representation simplifies the configuration and modification of the simulation setup, ensuring that all required elements are accurately defined and easily accessible to both users and external software.

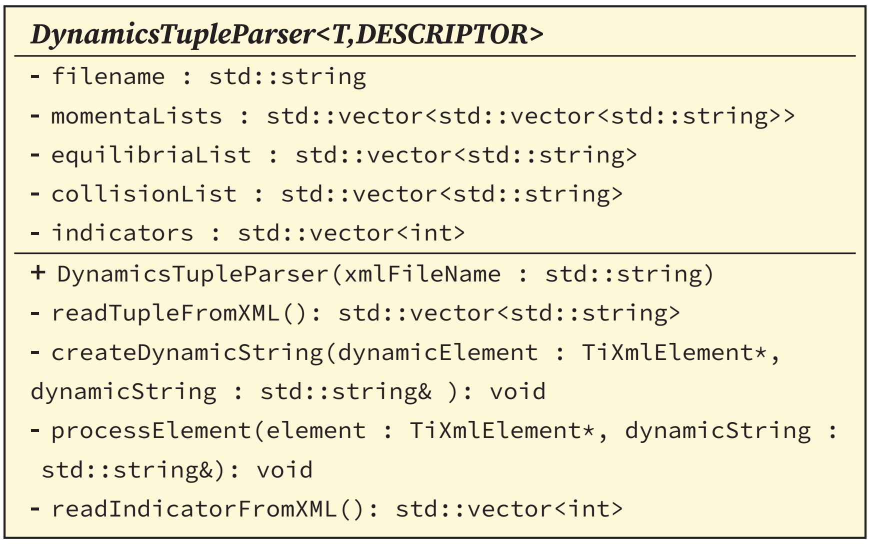
```
<Methods>
<Map>
  <Indicator>1</Indicator>
  <Dynamic>
    <Momenta>
        <BulkDensity/>
        <PorousMomentum>
            <BulkMomentum/>
        </PorousMomentum>
        <BulkStress/>
        <DefineToNEq/>
    </Momenta>
    <Equilibria>
        <SecondOrder/>
    </Equilibria>
    <Collision>
        <SmagorinskyEffectiveOmega>
            <BGK/>
        </SmagorinskyEffectiveOmega>
    </Collision>
  </Dynamic>
</Map>
</Methods>
```

Listing 2.2: *Dynamic tuple in XML structure.*

2.2.3 Parsing XML to Dynamics

To construct the dynamic from an xml, a class `DynamicsTupleParser<T, DESCRIPTOR>` was developed, as shown in Figure 2.3.

Since direct conversion from a string to a type is not possible in C++17, a predefined list of available dynamics in OpenLB is necessary. The `DynamicsTupleParser` compares the string representation of the dynamic with this list to find a match, leveraging the fact that every dynamic in OPENLB has a unique name and can be represented in its tuple form.



DynamicsTupleParser<T,DESCRIPTOR>

```
- filename : std::string
- momentaLists : std::vector<std::vector<std::string>>
- equilibriaList : std::vector<std::string>
- collisionList : std::vector<std::string>
- indicators : std::vector<int>
+ DynamicsTupleParser(xmlFileName : std::string)
- readTupleFromXML(): std::vector<std::string>
- createDynamicString(dynamicElement : TiXmlElement*,
dynamicString : std::string& ): void
- processElement(element : TiXmlElement*, dynamicString :
 std::string&): void
- readIndicatorFromXML(): std::vector<int>
```

Figure 2.3: *The DynamicsTupleParser class used to extract the dynamic strings from a XML file.*

```
void prepareLattice(SuperLattice<T,DESCRIPTOR>& sLattice,
                    SuperGeometry<T,3>& sGeometry, const
    UnitConverter<T,DESCRIPTOR>& converter, IndicatorF3D<T>&
    volume)
{
    OstreamManager clout(std::cout, "prepareLattice");
    const T omega = converter.getLatticeRelaxationFrequency();
    sLattice.defineDynamics("city3d.xml",sGeometry);
                                   .
                                   .
```

Listing 2.3: *Point-in-Polygon Check*

This approach ensures a flexible and robust configuration of dynamics in OPENLB, making the simulation setup both user-friendly and compatible with existing software structures.

2.3 Automated Large-Scale Geometry Preprocessing

When it comes to parsing large-scale geometries, such as urban environments used in this thesis, there is a need to quickly and efficiently generate them. Each object within these environments can have specific characteristics that must be considered during the simulation setup. These can include, but are not limited to, individual trees or vegetation areas, streets, and buildings. Therefore, a parsing method is required that can distinguish between different objects and account for their unique properties.

When parsing geometries in OpenLB, the common formats are typically STL files or inline representations composed of simple shapes (e.g., rectangles, circles). Automating this process presents several challenges, including the following:

- **Watertight geometries requirement:** In OpenLB, all geometries must be watertight, meaning that surfaces must be fully enclosed with no gaps or non-manifold edges. Ensuring watertightness is often a challenge when working with complex geometries or when combining multiple parts that must seamlessly fit together.

- **STL file handling:** Editing or modifying STL files can be tedious and prone to errors, making it difficult to efficiently adapt geometries for large-scale simulations.

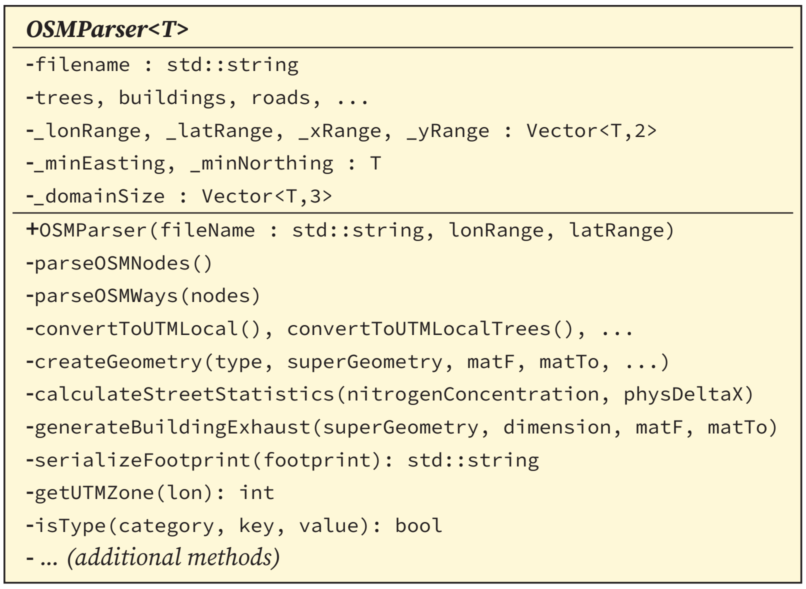
- **Limited detail in simple shapes:** Basic shapes such as rectangles or circles may simplify geometry creation, but do not capture the intricate details necessary for accurate simulations. This limitation is particularly relevant in urban modeling, where complex structures and irregular shapes are common.

- **Lack of automation:** Every object with a unique material number must be loaded separately or manually adjusted after loading using OpenLB rename functions, increasing the complexity of preprocessing.

To address these challenges, OSM is used. Maps extracted from the OSM API provide data on various objects in the form of XML files. These objects can be tagged with names and other relevant parameters, while the necessary information about their geometric shape is stored in nodes containing latitude and longitude coordinates. This structured data can then be used to generate the corresponding geometry, enabling a more efficient and automated preprocessing workflow.

2.3.1 Parsing of OpenstreetMap Data

In order to read and parse OSM data, a class `OSMParser` shown in Figure 2.4 is introduced. It allows reading and manipulating an OSM map, where the whole map or a specific domain size is defined by latitude and longitude with the constructor parameters `lonRange` and `latrange`. After calling the constructor, nodes containing the coordinates and ways containg object information are extracted.



```
OSMParser<T>
-filename : std::string
-trees, buildings, roads, ...
-_lonRange, _latRange, _xRange, _yRange : Vector<T,2>
-_minEasting, _minNorthing : T
-_domainSize : Vector<T,3>
+OSMParser(fileName : std::string, lonRange, latRange)
-parseOSMNodes()
-parseOSMWays(nodes)
-convertToUTMLocal(), convertToUTMLocalTrees(), ...
-createGeometry(type, superGeometry, matF, matTo, ...)
-calculateStreetStatistics(nitrogenConcentration, physDeltaX)
-generateBuildingExhaust(superGeometry, dimension, matF, matTo)
-serializeFootprint(footprint): std::string
-getUTMZone(lon): int
-isType(category, key, value): bool
- ... (additional methods)
```

Figure 2.4: *The OSMParser class extracts and converts OpenStreetMap data into simulation-ready geometry.*

In `parsOSMWays`, information about the ways with their object type is stored in predefined structures. After parsing, the UTM coordinates are transformed into Cartesian coordinates, and the data origin is moved to zero using `convertUTMLocal()`.

2.3.2 Geometry Creation

The extracted objects are created using the function `createGeometry(...)`, which places them within the simulation domain and assigns a material number based on their type. This function utilizes the `Indicator3D` class from OPENLB, which has been extended with a new class, `IndicatorPolygon3D`. This extension enables the computation of the plane equation from three initial points and determines whether a given point lies within the projected polygon using the ray-casting algorithm. The corresponding UML flow chart in Figure 2.5 visually describes this algorithm, depicting the sequence of operations from projection to containment checking and classification. For each point in the list of nodes

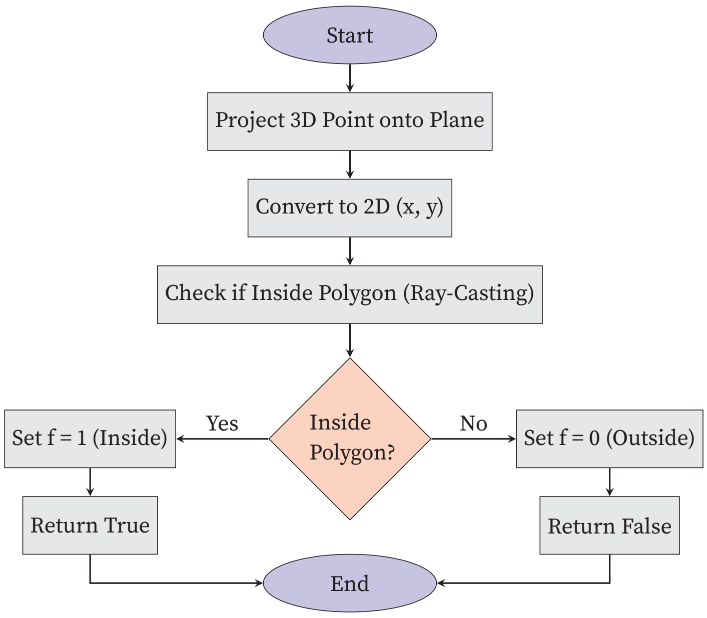


Figure 2.5: *Workflow of the `IndicatorPolygon3D` class for 3D point-in-polygon evaluation. A 3D point is projected onto a plane and converted to 2D coordinates. A ray-casting algorithm is used to determine whether the point lies within the polygon. The method returns true if the point is inside and false otherwise.*

that define a building, the algorithm follows a structured process:

- Plane projection: The normal vector of the plane is computed by taking the cross-product of two edge vectors from the polygon. Using this normal, the 3D point is projected onto the polygon's plane.
- 2D transformation: The projected point is converted into a 2D coordinate system by extracting only its x- and y-components. This simplification allows efficient containment checks.
- Containment check (Ray-Casting Method): The algorithm iterates through the edges of the polygon and casts a horizontal ray from the point. It counts the number of intersections between the ray and the edges of the polygon:

  - Odd number of intersections → The point is inside the polygon.
  - Even number of intersections → The point is outside the polygon.
- Classification and return: Based on the containment result, the function assigns f=1 if the point is inside the polygon and f=0 otherwise.

This process is repeated for each point in the list of nodes to ensure a correct geometric representation of the building structures within the simulation domain.

The use of the class is illustrated in Listing 2.4. The geometry can be built by instantiating the constructor `OSMParser` with the filename of the OSM map. This allows for the extraction of the general bounding box that encompasses the domain or manual definition of it. The `createGeometry` method is then used to build the geometry with the corresponding type (e.g., building, tree).

```
    OSMParser<T> osmParser("map.osm");
    olb::Vector<T,3> origin(0,0,0);
    auto extent = osmParser.getDomainSize(30.);
    olb::IndicatorCuboid3D<T> bounding(extent, origin);
    olb::CuboidDecomposition3D<T> cGeometry(bounding,1,singleton::
        mpi().getSize());
    olb::BlockLoadBalancer<T> loadBalancer(cGeometry);
    olb::SuperGeometry<T,3> sGeometry(cGeometry, loadBalancer);
    osmParser.createGeometry(sGeometry,0,1,type.BUILDING);
```

Listing 2.4: *Use of class OSMParser*

3 Visualization Framework for Interactive Digital Twins

This chapter is based on the published article

Teutscher, D. et al. "Interactive Scientific Visualization of Fluid Flow Simulation Data Using AR Technology-Open-Source Library OpenVisFlow." In: Multimodal Technologies and interaction *6.9 (2022), p. 81.* DOI: *10.3390/mti6090081.*

The abstract has been omitted, and formatting has been adjusted to fit the thesis style. The main content remains unchanged.

3.1 Introduction

Computational fluid dynamics (CFD) is a very useful tool to understand and optimize complex processes and machines. It is widely used in industries such as automotive [29], aerospace [30], and chemical engineering. Thus, the simulation results serve as a basis for decision making. Since CFD already is part of the everyday life of an engineer, this tool is being taught to students as standard during studies. However, building and understanding resulting simulation data can be very time-consuming, and additional tools such as Paraview or Gnuplot are needed to bring the results into a suitable and presentable form. Depending on the discipline, this can be a problem for teaching, as students can lose their concentration over time [31]. This in turn leads to a worse learning experience. It can be helpful if the simulation results are visualized in advance or just in time, so that students can better interact with them.

In the past few years, there has been a strong growth in the demand for augmented reality (AR) and virtual reality (VR) applications as Mojtaba Noghabaei *et al.* [32] show in their work, taking into account extensive surveys. AR in particular is increasingly used in areas such as medicine, mechanical engineering, and process engineering. Even for the medical field, especially for surgery, it can be a powerful tool to provide additional information to the surgeon as shown in the work by Raabid Hussain *et al.* [33]. In the work by Anna V. Iatsyshyn *et al.* [34], they analyzed in detail the usability in the modern era. They show that AR has a significant value for teaching. Furthermore, they explain that there are currently not many experts in this field and that future experts should be trained for the new technological era. While there are already specific applications that combine CFD with AR, there is not yet one that provides an interface to easily visualize any simulation.

Several AR applications are already available that are geared toward the teaching field, and many areas benefit from these. The application cleARmath [5] can visualize vector geometries in AR and thus provide a playful learning experience as well as a better understanding of the position of planes, vectors, etc. TeachAR [35] is an application that allows non-English speakers to playfully learn colors, 3D shapes, and spatial relations in English. The BARETA [36] application is designed to provide medical students with a better understanding of where the ventricular system is located in the brain. Evaluations were carried out for each of the applications mentioned. The majority of the persons tested were able to derive a benefit from the respective application. Based on this information, an application that visualizes complex processes such as fluid flow in AR would be a viable tool to better understand them. There are already AR applications with respect to computational fluid dynamics (CFD). In their work, Jia-Rui Lin *et al.* [37] visualized the thermal environment that occurs indoors in AR. They developed their own file format (cfd14a) which can be accessed via a server on the smartphone. There are several papers dealing with the visualization of CFD simulations [38–42].

Serkan Solmaz and Tom Van Gerven [43] describe in their work that CFD simulations are difficult to understand for non-experts, which limits their use. They present a method using Unity with the Vuforia extension to retrieve CFD simulations from a server and display them in AR. In our work, we take a different approach. We do not use the closed source extension Vuforia, but the open-source variant Google AR Core. With this, we are able to develop an open-source modular library. The goal is to create an interface between powerful tools like CFD and AR visualization that can be used by developers to create applications that can visualize complex processes.Furthermore, the main goal of the library is to fully integrate visualization into the simulation itself to allow users to visualize the mentioned processes on a cluster and stream the results in real time. Such a feature would enable interesting training and testing applications in a variety of domains. In this paper, we present the framework for such a library, which allows us to embed computed transport fields into 3D objects. In addition, the library allows the developer to choose from different visualization methods. One of these methods is the markerless geometry overlay that we developed and which is also presented in this paper. Since the ARCore extension is open source, everything can be customized to the user's personal needs. Extensions like Vuforia do not allow this because they are closed-source. This enables research into new visualization approaches, as shown in this paper with the novel markerless geometry overlay.

The contributions in our paper can be summarized as follows: Development of an application OpenLBar with updateable simulations; A way of geometry overlay without using a neural network; Introducing an open source library for AR application development. The resulting framework allows an easy extension of visualization methods and offers the possibility to fully integrate simulations into the code.

In the remainder of the paper, we first present the basic concept of our project. Then, the functionalities of the application and the open-source library are presented in the realization part using the example application OpenLBar. Finally, the advantages of visualizing simulations in AR are quantitatively and qualitatively reviewed.

3.2 Concept

The main aim is a scientific visualization concept that allows CFD data to be experienced with AR/VR and provides an interface between simulation- and visualization software. Creating a direct link between the two, allows for the direct visualiation and subsequent streaming from the cluster aswell as an option to integrate changes made in VR back to the simulation on the cluster. In short it would allow for the creation of a truly interactive simulation environment.

The visualization should be in different display modes like screen, AR, and

VR. As shown in Figure 3.1, it should also serve as a direct interface between simulation software and the visualization tool. With it, the different applications like OPENLBAR, which is presented in this paper, can be created. OPENLBAR is the first result of this library.

In the CFD area, the Navier-Stokes equation (NSE) is of particular importance. For simplicity, we consider the simplified NSE for non-compressible fluids [44]. This is defined as

$$\frac{D\boldsymbol{u}}{Dt} = -\frac{\nabla p}{\rho} + \nu \cdot \triangle \boldsymbol{u} + \boldsymbol{F}, \tag{3.1}$$

where $\frac{D}{Dt} = \frac{\delta}{\delta_t} + \boldsymbol{u} \cdot \nabla$ is the material derivative which indicates the temporal and spatial changes.; ρ is the density; $\triangle$ is the Laplace operator; ν is the kinematic viscosity; ∇p is the pressure gradient; $\boldsymbol{F}$ is the force vector and $\boldsymbol{u}$ is the velocity vector. By solving this equation with the finite difference or lattice Boltzmann method (LBM), one then obtains the information about the pressures p and the velocities $\boldsymbol{u}$. It should be noted that the LBM equations are not the same as the NSE, since it describes the mesoscopic level, however the NSE can be recovered from LBM with the Chapman-Enskog analysis [45]. Although, this will not be discussed further here, since it is not the focus of this work.

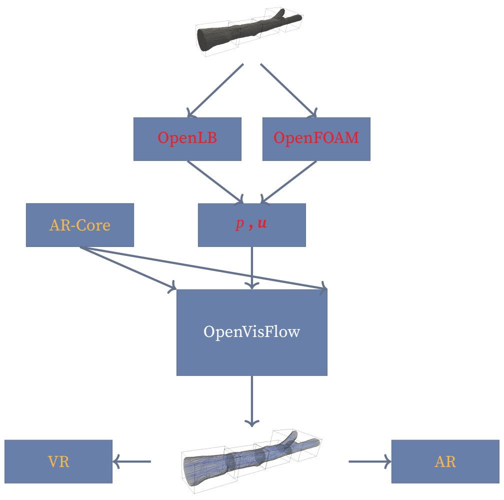


Figure 3.1: *Concept of the OpenVisFlow library, bridging the gap between simulation software (red) and helping libraries (yellow) to directly write programs and apps that scientifically visualize the calculated flows in AR, VR, or on an ordinary screen. Some of the OpenVisFlows functions can be seen in the* OPENLBAR *application.*

There are several ways to make the results obtained by the simulation software usable for the library. In the first, we access the simulation software directly with functions. However, this only works if the simulation is running in near real-time. For this reason, this will be implemented at a later time . In the second solution, and thus the one pursued in this paper, ready-made simulations are provided externally. These are then to be displayed using different visualization techniques. To achieve this, a RES interface between web server and end device has been implemented. For the development of our visualization methods, we access the functionalities of UNITY and AR Core. To show the use of OpenVisFlow, an application named OPENLBAR was created with it. This is intended to allow students to interact with simulations of fluid flows and thereby gain a better understanding of the complex processes involved.

3.3 Realization

In this section, we first discuss the method for providing the simulation data. For this, we use a server as in [43] on which the data are stored ready for retrieval. What follows is the visualization of the CFD data on a QR code. Here, instead of the closed source extension Vuforia as in [43], the open-source variant Google ARCore is used. After that, the markerless geometry overlay is described. Since the making of 3D objects and the implementation of AR-Code can be a lengthy drawn-out process, we propose and implement a framework for an opensource library in order to simplify and streamline the visualization process. As already stated in Section 3.2, the goal of this library is to bridge the present gap between simulation software and visualization. Currently, in order to visualize simulations, one must resort to separate programs such as Paraview. It is our goal to implement a library so that simulation and visualization can be implemented and executed at the same time to allow for a more optimized and efficient workflow, and eventually to allow for real-time physical simulations that are interactive. As can be seen from Figure 3.1, the library will use well-known and already tested simulation software such as OPENFOAM and OPENLB, take the results thereof, and visualize them using diffent tools and devices.

3.3.1 Providing the Simulation Data

But creating simulation data and making it presentable can be very time-consuming and computationally intensive. Students or teaching staff usually do

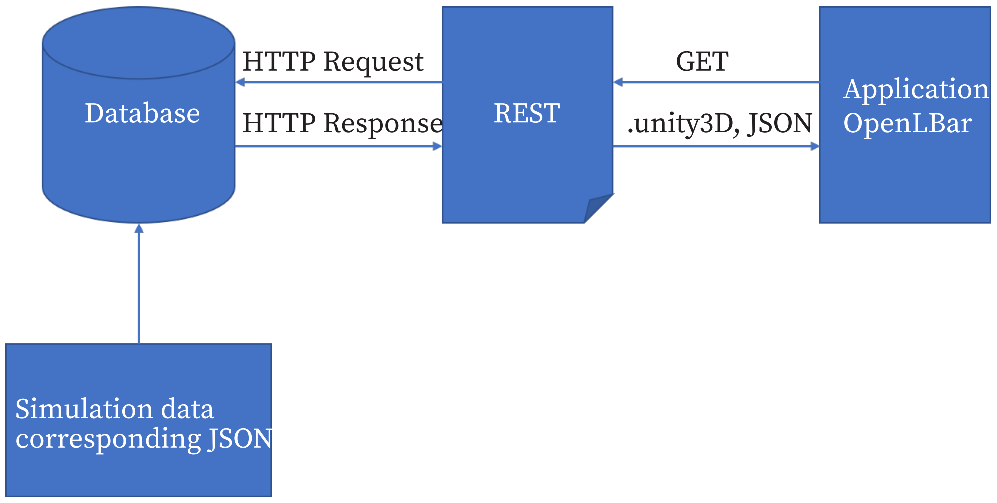
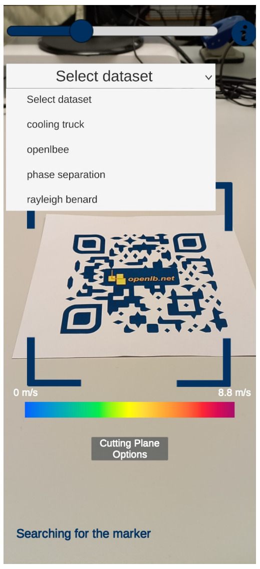


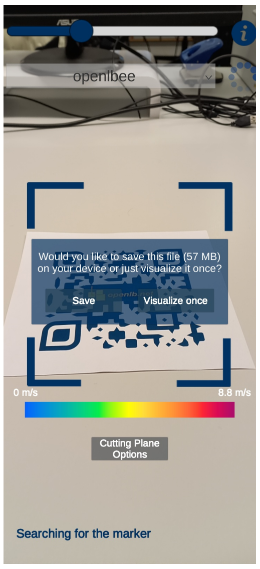
Figure 3.2: *Schematic representation of the requests and dataflows used in* OpenLBar *using a REST interface.*

not always have the time and/or expertise to create these themselves. Furthermore, simulations can take up a significant amount of memory. Consequently, this has a strong impact on the use of the application for teaching. For this reason, the application OpenLBar allows the simulation data intended for teaching to be loaded onto the smartphone via a cloud as shown in Figure 3.2. The data is provided by experts in the simulation field. Additionally, the user also has the option to store the simulation data locally or temporarily in the cache so that the storage space of the smartphone is not taken up by the transferred data. Figure 3.3 (a) shows that in the application, the simulations are selectable from a drop-down menu. The available data updates with changes on the cloud and does not need to be synchronized manually. The Figure 3.3 (b) shows the user options after selecting the desired simulation.

This functionality is realized through the WebApiDataManager of OpenVisFlow, which can be seen in Figure 3.4. When connected to a correct REST interface on a server, it will generate a list of accessible data sets which are then linked with a name through a dictionary, so that they can be displayed in a more user-friendly way. Using this, the name loading request can be started in three different ways. First, the data set in question can be downloaded with the `startFileToMemoryDownload` function from the server onto the device, saving it on the local drive for later use. With the `startFileToCacheDownload`, function OpenLBar can download the data sets into the cache, so that the data will be



(a) *Data selection options that can be viewed.*



Would you like to save this file (57 MB) on your device or just visualize it once?

(b) *User prompt to decide how to handle large files.*

Figure 3.3: *Selection of the simulation data to be displayed above the QR code seen in the background.*

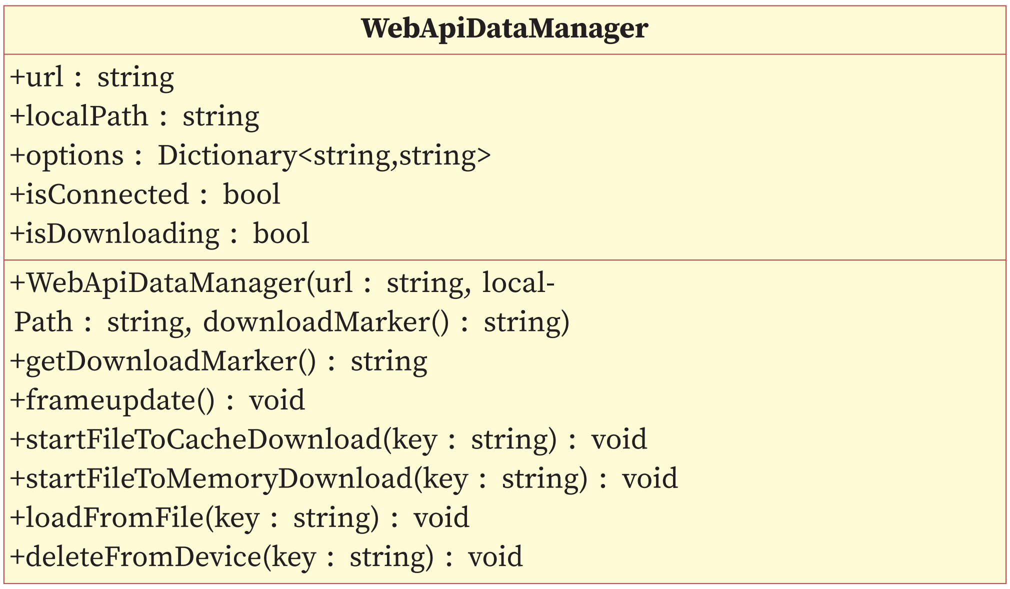
deleted after it is no longer used, so as to not fill up the devices storage. Lastly, if a data set has already been saved to local storage, it can be loaded by using the `loadFromFile` function. After loading has finished, the WebApiDataManager will initiate the transformation of the loaded simulation data into UNITY Game objects. This transformation will be covered more thoroughly in the appropriate section.

3.3.2 Visualization of the Simulations

This section presents the visualization methods that are currently provided by OpenVisFlow.

3.3.2.1 QR Code Tracking

Among other things, the application OPENLBAR allows one to visualize simulations above a QR code. As already mentioned, our application is based on UNITY and the open-source extension Google ARCore. For this particular case, it was decided to use the functionality trigger image, which allows one to over-



WebApiDataManager

+url : string
+localPath : string
+options : Dictionary<string,string>
+isConnected : bool
+isDownloading : bool

+WebApiDataManager(url : string, localPath : string, downloadMarker() : string)
+getDownloadMarker() : string
+frameupdate() : void
+startFileToCacheDownload(key : string) : void
+startFileToMemoryDownload(key : string) : void
+loadFromFile(key : string) : void
+deleteFromDevice(key : string) : void

Figure 3.4: *Class diagram of the WebApiDataManager with all essential functions and fields.*

lay objects and models over a given image. To this end, we implemented a QRCodeVisualizer class dedicated to tracking and managing trigger images, which in our case is a QR code. The corresponding class is shown in Figure 3.5. In essence, the QRCodeVisualizer checks if a trigger image is visible in every frame, and if yes, if we have already instantiated a SimulationData on it. To save resources, the QRCodeVisualizer checks conversely if simulation data has been instantiated on the trigger image and whether it is still visible. If this is the case, the simulation data is deleted as it is no longer needed.

As mentioned above, we need some kind of update function that is called at every frame and checks if the AR session is set up correctly as well as whether we are currently actively tracking with the camera. If this is the case, the `handleTrackables()` method can be called, where all tracked papers/OpenVisFlow22/Images are checked for visualizations. Depending on the situation, `addGameObject()` or `removeGameObject()` is invoked. Another part of the frame update function must be an update of all existing instantiated simulation data, where the scale and position can be changed. As for placement, the visualizer simply places the 3D-object at a certain distance above the QR-code, depending on its size, and scales the longest side of the object to the length of one side of the QR-code. Since the visualizer is responsible for the scale of our object, it is here where we implemented a zoom functionality with which one can use

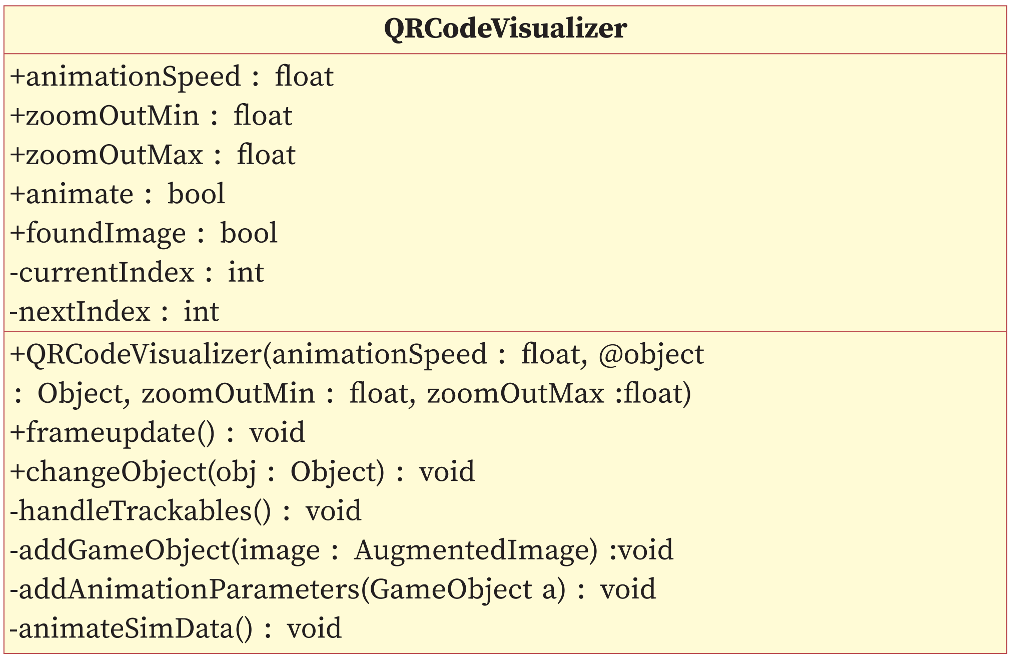

QRCodeVisualizer
+animationSpeed : float
+zoomOutMin : float
+zoomOutMax : float
+animate : bool
+foundImage : bool
-currentIndex : int
-nextIndex : int
+QRCodeVisualizer(animationSpeed : float, @object : Object, zoomOutMin : float, zoomOutMax :float)
+frameupdate() : void
+changeObject(obj : Object) : void
-handleTrackables() : void
-addGameObject(image : AugmentedImage) :void
-addAnimationParameters(GameObject a) : void
-animateSimData() : void

Figure 3.5: *Class diagram of the QRCodeVisualizer with an overview of the essential functions and fields.*

two fingers to enlarge or size down the simulation data and get a better view of certain areas of the Simulation results. The `zoomUpdate()` method takes user input and converts it into a linear scaling of the simulation data size, either up or downsizing it depending on the swipe direction. The `setMaterial()` method can be used to change the point sizes of the point cloud that is used to visualize the simulation results, or its color scheme, and the `changeObject()` method can be used to change the displayed 3D-simulation data to a different one so that one can view a different simulation. With this, we now have a working AR visualization tool for displaying simulation results.

3.3.2.2 Geometry Overlay

Visualizing simulations on a QR code is a handy way of viewing and interacting with complex processes to get a better understanding of the visualized process. However, in certain cases, it is advantageous to overlay the simulation results with an associated geometry to better and more intuitively understand the mentioned process. One of the biggest problems with this type of AR visualization is to figure out where to place the object, how big it is supposed to be, and what rotation needs to be applied. In , solid localization is needed for an AR application to be useful and convincing. For the result to be fully utilized, the visualized 3D data must be very close to the original in both position and scale. To achieve this, the functionality feature tracking was used. As the name suggests, this is a process in which distinguishable points are marked and tracked with the phone's camera. In this case, it was decided to use the built-in feature detection of Google's AR-Core Library. With it, OpenLBar is able to create a list of notable points in any room and track their positions even when the camera is moving. The next step is to determine the position and scale of our virtual object and fit it to the real object. We realize this by selecting two known points on the object and using the distance between them to scale the virtual object to the appropriate size. However, since we rely on feature detection to track points with the camera, we need to use points that are salient enough for AR-Core to detect. To make the selection process more consistent and user-friendly, we decided to use the edges of our object. The contrast on the edges of the real object allows reliable feature detection there. Thus, it is intuitively clear which points are to be used for scaling. For this edge selection, we implemented a user interface that allows the user to drag a slider over the correct location on the screen. When released, the system automatically finds the nearest feature point.

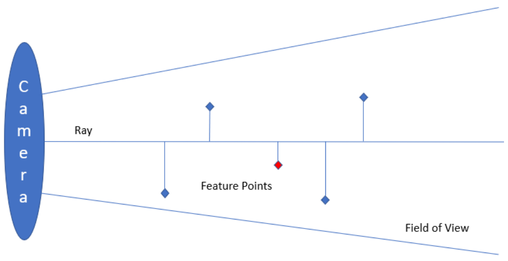


Figure 3.6: *A ray cast from the camera through a field of trackable feature points. The red point is the one chosen by the algorithm, since its distance from the ray is the smallest.*

This is achieved by casting a ray from the selected point on the screen and a ray along the view of the camera, as shown in Figure 3.6. The distance of each feature point to the thrown ray is then given by

$$d = \frac{||(p - O) \cdot dir||}{||dir||}, \tag{3.2}$$

with d representing the distance to the ray, $\boldsymbol{p}$ the location of the relevant feature point, $\boldsymbol{O}$ the position of the camera, and $\boldsymbol{dir}$ the direction of the ray which can be derived from the chosen screen point and the camera rotation. The feature point with the smallest distance to our ray is then selected. However, this method of point selection has one major drawback. Sometimes, this algorithm selects a point that is not at the same depth as the object because a point in the background is closer to the ray, as shown in Figure 3.7. This leads to significant errors in scaling and rotation. Therefore, this error must be taken into account, as shown in Figure 3.8.

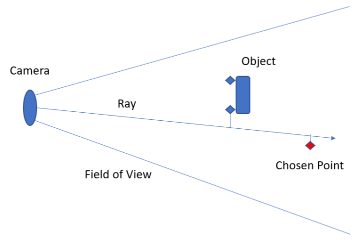


Figure 3.7: *Visualization of how the algorithm can chose the wrong feature point leading to large discrepancies in depth and subsequently scaling and positioning.*

To do this, we can consider the object as a plane perpendicular to the camera's line of sight and draw a line from the selected point to this plane. The intersection of these two is given by

$$\boldsymbol{P_a} = (\boldsymbol{P_2} - \boldsymbol{O}) \cdot \frac{\boldsymbol{P_1} \cdot \boldsymbol{n}}{(\boldsymbol{P_2} - \boldsymbol{O}) \cdot \boldsymbol{n}} \tag{3.3}$$

, where $\boldsymbol{P_a}$ is the depth-adjusted feature point, $\boldsymbol{P_2}$ is the feature point to be adjusted, $\boldsymbol{P_1}$ is the closest feature point to the camera, $\boldsymbol{O}$ is the camera position, and $\boldsymbol{n}$ is the normal of the plane and the camera view direction.

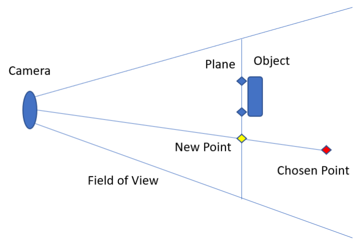
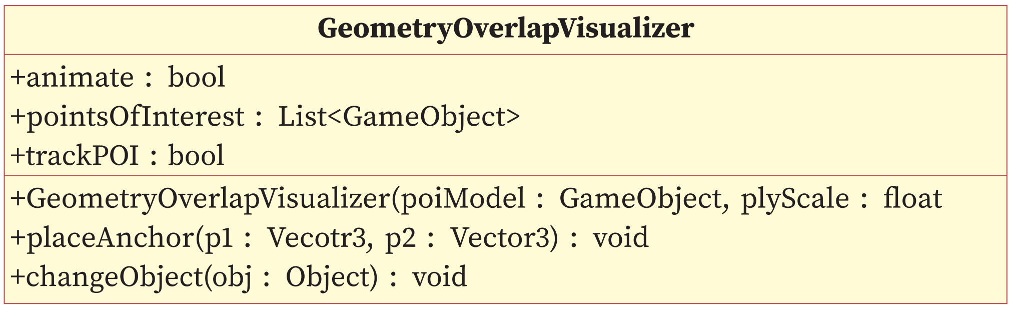


Figure 3.8: *Visualization of how the algorithm accounts for depth discrepancies and chooses a better suited point for the scaling and positioning of the virtual object.*

The intersection point can now be used as our depth-corrected point to place and scale the virtual object. This results in a much smaller error. We always use the point closest to the camera. The reason for taking this approach is that it is unlikely that the user will have many significant contrasts between himself and the object being viewed. This makes it much more likely that it is the point that is placed on the object. The placement and scaling itself can be done by calculating the scale factor, which is given by

$$scaleFactor = \frac{modelLength}{realLength}, \tag{3.4}$$

and we can position it in the middle of the two selected feature points. For objects with a wide base, we are still left with a significant depth error because placement is at the front end of the object. However, since we know how wide the base of the object is, we can move the z-axis back by half the length to compensate for this.



GeometryOverlapVisualizer

+animate : bool
+pointsOfInterest : List<GameObject>
+trackPOI : bool

+GeometryOverlapVisualizer(poiModel : GameObject, plyScale : float
+placeAnchor(p1 : Vecotr3, p2 : Vector3) : void
+changeObject(obj : Object) : void

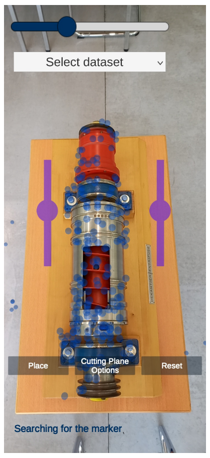
Figure 3.9: *Class diagram of the GeometryOverlapVisualizer with all relevant functions and fields.*

The GeometryOverlapVisualizer implements this functionality using points in worlds space as anchors, which are set with the `placeAnchor()` method. Figure 3.9 shows all functions that are relevant in order to use the GeometryOverlapVisualizer.

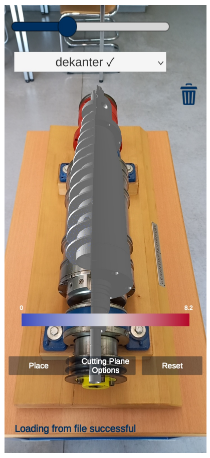
The Figure 3.10 (a) and Figure 3.10 (b) show the results using a real decanter and the CAD counterpart.

3.3.2.3 Animation

The simulation software OpenLB, which is mainly used for the creation of the simulations, outputs the simulation results as a collection of files of the type visualization toolkit (VTK). Each of these files represents a time step of the



(a) *Decanter pre-placement with visible feature point tracking.*

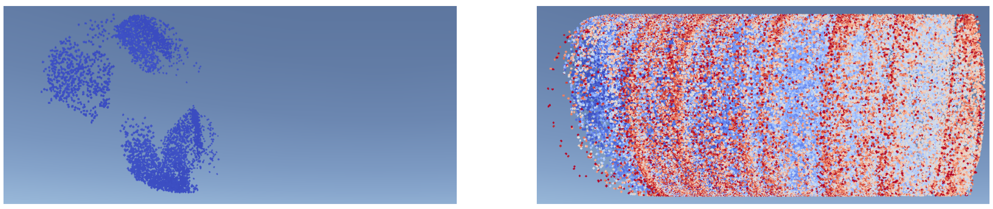


(b) *Decanter overlayed over real decanter centrifuge.*

Figure 3.10: *Example of geometry overlay on a decanter centrifuge.*

simulation in question and contains all data in regards to the calculated fields. Using programs like Paraview, this data can then be converted into different 3D data types like the binary GL transmission file format (GLB), or the polygon file format (PLY). On the one hand, the GLB file format could be used. Due to the possibility to save rotations and translations with the 3D file, this is a widely used file format in the AR/VR scene. However, it is not suitable for our purposes because the particle motion of simulations can be very complex. This makes it very costly to store our simulations in a GLB file. For this reason, we decided to convert the data of each time step into a PLY 3D model. In this file format, the 3D data is built up from polygons. They are stored in a list with vertices, triangles, and edges. In this way, we get the state of each time step as a 3D file, which can then be shown one after another, creating an effect similar to a stop motion animation. Figure 3.11 shows this with the example of a simulation of particles from a decanter. The first time step is shown in Figure 3.11 (a) and the last is shown in Figure 3.11 (b).

In order to achieve the aforementioned effect, a child object named simulation data, with all of the .ply meshes attached as children, is added to our main object. We can then circle through the meshes activating the next and deactivating the current mesh in order to achieve a fluid animation. Since the method of animation is specific to the case, and we want the library OpenVisFlow to be



(a) *First time step.* **(b)** *Last time step.*

Figure 3.11: *Results of a particle simulation in a decanter centrifuge, visualized using a polygon file format. (**a**) shows the first time step and (**b**) the last time step of the simulation. With each time step, the particles migrate further.*

as expandable as possible, the implementation for the animation falls into the case-specific visualizers, in this case the GeometryOverlapVisualizer and the QRCodeVisualizer. In this way, it is possible to take advantage of different file formats and animation methods using different visualizers.

QRCodeVisualizer
+animationSpeed : float +animate : bool -animationIndex : int -previous : int -nextFrame : float
+QRCodeVisualizer(animationSpeed : float, @object : Object, zoomOutMin : float, zoomOutMax :float) +frameupdate() : void -addDependencies(obj : GameObject) : void -addAnimationParameters(GameObject a) : void -animateSimData() : void

Figure 3.12: *Class diagram of the QRCodeVisualizer with all relevant fields and functions regarding animations.*

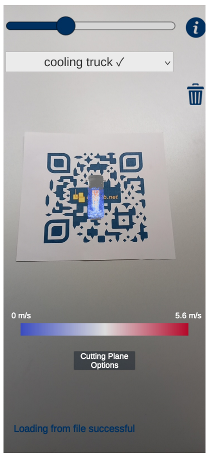
As seen in Figure 3.12, the animation is controlled using the public float `animationspeed` and automatically set up for new simulation data, using the `addAnimationParameters` method, when the `changeObject()` function is called. The animation is then started with the `animateSimData()` method, using a background thread, so that it does not become frame-dependent when loading sequences and other calculation-intensive operations are launched.

3.3.2.4 User Interactions

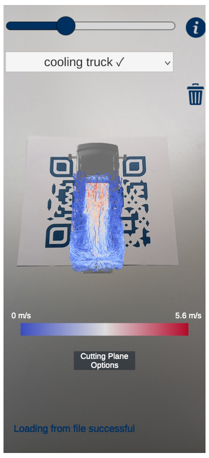
User interactions are very important to give users, a better experience and to make them have more fun exploring. They also help them better understand the results of the simulations. By its very nature, augmented reality provides the ability to view the simulation results from different angles. However, from our perspective, the benefits of AR can be better utilized if additional engaging features are implemented. As already mentioned in Section 3.3.2.3 , we offer users the possibility to control the speed of the simulations themselves. Thus, for example, the user is able to slow down the simulations in order to look at and understand parts of them in more detail. Furthermore, we have implemented a zoom function so that the user can scale simulations as desired, making it possible to focus in on specific areas to deepen ones understanding of the processes at work. To give our application a bit more specificity, we also decided to implement a color-legend to make it easier for users to actually use these images to figure out how fast the fluids move at which point in their object. Figure 3.13 shows the zoom function and color legend using the visualization of a cooling truck and a bee. Finally, we wanted to further improve the ability to view virtual data on a real object. To this end, we implemented a way of visualizing only parts of the object. Basically, one can control which parts of the simulation will be shown and which parts will just show the outside wall of the object, giving one the options of seeing cross section shots and isolating area for inspection, which otherwise would not have been visible behind layers of simulation data. This works by putting in what we call a cutting plane. If a point is on the visible side of it we display it, if it is on the other side we set its alpha to zero, in essence making it invisible. The results of this can be seen in Figure 3.14 . Future versions may include more differentiated methods of defining the visible areas. At the moment, the cross sections are made visible with shaders written by Abdullah Aldandarawy out of the Cross Section Shader addon from the UNITY Asset Store.

3.3.3 Open-Source Library

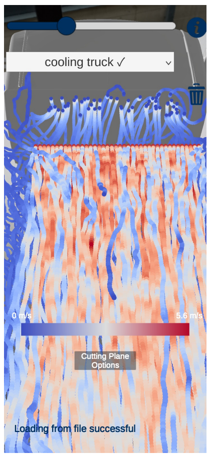
UNITY is a powerful tool for developing games and AR/VR applications. We see a great potential for teaching, but also for industry to visualize complex processes. This motivated us to start developing a modular open-source library that allows users to quickly, easily, and efficiently develop AR/VR applications for complex processes like fluid flow. As this paper is also the start of the library,



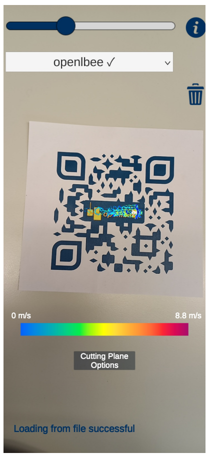
(a) *Cooling truck zoomed out.*



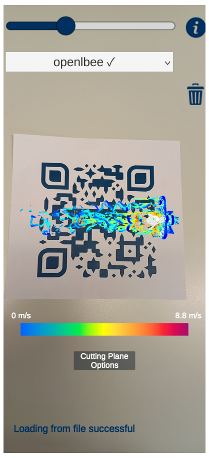
(b) *Cooling truck medium zoom.*



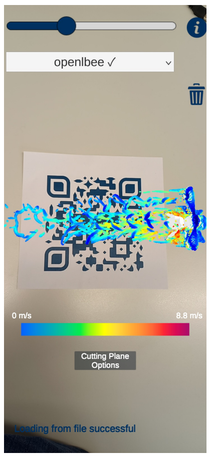
(c) *Cooling truck zoomed in.*



(d) *OLBee zoomed out.*

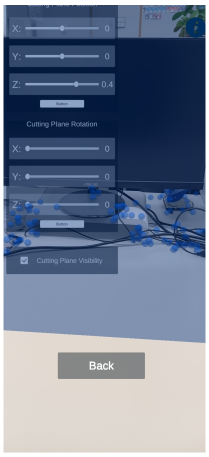


(e) *OLBee medium zoom.*

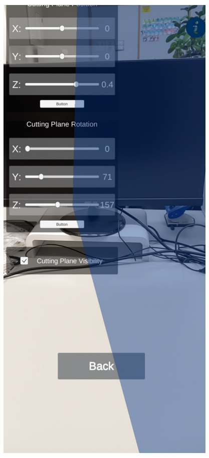


(f) *OLBee zoomed in.*

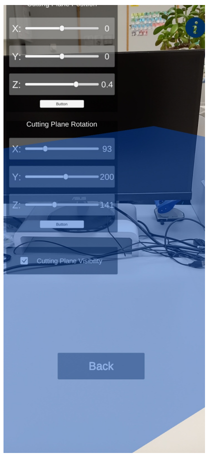
Figure 3.13: *Visualization of the simulations on the QR code at different zoom levels. Figure 3.13 (a) - 3.13 (c) illustrate this for the cooling truck, while Figure 3.13 (d) - 3.13 (f) show it for the OLBee.*



(a) *CuttingPlane Frontal orientation.*



(b) *CuttingPlane Rotated sideways.*



(c) *CuttingPlane Rotated over all axes.*

Figure 3.14: *Implementation of the CuttingPlane UI by Abdullah Aldandarawy from the Cross Section Shader Addon from the* UNITY *Asset Store.*

it currently only contains the functionality to create the application OPENLBAR presented here. Further functionalities like an interface between UNITY and the simulation software such as OPENLB will be added soon. In the following sections, the already existing classes and methods of the library are explained. Furthermore, we show how the application OPENLBAR can be easily reproduced with its help.

3.3.3.1 Library Overview

As previously stated, the main goal of this library is to simplify the development process of visualization tools. The library is structured around a main workflow, as can be seen in Figure 3.15. This consists of three different parts: The DataManager, the Transformer, and the Visualizer. In OpenVisFlow, the DataManager, as its name implies, implements the acquisition and management of incoming data. It is supposed to act as a bridge between our data source and our application to get the right data at the right time. The goal is to keep the main workflow as simple and expandable as possible, so the classes DataManager, Transformer, and Visualizer have been implemented as abstract requiring only a few key features and leaving everything except the essentials to a case by case implementation of an inherited class. As cases can also vary in the data types required, the implementation of templates was unavoidable, both the Transformer and the Visualizer are dependent on a certain input type of data, which is why both of them are templated to their respective input datatype T. In our example we only implemented a single data manager, the WebAPIDataManager, for both cases. The WebAPIDataManager allows the sending of web requests and the subsequent saving and/or loading data as binary Files. Next in line is the Transformer, which takes the provided data and transforms it into a form needed by the visualization side. In the example we implemented a ByteToAssetTransformer, which takes in a byte array and generates a UNITY GameObject from it, which can then be used by the Visualizers. As the name suggests, the Visualizers implement the visualization side of the program. In it is the code for animation, placement and rotation of the Objects in the World. In our Example case we use two different Visualizers depended on the case. The QRCodeVisualizer enables Visualization above a Triggerimage, in our case a QRCode. Whenever a Triggerimage is found it will place the GameObject the Transformer supplied right above it. It also implements scaling between a min and max value as well as a public variable for animation speed. For the other case we implemeted the GeometryOverlapVisualizer, which allows one to overlay a Object in the real World with a virtual twin, by supplying two of the edge points of the object. As of now the Library will only work with UNITY as we use UNITY GameObjects and AR-Core to realize the AR-visualization. We hope to expand this to Android studio and other Development tools in the Future.

As seen in the Figure 3.15 to make OpenVisFlow work a programm needs a class pulling all ties together. This case File needs to hold references to DataManager, Transformer, and Visualizer and is repsonible for controlling all the different aspects of our app through a UI. It also needs to implement some sort of frame update function and call the `frameUpdate` method of the Visualizer and the DataManager to properly function. The following section will contain more details about the aforementioned classes.

3.3.3.2 DataManager

As stated above the DataManagers main objective is the Acquisition and/or managment of simulation data. Since this data can be obtained in a variety of different ways and file formats, the DataManagers give the users an easy gateway into the library. The Figure 3.16 shows the WebApiDataManager with all of its functions and how it implements the standard DataManager.

The WebApiManager is supposed to acquire a list of available simulation

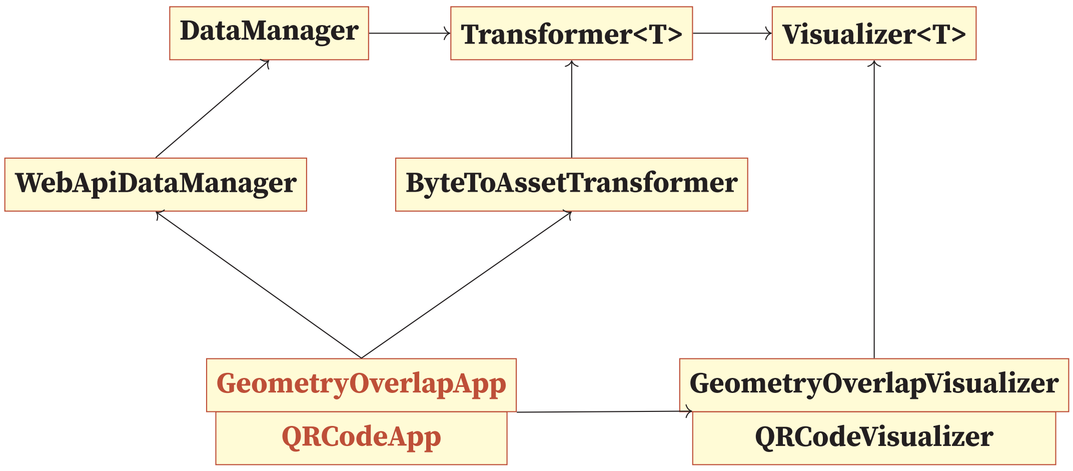


Figure 3.15: *Overview of the classes that are currently implemented in the library and how they relate to each other. The red classes are the case classes implementing the library and managing the UI.*

data from our web server, which it can then download and manage on the local drive of the device. To achieve this, the WebApiManager has two main parts, the first being the populateOptions() function. In it, we check the REST-API and catalog all the available options and compare them to the found data on our Device. These options can then be accessed and displayed in whatever way the developer sees fit. Secondly, it gives the developer the option to use these options to initiate a download or local load routine and the following transforming and displaying of said data.

3.3.3.3 Transformer

Next, we will take a look at the Transformer class. It is meant as a bridge between Visualizer and DataManager where we can take what-ever data is supplied from the DataManager and convert it into whatever form the Visualizer needs. We decided to make an extra Transformer class in order to make it easier to write and add in ones own Transformers and to keep a cleaner structure for better overview. In the case below, we read a byte array from a REST website API and need to convert it into a Unity object in order to visualize it using AR-Core. To this end, the Transformer was implemented as an abstract class templated to the datatype it needs to receive from the DataManager. The child then implements both the templated datatype as well as the transformData function which can

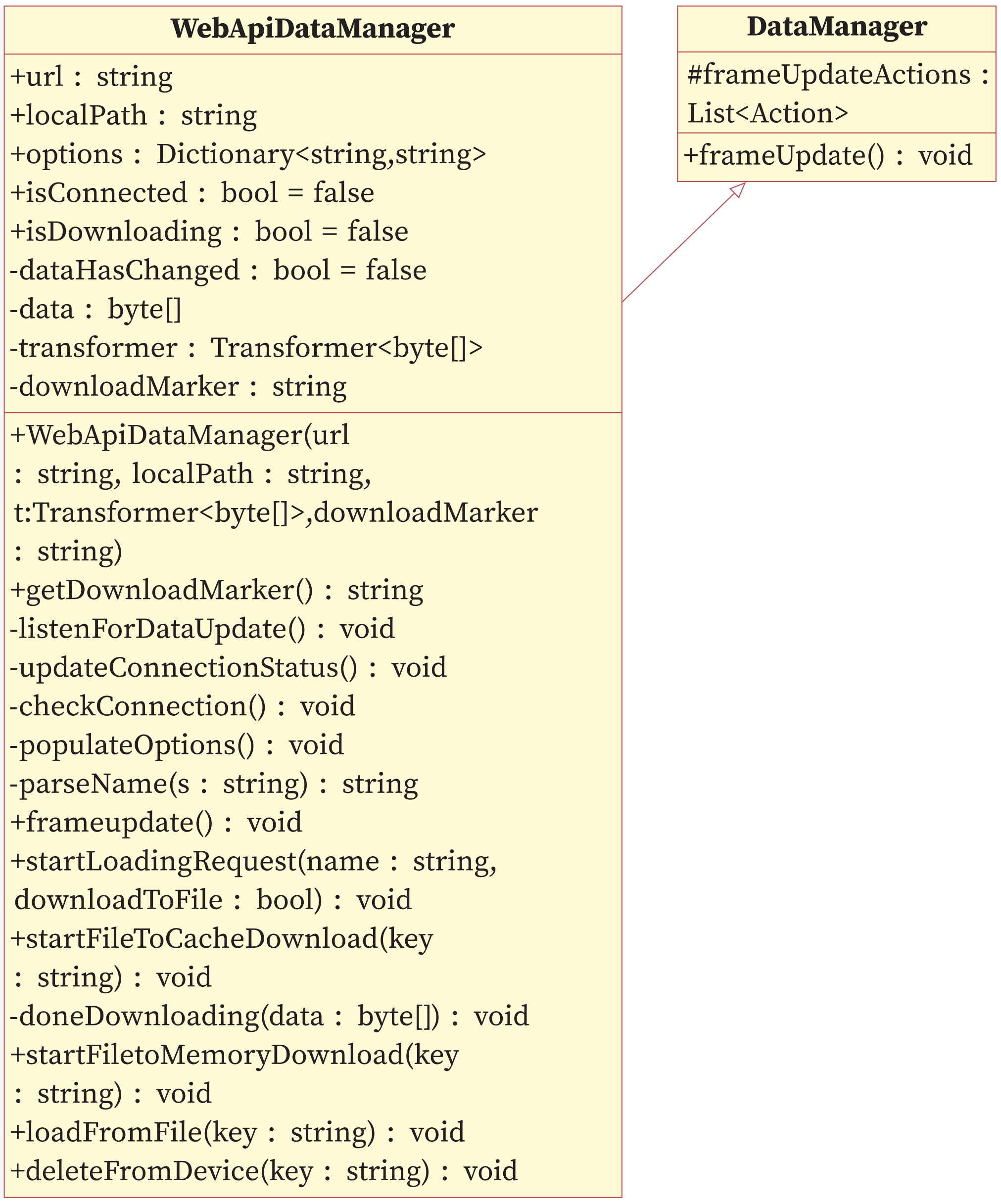

WebApiDataManager
+url : string
+localPath : string
+options : Dictionary<string,string>
+isConnected : bool = false
+isDownloading : bool = false
-dataHasChanged : bool = false
-data : byte[]
-transformer : Transformer<byte[]>
-downloadMarker : string
+WebApiDataManager(url : string, localPath : string, t:Transformer<byte[]>,downloadMarker : string)
+getDownloadMarker() : string
-listenForDataUpdate() : void
-updateConnectionStatus() : void
-checkConnection() : void
-populateOptions() : void
-parseName(s : string) : string
+frameupdate() : void
+startLoadingRequest(name : string, downloadToFile : bool) : void
+startFileToCacheDownload(key : string) : void
-doneDownloading(data : byte[]) : void
+startFiletoMemoryDownload(key : string) : void
+loadFromFile(key : string) : void
+deleteFromDevice(key : string) : void
DataManager
#frameUpdateActions : List<Action>
+frameUpdate() : void

Figure 3.16: *Class diagram of the WebApiDataManager with all the functions und fields and its connection to its Parentclass.*

then be called in the DataManager to start the transformation routine. This makes it easier for developers to see which Transformers already exist. At the same time, it is easier to work with a vast variety of different data types. The functions and fields of the transformer as well as its connection to its parent can be seen in Figure 3.17. How the Transformer interacts and relates to the rest of the library can be seen in Figure 3.15.

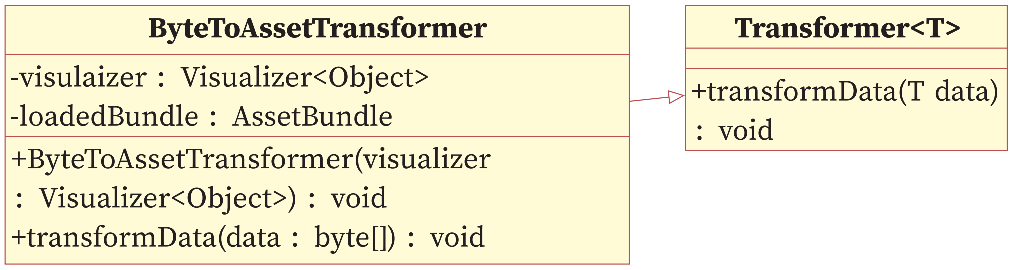

ByteToAssetTransformer
-visulaizer : Visualizer<Object>
-loadedBundle : AssetBundle
+ByteToAssetTransformer(visualizer : Visualizer<Object>) : void
+transformData(data : byte[]) : void
Transformer<T>
+transformData(T data) : void

Figure 3.17: *Class diagram of the ByteToAssetTransformer with all the functions and fields and its connection to its parent class.*

3.3.3.4 Visualizer

As the name suggests, the Visualizer serves to implement visualization of the data. Since the visualization methods vary widely, the Visualizer needs to be unrestricted by default so the only important decision to make is what datatype to use for your visualization and template the child class to it. We only implemented two examples to visualize 3D-models in AR using AR-Core, the class diagrams of which can be seen in Figure 3.18. In this case we use the UNITY object type as our basis for visualization. In the future, we want to implement a variety of different visualization options using VR, AR, and other technologies.

3.4 Results

In this section, the appearance and fluidity of the resulting animations are discussed. Furthermore, an evaluation with students has been performed to determine student's opinion concerning OPENLBAR and AR/VR simulations. Finally, we look at the stability of the QR tracking as well as the accuracy of the manual geometry overlay.

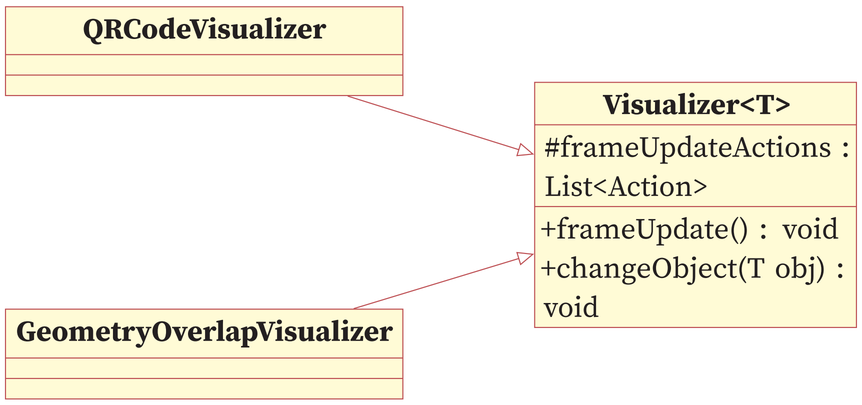

QRCodeVisualizer
Visualizer<T>
#frameUpdateActions : List<Action>
+frameUpdate() : void
+changeObject(T obj) : void
GeometryOverlapVisualizer

Figure 3.18: *Class diagram of the GeometryOverlapVisualizer with all essential functions and fields and its connection to its parent class.*

3.4.1 Quantitative Results

In this section, the OpenLBar application created with OpenVisFlow is tested for its suitability with the help of evaluation forms. The evaluation was conducted with 13 master students. Most of the respondents were studying chemical and process engineering. The rest of the respondents have studied either mathematics or computer science. The evaluation forms consist of three sets of questions. The first set is primarily about the handling of the app. Since a practical application must be easy to use, the installation process and the occurrence of errors or circumstances are mainly queried here. In addition, the first set also asks to what extent the application appeals to the user. In the second set of the survey, sense and usefulness are examined. For this purpose, the respondents were asked to indicate, for example, to what extent they see OpenLBar as a useful supplement to the lectures. In addition, they were asked whether the understanding of engineering problems, especially in the field of fluid dynamics, is simplified by the application. In the third set, exercise tasks were to be examined. For this purpose, the subjects were asked to evaluate the extent to which the exercises provided them with new insights into fluid mechanics. Below, the evaluation of the question sets is discussed.

3.4.1.1 First Set of Questions

Figure 3.19 shows a diagram of the evaluated the questions on usability. As can be seen, the OpenLBar application is an easy-to-use application in the eyes of the students. It is positive to see here that overall, there is very strong

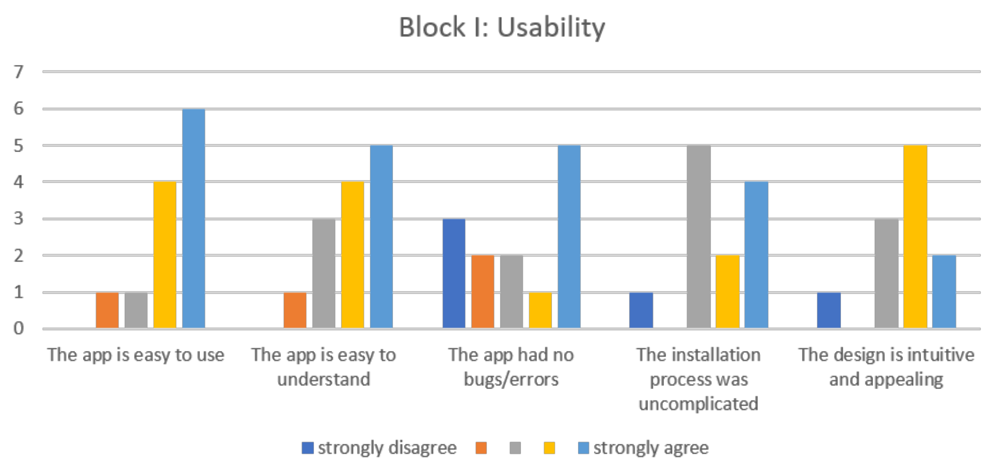


Figure 3.19: *First set of questions regarding the usability of* OpenLBar.

agreement among the respondents. In addition, it is evident from the survey that OpenLBar is easy to understand by the majority, but a certain proportion of respondents encountered comprehension problems. Regarding the question as to whether the application had errors, a similar picture emerges: The majority of the students is of the opinion that no errors occurred, however, a large part of them express having encountered problems. The answer as to whether the installation process ran smoothly was answered positively or neutral by most of the respondents in the positive, whereby the neutral attitude made up for the largest share of the results. The last question in the first block, as to whether the design of the application was intuitive and appealing, was answered relatively diffusely. It should be noted that there is a small tendency toward agreement, which is why we assume of a positive tendency. Overall, the usability seems to be good, and the application is very easy to understand due to its design and simplicity, while there is still room for improvement in the occurrence of errors.

3.4.1.2 Second Set of Questions

The questions from the second block will be used to evaluate the usefulness of the OpenLBar app. The data obtained is shown in Figure 3.20.

The majority of the students were of the opinion that OpenLBar should be included in the lecture course. It is noticeable that no student was negative toward this statement. At the same time, the neutral stance makes up for the

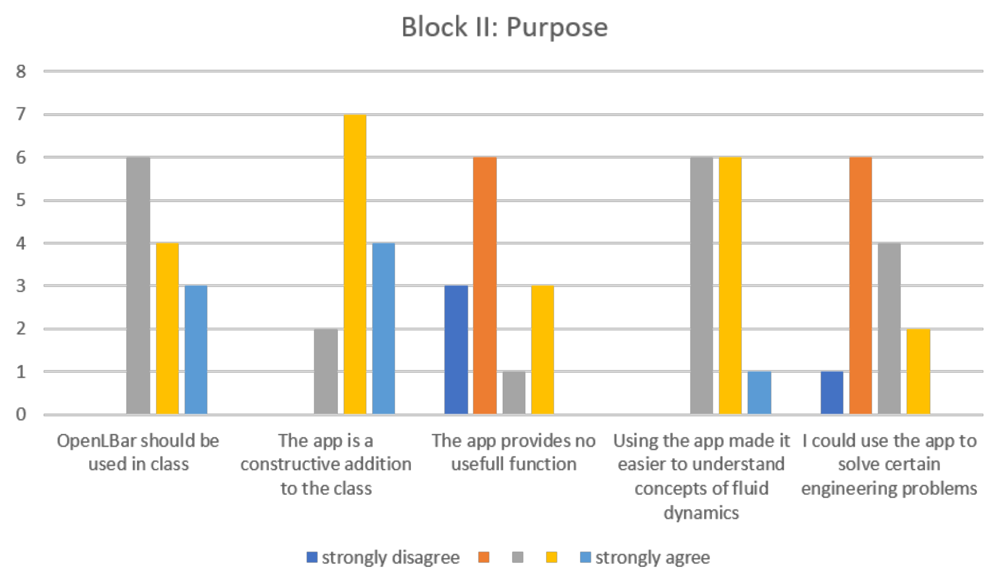


Figure 3.20: *Second set of questions regarding the purpose of* OPENLBAR.

majority of the survey results for the first question. Likewise, a majority of respondents believe that the application is a useful addition to the lecture. Here, too, it is positively noticeable that there are no negative evaluations and there is a strong tendency toward strong agreement. The third question, i.e. as to whether the application does not provide a useful function, was answered quite diffusely. However, it should be noted that the question has been deliberately negated in order to stimulate active thinking. Despite the change in question type, it is positively noticeable that the majority of students think that OPENLBAR has useful functions. In addition, responses to the question about simplifying the understanding of fluid-named concepts were diffuse as well. Although there is a tendency to agree here as well, it should be noted that there are also many neutral attitudes. One person even completely agrees with this statement. Finally, the last question evaluates the usefulness for solving engineering problems. Most of the students cannot imagine using OPENLBAR to solve certain problems. For solving engineering-specific problems, as shown below, students primarily want customizable controllers and parameters. However, they assign high utility to the functionality of the application. Overall, students rate the applications, usefulness as positive. It is noteworthy that some students would like AR applications to be included in academic courses.

3.4.1.3 Third Set of Questions

On the evaluation sheet, the students were given tasks to be answered using the application. The first task asked where the air conditioner was located in the cooling truck. The second task asked what conclusion could be drawn about the flow rate of the bee. Afterwards, questions were asked regarding the completion of the tasks. Figure 3.21 shows the data obtained.

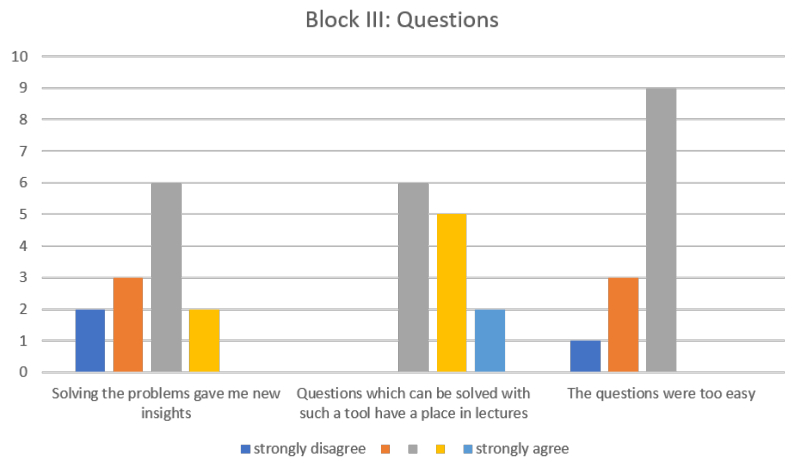


Figure 3.21: *Third set of questions regarding the applicability questions.*

As explained above, the scope of the problems is exactly two examples. On the one hand, the "cooling truck" and on the other hand, the "olbee". The first question on these tasks aims at gaining knowledge and should clarify whether the students have gained new insights into fluid dynamic concepts by working out and solving the case studies. Here, it can be seen that overall, there is a neutral to negative attitude toward the question. Next, the second question evaluates to what extent general AR applications are a good tool or aid for problem solving. Here, it is positively noticeable that, in addition to the general tendency to agree, a small group of two people even agree very much. Lastly, the students are of the opinion that the questions were not too easy, with the note that the majority has a neutral attitude. Overall, we can conclude, that the application can be useful for solving such tasks.

3.4.1.4 Free-Response Questions

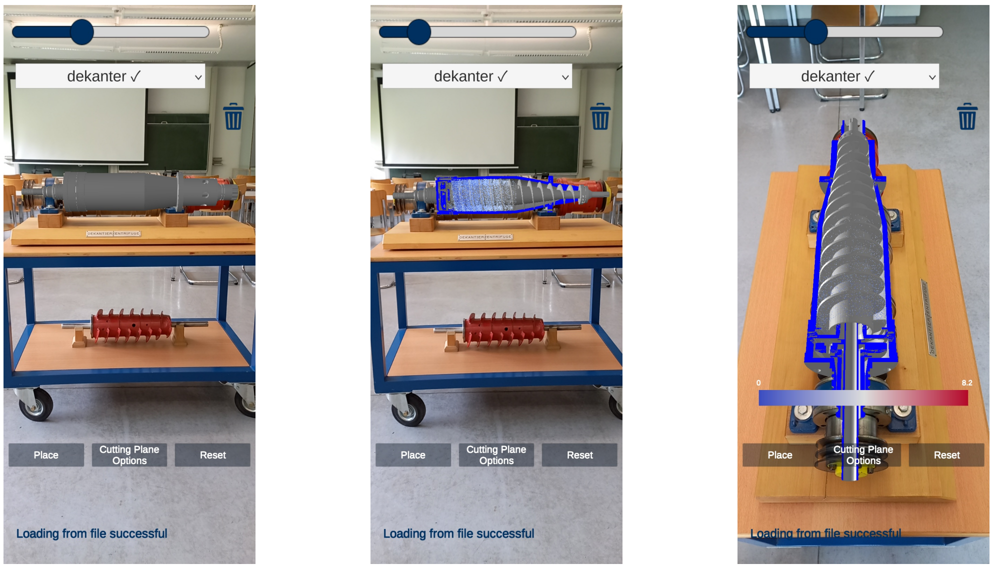
In the last questionnaire block, students were offered free text blocks to express comments, additions, and other remarks. This is aimed at obtaining more accurate feedback. The question whether the students encountered difficulties in general was answered by saying that there are still some bugs that need to be fixed. Furthermore, suggestions for improvement were also made. The next question asked was as to whether AR/VR applications should be integrated into teaching. The overall response was very positive about including modern technologies in teaching. The answers to the question of whether OPENLBAR should be used in other lectures were also positive. At last, the question was asked what additions to the application would be desirable. Here, it is noticeable that the students would like to adapt the application more according to their needs.

3.4.2 Qualitative Results

In Figure 3.23, two example simulations have been visualized. Figure 3.23 (a) - Figure 3.23 (d) show a cooling truck, and Figure 3.23 (e) - Figure 3.23 (h) show the simulation of a bee from different angles. It can be seen that the simulations do not have any noticeable flaws at first glance. Furthermore, with the help of the color legend, one can roughly identify the velocities of the flows. In the case of cooling trucks, for example, the velocities are fastest at the exit from the air-conditioning unit mounted on the ceiling of the cargo hold. It can also be seen that the flows slow down over time and approach a constant speed. In the case of the bee, it can be observed that the velocities are zero at the apex of the wings and fastest just beyond. Figure 3.22 shows the geometry overlay function of a decanter simulation on a real decanter centrifuge. It can be seen that the simulation covers the real object well and all parts of the simulation are still well recognizable.

3.4.3 Stability

QR code tracking and placing simulations usually do not show any stability problems. However, when viewing the marker at an angle, the position of the object to be placed may not fit exactly. This can be seen for example in Figure 3.23 (b). If this is ignored, the application is stable in QR mode. On the other hand, the stability of the geometry overlay highly depends on the lighting conditions. The worse the light, the fewer reference points are detected, which



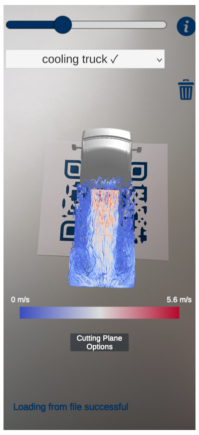
(a) *Decanter Front Uncut.* **(b)** *Decanter Front Cut.* **(c)** *Decanter Top Cut.*

Figure 3.22: *Results of geometry overlay using a decanter centrifuge as an example.*

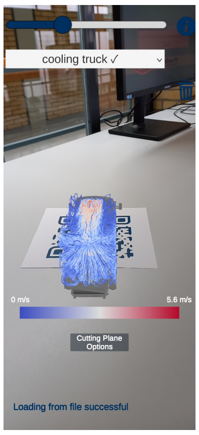
enable the geometry overlay. In addition, after placing the simulation, strong camera movements can cause the simulation to shift and no longer cover the real geometry.

3.5 Discussion

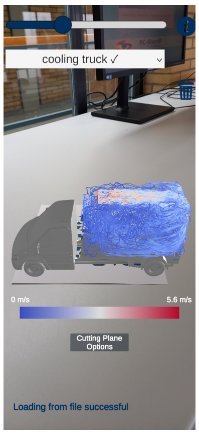
The results presented in Section 3.4 show that the OpenLBAR application is able to visualize simulation data in a sophisticated way for the eye and allows the user to interact with it. This includes, on the one hand, the ability to view the simulations from different angles, given by AR by default. Not to be neglected, however, is the zoom function for looking at parts of the simulation more closely, the cutting plane for removing 3D data that is in the way, and the color legend that allows the user to develop an understanding of the velocities of the moving flows of the simulation. In our opinion, this application is good for the user to get insight into complex processes, which are usually not visible. In universities and schools, often only the theory of some topics is taught in class, as visualization is sometimes only feasible with difficulty. Thus, this is especially a useful tool for education to provide students with a representation to theory in the form of simulations. We also see the potential benefit for engi-



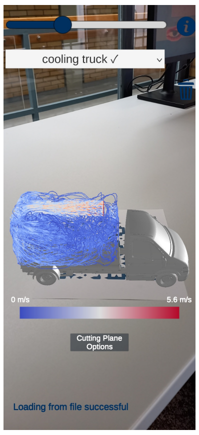
(a) *Cooling Truck Top-down View.*



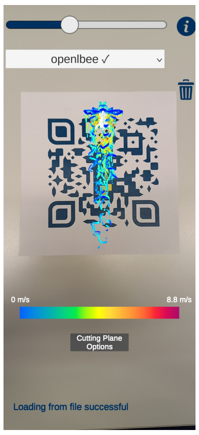
(b) *Cooling Truck Back Side View.*



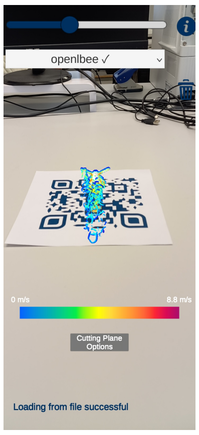
(c) *Cooling Truck Left Side View.*



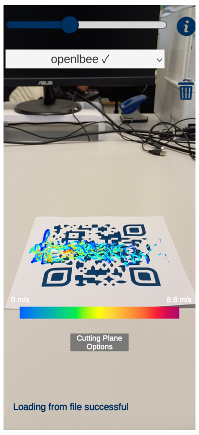
(d) *Cooling Truck Right Side View.*



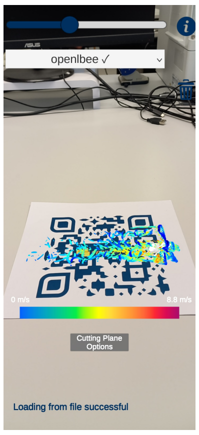
(e) *OLBee Top-down View.*



(f) *OLBee Back Side View.*



(g) *OLBee Left Side View.*



(h) *OLBee Right Side View.*

Figure 3.23: *Viewing the results on the QR code from different angles. Figure 3.23 (a) - 3.23 (d) show this using the cooling truck, and Figure 3.23 (e) - 3.23 (h) using the OLBee.*

neers to see the simulations from a new perspective, as well as a tool to bring the simulations and possible problems closer to the customer. However, AR visualization is not without problems. While it provides one with an easy-to-use intuitive way of viewing simulation data, it is not as well suited to actually work with. Whilst generating graphs rendering videos and such functions could all be implemented, they would all come with a significant drawback in usability, since the phone's touchscreen surface is not the best tool for fine-tuning such selections. Furthermore, AR visualization is dependent on how stable the AR side of the application works. If the tracking does not work properly, the scene will shake and turn in unnatural ways, making proper AR visualization impossible. These problems were solved with the OpenVisFlow library, as it includes a stable AR implementation and provides an interface to ready-made simulations. Thus, the data to be used can be processed externally with powerful tools such as Paraview or Blender.

With this paper the framework of this vizualization library has been implemented and the with this created application shows a stable and reliable way of visualization. The evaluations in Figure 3.4 show that there is a demand for application which allow users to interact with CFD simulations in AR. However they also stated, that they want customizable simulations. This can not be achieved with precompield simulation. Therefore, the main goal of this library is to implement visualization tools and including real time visualization. To achieve this the next step in the development of this libaray will be the implementation of cluster oriented tools to visualize simulations just in time.

4
Virtual CFD Lab: Mobile Just-in-Time Fluid Flow Simulation

This chapter is based on the published article

Teutscher, D. et al. "Just-in-Time Fluid Flow Simulation on Mobile Devices Using OpenVisFlow and OpenLB." in: Applied Sciences *14.5 (2024).* ISSN*: 2076-3417.* DOI*: 10.3390/app14051784.*

The abstract has been omitted, and formatting has been adjusted to fit the thesis style. The main content remains unchanged.

4.1 Introduction

Computational fluid dynamics (CFD) has been a vital tool for understanding fluid behavior across various industries and academic domains. However, the practical application of CFD has been limited by the long computational time and complexity involved in building a simulation setup. In recent years, the integration of CFD into various software, including CAD, 3D computational graphic software like BLENDER version 4.0.2 [46], and game engines like UNITY version 2023.2.10 [47] and UNREAL ENGINE version 5.3[48], has become more prevalent due to its usefulness in observing fluid flow behavior.

Various researchers have proposed innovative methods for applying CFD to different areas of interest. Mathias Berger and Verina Cristie [49] proposed using game engine technology to bridge the gap between architects and engineers in evaluating the effect of buildings on urban climate through CFD methods. Jos Stam [50] presented a rapid implementation of a fluid dynamics solver for game engines based on the physical equations of fluid flow, emphasizing stability and speed for *just-in-time* performance. Wangda Zuo and Qingyan Chen [51] proposed the Fast Fluid Dynamics (FFD) method as an intermediate approach between nodal models and CFD, providing much richer flow information while being 50 times faster than CFD for conducting faster-than-just-in-time flow simulations for emergency management in buildings. Angela Minichiello et al. [52] introduced a mobile instructional particle image velocimetry (mI-PIV) tool for smartphones and tablets running Android that provides guided instruction to learners, enabling them to visualize and experiment with authentic flow fields in real time. Jia-Rui et al. [37] explore the use of augmented reality (AR) technology in mobile devices for visualizing and interacting with CFD simulation results in the context of indoor thermal environment design. Harwood et al. [53] developed a GPU-accelerated, interactive simulation framework suitable for mobile devices, enabling the visualization of flow around particles.

The Lattice Boltzmann Method (LBM), a versatile computational technique for simulating fluid flow, is integral to our approach. Renowned for its capability to handle intricate geometries, multiple phases, and mesoscale phenomena [54], the LBM operates on a lattice grid, utilizing probability distribution functions to model fluid behavior, and it is adaptable for parallel processing on diverse platforms. Recent advancements include the integration of multiple-relaxation-time schemes to enhance stability and efficiency, along with extensions for simulating thermal and multiphase flows [55, 56]. The LBM finds application in various scenarios, such as solving evaporating and boiling problems [57], observing particle behavior in particle-laden flows [58], simulating heat transfer [59], and more.

Previously mentioned work [53] utilized a static domain with fixed inlets and outlets to create 2D simulations on tablets, which is limited to NVIDIA GPU-based devices.

In this paper, we present a new application called PAINT2SIM version 0.1, which extends the capabilities of OPENVISFLOW version 0.1 [1], a visualization library that introduced novel solutions to the challenges of a long computational time and complexity when targeting mobile devices. Leveraging the power of the LBM, PAINT2SIM utilizes the open-source library OPENCV version 4.8.1 [60] to enable on-the-fly, *just-in-time* 2D simulations using the camera of a mobile device. Our approach is not limited to specific NVIDIA GPU-based mobile devices but is applicable to all Android devices. Furthermore, we incorporate AR capabilities through the scanning of physical objects. Moreover, in this paper, we compare the performance, demonstrating better results even while utilizing only one core.

The aim of this approach is to enable users to generate a digital twin of a fluid domain using hand-drawn sketches, effectively converting their mobile devices into virtual laboratories for fluid dynamics. The objective is to facilitate the scanning of a simulation domain and provide real-time visualization of calculated results *just in time*, eliminating the need for precompiled simulations or specialized expertise. By seamlessly connecting physical sketches with real-time simulations, PAINT2SIM strives to function as a digital twin for 2D fluid flow simulations.

Our contributions include the integration of the Lattice Boltzmann-based library OPENLB version 1.5 [61] into the mobile device, *just-in-time* simulation and visualization, as well as stable simulations for most cases. The application PAINT2SIM plays a pivotal role in advancing applied CFD by providing students and engineers with a user-friendly platform for quick insights into 2D fluid dynamics. The unique feature of generating *just-in-time* simulations on mobile devices empowers users to swiftly visualize and analyze fluid behavior in real-time, enhancing the accessibility and efficiency of fluid flow studies.

This technology has the potential to revolutionize how we teach and learn fluid dynamics, as well as how we design and optimize fluid-based systems. PAINT2SIM has the potential to benefit a wide range of users and applications, from students learning the fundamentals of fluid dynamics to engineers designing complex systems in the chemical, aerospace, and automotive industries. The technology can also be applied in medical research and environmental studies, where fluid behavior plays a crucial role. By simplifying the simulation process and making it more accessible, PAINT2SIM has the potential to democratize the field of CFD and encourage a wider range of users to explore the fascinating world of fluid dynamics.

In the remainder of this paper, we delve into the method employed by PAINT2SIM, present the numerical results obtained through its implementation, and engage in a comprehensive discussion of these results. A user guide with a download link for PAINT2SIM can be found in Appendix C.

4.2 Method

There are three critical requirements that must be fulfilled in order to run scanned hand-drawn simulation domains and simulate them locally on mobile devices: a high performance to simulate and visualize the simulation *just in time*, a high stability due to the various domains that can be scanned and the adjustable Reynolds number, and the physical accuracy should have quantitatively minimal errors and should be qualitatively comparable to reality.

In the following sections, we discuss the LBM, the simulation model used in this study, and its suitability for a high performance and physical accuracy. Following that, we describe the methods utilized to ensure the stability of the simulation.

4.2.1 Lattice Boltzmann Method (LBM)

The LBM offers several advantages when it comes to both performance and physical accuracy in simulating fluid behavior. One advantage is that the LBM can be easily parallelized, allowing for faster computation times and higher performance. Another advantage is that the LBM inherently models the fluid at a mesoscopic level, allowing for an accurate representation of complex physical phenomena such as turbulence and multiphase flows. This makes the LBM well-suited for simulating a wide range of fluid dynamics problems. In the remainder of this section, we provide a brief introduction to the LBM and the target equations, the Navier–Stokes Equations (NSE) for mass and momentum conservation in fluid dynamics.

The NSE is the fundamental equation that governs the behavior of fluids, and it is widely used in CFD simulations. The NSE in full form can be solved only numerically using various discretization methods, such as the finite difference, finite volume method or the LBM.

The NSE can be written as follows:

$$\nabla \cdot \boldsymbol{u} = 0, \tag{4.1}$$

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} = -\frac{\nabla p}{\rho} + \nu_m \nabla^2 \boldsymbol{u} + \frac{\boldsymbol{F}}{\rho}, \tag{4.2}$$

where $\boldsymbol{u}$ is the velocity vector, p is the pressure, ρ is the fluid density, ν_m is the molecular kinematic viscosity of the fluid, and $\boldsymbol{F}$ is the external force acting on the fluid.

The LBM approximates the conservation equations in its limit (Chapman–Enskog expansion) on a discrete grid of points connected by a set of links that represent the paths along which the fluid particles can move [62]. In the LBM, the spatial and temporal states of these particles are represented by the probability distribution functions (PDFs) that evolve according to the lattice Boltzmann equation.

$$f_i(\boldsymbol{x} + \boldsymbol{e}_i \Delta t, t + \Delta t) - f_i(\boldsymbol{x}, t) = \Omega^C + \Omega^F, \tag{4.3}$$

where f_i is the PDF at lattice node i and time t, $\mathbf{e}_i$ is the normalized discrete velocity in the i-th direction, $\Omega^C = -(f_i - f_i^{eq})/\tau$ is the collision operator, and Ω^F is the Guo forcing term [63].

In the current simulations, the D2Q9 lattice is used, where D is the number of dimensions and Q is the number of the normalized discrete velocity directions. The corresponding lattice cell is shown in Figure 4.1.

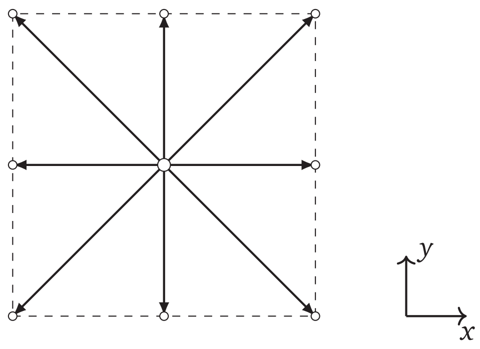


Figure 4.1: *A schematic illustration of the discrete velocity set for the D2Q9 lattice*

4.2.2 Smagorinsky BGK Collision Model

The Smagorinsky model [64] is a subgrid-scale model used in a large eddy simulation (LES) of turbulent flows. The model introduces a turbulent viscosity term to the governing equations of fluid flow, which is based on the strain rate tensor of the flow field. The model filters out the small unresolved vortices by replacing them with an artificial viscosity increase. The large eddies are preserved. The modified strain rate tensor describes the production of turbulent kinetic energy in the flow.

The turbulent viscosity term is given by the following:

$$\nu_t = (C_S \triangle x)^2 |\mathbf{S}|, \tag{4.4}$$

where ν_t is the turbulent eddy viscosity, $\triangle x$ is the grid spacing, C_S is the Smagorinsky constant, and $|\mathbf{S}|$ is the magnitude of the strain rate tensor. The Smagorinsky constant is the filtering parameter that determines which eddies are neglected.

The modified momentum equation with the addition of the turbulent viscosity term becomes the following:

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} = -\frac{1}{\rho} \nabla p + \nu_{eff} \nabla^2 \boldsymbol{u} + \frac{\boldsymbol{F}}{\rho}, \tag{4.5}$$

$$\nu_{eff} = \nu_m + \nu_t. \tag{4.6}$$

For the incompressible NSE with a Smagorinsky LES approach, the lattice Boltzmann equation using the BGK collision operator [65] can be rewritten as follows:

$$f_i(\boldsymbol{x} + \boldsymbol{e}_i \Delta t, t + \Delta t) - f_i(\boldsymbol{x}, t) = -\frac{\triangle t}{\tau_{eff}(x,t)} (f_i - f_i^{eq}) + \Omega^F, \tag{4.7}$$

where $\tau_{eff}(x,t) = \frac{\nu_{eff}(x,t)}{c_s^2} \frac{\Delta t}{\Delta x^2} + \frac{1}{2}$ is the effective relaxation time adapted to the Smagorinsky model. Here, c_s is the discrete speed of sound.

Due to obstacles in the path of a fluid, flow instabilities can be induced even at low Reynolds numbers. Therefore, it is necessary to adjust the relaxation time accordingly. The Smagorinsky BGK model accomplishes this by automatically increasing the relaxation time at the cells with a high shear rate. In the case of a laminar flow, the turbulent viscosity $\nu_t \approx 0$. Choosing a correct Smagorinsky constant secures a stable run of the simulation.

4.2.3 Fringe Region Technique

A fringe region technique [66] is used to eliminate instabilities at the outflow boundary condition. The outlet can become divergent if a large eddy flows through it. In order to compensate for this, a fringe zone is applied to laminarize the outflow. To achieve this, the NSE in the near-to-outlet region is forced with a special term.

$$\boldsymbol{F} = \lambda(x) \cdot (\boldsymbol{U} - \boldsymbol{u}), \tag{4.8}$$

where $\boldsymbol{U}$ is the prescribed velocity, $\lambda(x)$ is the fringe function that varies smoothly from 0 to 1 over a distance of a few grid points, and $\boldsymbol{u}$ is the computed velocity.

In the fringe region technique, the prescribed velocity $\boldsymbol{U}$ is obtained using a mixing length model. The mixing length model can be written as follows:

$$\boldsymbol{U}(x) = \boldsymbol{U}(x) + [\boldsymbol{U}(x_{out}) - \boldsymbol{U}(x)]S\left(\frac{x - x_{mix}}{\Delta_{\mathrm{mix}}}\right), \tag{4.9}$$

where $\mathbf{U}(x)$ is the velocity at a point x, x_{out} is the outlet coordinate, x_{mix} and Δ_{mix} are tuning parameters for transition between real and prescribed velocities, and S is a smooth function that varies from 0 to 1 over a distance.

4.2.4 Concept and Realization

The implementation of the system used two separate shared libraries: one for OpenCV and one for OpenLB. OpenCV provided the image processing capabilities, while OpenLB provided the CFD simulation capabilities. The shared libraries were written in C++ and can be compiled on a variety of platforms. The implementation involved creating a set of functions that could be used by both the OpenCV and OpenLB libraries. These functions were used to perform image processing and CFD simulation, respectively. In addition, Unity was used to create the application for the mobile device. Both OpenCV and OpenLB communicate independently with Unity through their respective shared libraries.

4.2.4.1 Concept

In order to achieve just-in-time simulation and visualization, there must be a clear separation between the frontend, which is the application itself, and the backend, which consists of the OpenLB and OpenCV shared libraries. Each of the shared libraries is called in separate threads which allows for the decoupling of the simulation and visualization thereof. The communication between the frontend and backend consists primarily of the exchange of the simulation results for a timestep in the form of a pressure or velocity array as shown in Figure 4.2.

4.2.4.2 Structure of the OpenLB Shared Library

In this section, we present an overview of the OpenLB shared library structure that we employed for performing the simulations on smartphones. Specifically,

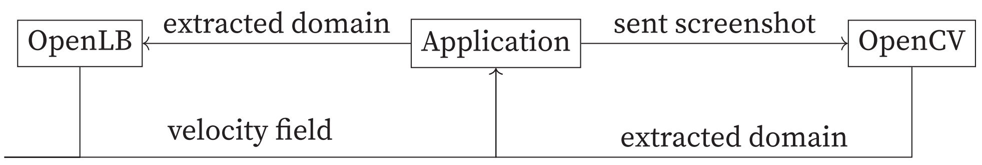


Figure 4.2: *Communication between the application,* OpenCV*, and* OpenLB

Algorithm 1 illustrates the main loop of the library, which adheres to the standard Lattice Boltzmann simulation structure with OpenLB. Initially, OpenLB is instantiated, and crucial classes are initialized. The unit converter, which stores the lattice relevant data, is then declared. Next, the simulation domain is instantiated, and its dimensions correspond to the domain scanned with the smartphone. This domain is then passed to the load balancer, which distributes the cuboid into subsections necessary for parallel computing. The material number map is an array of numbers that refer to the materials in the simulation. However, it cannot be utilized as is and necessitates a transfer to the OpenLB-specific class, `superGeometry`. After preparing the geometry, the lattice is ready for simulation, and boundary conditions can be set. In particular, we apply the Smagorinksy BGK model in Section 4.2.2 to the material numbers of the fluid outflow and inflow. Additionally, we utilize the fringe region technique around the outflow. Moreover, we need to specify which material numbers define the inlet and outlet. We also add the postprocessor, which receives relevant dimensions and pointers to the result arrays. At step 10 of Algorithm 1, the simulation commences with the **for**-loop. In this loop, T corresponds to the total simulation time, while iT represents the current timestep. For each timestep, we update and set the boundary values. Subsequently, we call the collide and stream functions to retrieve the values for the subsequent step. Finally, we synchronize the number of timesteps saved per second with the frames per second (t_{fps}) of the smartphone application (step 13 to 21). This synchronization ensures better performance as we do not save every timestep of the simulation but still maintain a fluid visualization for the user.

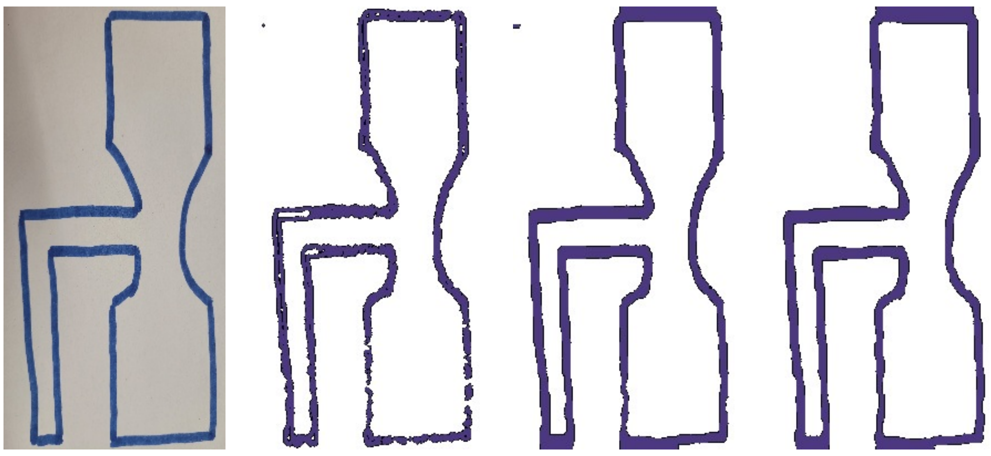
4.2.4.3 Structure of the OpenCV Shared Library

The OpenCV shared library plays a critical role in enabling OpenVisFlow to extract contours, which is a vital step in obtaining the simulation domain from an image. Contour extraction involves identifying the object's boundary in an image and approximating it with a curve. The curve consists of continuous

Algorithm 1 Mainloop of the OpenLB Shared Library

```
1: init OpenLB
2: declare unit converter
3: instantiation of the simulation domain
4: instantiation of a load balancer
5: preparing of the geometry
6: preparing of the lattice
7: add postprocessor
8: calculate the number of timesteps from the total simulation time $T$
9: start timer $t_{fps}$
10: for $iT = 0; iT \leq T; i++$ do
11:     set boundary values
12:     collide and stream
13:     if $t_{fps} \leq 1$ then
14:         if $\Delta t \leq 1/fps$ & $count \leq fps$ then
15:             write results via postprocessor
16:             write Mega Lattice Updates per Second
17:         end if
18:         count++
19:     end if
20:     reset $t_{fps}$
21:     count = 0;
22:     if endSimulation then break;
23:     end if
24: end for
```

points along the boundary with the same color or intensity. The process of domain extraction begins by resizing the image to the desired simulation resolution, followed by applying a threshold to enhance the contrast between the domain and the background. The `findContours` function is then utilized to extract the domain boundaries. Finally, morphological functions are applied to postprocessing of the extracted domain. Specifically, a morphological close function is used to fill potential holes in the boundary, while a morphological open function is used to eliminate noise from the domain. Figure 4.3 displays the results of the processing steps, where Figure 4.3 (a) depicts the photo of the hand-drawn domain to be extracted, and Figure 4.3 (b) presents the outcome of the initial extraction with a threshold; Figure 4.3 (c) - (d) represent the post-image processing steps.



(a) *Photo of the drawn domain.* **(b)** *Extracted contour.* **(c)** *Morphological close.* **(d)** *Morphological open.*

Figure 4.3: *The figure illustrates the domain extraction process, starting with the original image (**a**), followed by the resulting image after the initial extraction with a threshold (**b**). Post-image processing steps are then applied, leading to the final processed images shown in (**c**, **d**).*

4.2.5 Expanding OpenVisFlow for Mobile Fluid Flow Simulation with OpenLB

The OpenVisFlow library, based on Unity, is designed to be easily expandable, allowing it to handle and visualize various data types. To achieve this, two new classes are required that inherit from the parent `DataManager` class. These classes enable communication between OPENVISFLOW and the shared libraries OPENLB and OPENCV. Figure 4.4 depicts the class diagram for the new classes, `OpenCVDatamanger` and `OpenLBDatamanger`, which inherit from the parent class `Datamanager`.

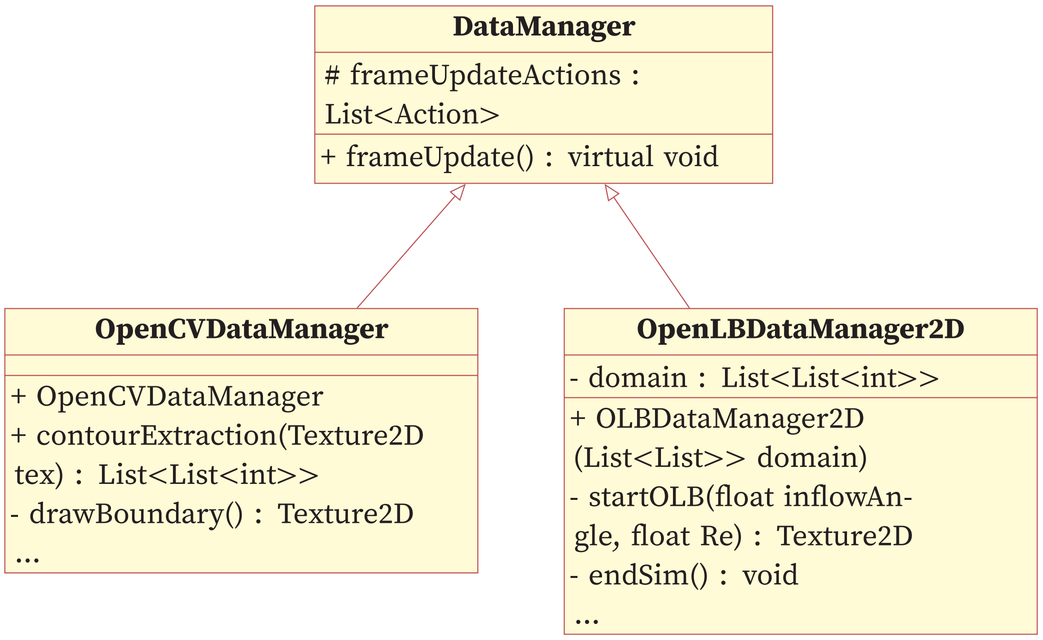

DataManager
# frameUpdateActions : List<Action>
+ frameUpdate() : virtual void
OpenCVDataManager
+ OpenCVDataManager
+ contourExtraction(Texture2D tex) : List<List<int>>
- drawBoundary() : Texture2D
...
OpenLBDataManager2D
- domain : List<List<int>>
+ OLBDataManager2D (List<List>> domain)
- startOLB(float inflowAngle, float Re) : Texture2D
- endSim() : void
...

Figure 4.4: *Class diagram for the* OPENCV *and* OPENLB *DataManager, highlighting the most essential functions.*

The `OpenCVDataManager` class adds necessary functions for communication between OPENVISFLOW and the OPENCV shared library. Similarly, the `OpenLBDataManager2D` class inherits from `DataManager` and includes the essential functions required to initiate and terminate a simulation. The class also contains additional functionalities such as an optional placement of a fringe zone. Overall, these new classes ensure seamless communication between OPENVISFLOW and the OPENLB and OPENCV shared libraries, allowing for efficient data handling and visualization.

Visualization of the Simulation Data

In order to visualize the calculated results of the OPENLB shared library introduced in Section 4.2.4.2, a new visualizer class has to be implemented which has the ability to handle 2D arrays of the type float, transform this information to a texture, and display it. The class diagram of the new class is depicted in Figure 4.5. Following the OpenVisFlow framework, the visualizer is initialized with a `Colorscheme`, which consist of different colors that are used to visualize the flow and an instance of the `OLBDataManager2D` in order to get the simulation data. The function `timeStepUpdate` is added to the inherited action list. This function is called on every frame and extracts the minimum and maximum bounds for the macroscopic moments of the current timestep. Based on that, the color scheme can be mapped to each cell and rendered to a texture for display.

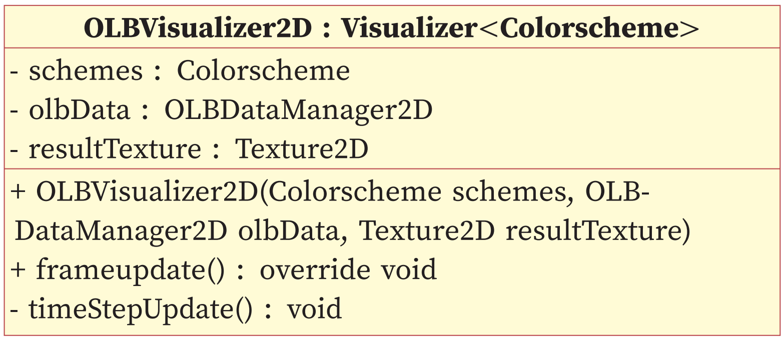

OLBVisualizer2D : Visualizer<Colorscheme>
- schemes : Colorscheme
- olbData : OLBDataManager2D
- resultTexture : Texture2D
+ OLBVisualizer2D(Colorscheme schemes, OLBDataManager2D olbData, Texture2D resultTexture)
+ frameupdate() : override void
- timeStepUpdate() : void

Figure 4.5: *Class diagram of the OLBVisualizer2D, highlighting the most essential functions.*

4.3 Numerical Experiments and Discussion of Results

This section evaluates the precision and performance of the LBM as implemented in PAINT2SIM. We categorize the results into qualitative and quantitative aspects. The qualitative section focuses on the visualization of the simulation results and their correspondence to physical reality. For the quantitative part, we assess the performance and physical accuracy of the simulation.

4.3.1 Test Case Setup

To validate the capabilities of PAINT2SIM, we chose the 2D cylinder test case provided by the OpenLB library. This test case replicates the configuration detailed by Schäfer et al. [67]. This selection enables a direct comparison between the results generated by PAINT2SIM and the established outcomes in the relevant domain. In addition, we recreated the same scenario through manual drawing, ensuring consistent proportions as outlined in the reference work [67].

Subsequently, the hand-drawn representation was scanned using PAINT2SIM, enabling a precise evaluation of its accuracy in replicating the expected flow patterns and attributes. The geometry used for validation is visually presented in Figure 4.6.

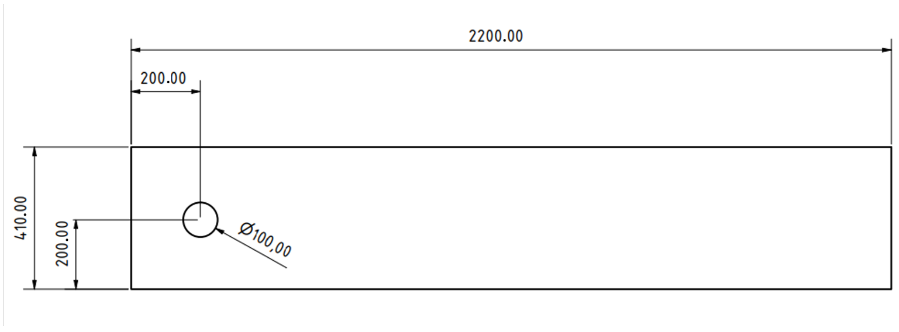


Figure 4.6: *Geometry employed for validation, with dimensions in SI millimeters.*

It is important to note that, in this study, PAINT2SIM incorporates the Smagorinsky BGK Model and the Fringe Region Technique to maintain simulation stability even at lower resolutions. Notably, the validated cylinder2D case does not utilize either of these techniques.

4.3.2 Choice of Discretization Parameters

To accommodate various performance capabilities on mobile devices, PAINT2SIM offers four resolution options in terms of Δx, representing the voxel length. In alignment with this, we conducted corresponding simulations using the OPENLB framework. The timestep ΔT scales diffusively with the resolution, meaning that it decreases or increases quadratically in correspondence with Δx. To demonstrate that OPENLB produces consistent results with those presented in Schäfer et al. [67], we also incorporated a higher resolution. Due to the performance restriction inherent in mobile devices, this higher resolution cannot be executed on mobile devices. PAINT2SIM differs from the validation case in three aspects. Firstly, it employs the Smagorinsky BGK Collision Model. Secondly, it incorporates the fringe region technique to ensure stability. Thirdly, the application exclusively employs the Bounceback boundary condition instead of the Bouzidi second-order condition. The decision between the first and second orders is rooted in numerical analysis and depends on the nature of the boundary—being first order for curved boundaries and second order for axis-aligned cells. The prevalence of a first-order condition in most scenarios is attributed to the staircase approximation. This approach is preferred due to the challenging extraction of the real geometry surface required by Bouzidi from an already discretized scanned domain.

paint2sim-1: The fringe region technique as well as the Smagorinsky BGK Collision Model are used with the Smagorisnky Constant $C_s = 0.15$.

paint2sim-2: The fringe region technique is used while the Smagorinsky BGK Model is replaced with the BGK Collison Model.

paint2sim-3: The fringe region technique is removed, and the Smagorinsky BGK Model is replaced with the BGK Collison Model.

paint2sim-4: The fringe region technique is removed, and the Smagorinsky BGK Model is used.

In order to perform a comparison between Bouzidi and Bounceback, we also ran the validation case from OpenLB with Bouzidi ***OpenLB-1*** and with Bounceback ***OpenLB-2***.

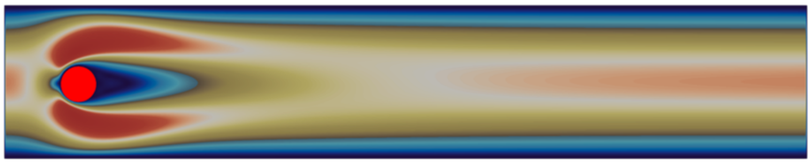
4.3.3 Validation

4.3.3.1 Qualitative Results

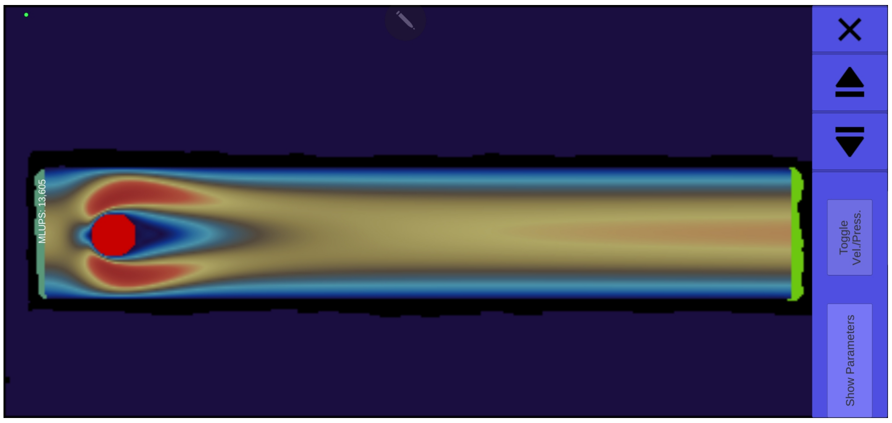
Figure 4.7 presents a side-by-side comparison of the results. Observing the laminar flow at $Re = 20$, there is no noticeable difference between the validated case shown in Figure 4.7 (a) and the result obtained using PAINT2SIM, as shown in Figure 4.7 (b). Furthermore, when comparing the unstable flow in Figure 4.7 (c) - (d), the qualitative results are also in agreement.

4.3.3.2 Quantitative Results

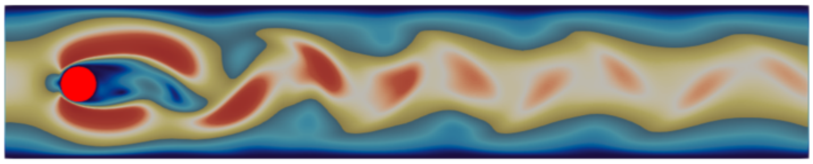
This section presents a comparative assessment of the validation results obtained from [67], as depicted in Table 4.1, concerning the drag and lift coefficients on the cylinder, with both OpenLB and PAINT2SIM. Table 4.2 presents the results of simulations conducted at $Re = 20$, detailing the parameters and resulting drag and lift coefficients for specific cases. The results from OpenLB align within the predefined bounds set in Table 4.1 when utilizing a sufficiently high resolution. Conversely, PAINT2SIM at its maximum resolution yields results that deviate by approximately 11% for the drag coefficient. Additionally, the lift coefficient exhibits considerable variation, failing to closely match the specified margin. This discrepancy primarily stems from errors introduced during the hand-drawn domain scan, where image processing techniques for domain extraction introduce slight shape differences, particularly in the representation of



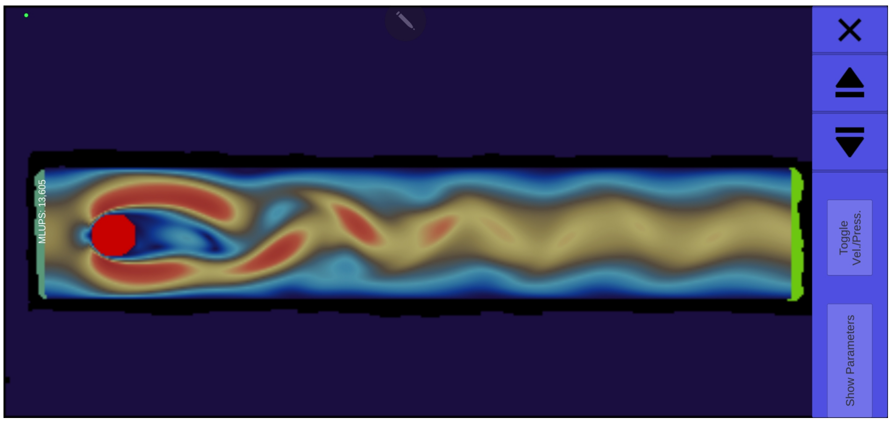
(a) *Example of the 2D cylinder from* OPENLB *with Re* = 20.



(b) *Simulation of a hand-drawn domain using* PAINT2SIM *with Re* = 20*, replicating the example used for validation.*



(c) *Example of the 2D cylinder from* OPENLB *with Re* = 100.



(d) *Simulation of a hand-drawn domain using* PAINT2SIM *with Re* = 100*, replicating the example used for validation.*

Figure 4.7: *Comparison of qualitative results for the flow around a cylinder between the OpenLB simulation cases validated by Schäfer et al. [67] and the paint2sim simulations.*

the cylinder within the domain. Given the current computational limitations of the mobile devices used for the scan and corresponding simulation resolution, addressing this issue comprehensively is presently impractical. Nevertheless, the results emphasize that an increase in resolution contributes to a reduction in the margin of error. In Table 4.3, the outcomes for flow simulations at $Re = 100$

Table 4.1: *Results of drag and lift coefficients from Schäfer et al. [67] in a laminar flow around a cylinder with Re = 20 for the stable case and Re = 100 for the unstable case.*

Reynolds Number (Re)	20	100
Characteristic Length [m]	0.100	0.100
Voxel Length [m]	-	-
Drag Coefficient	5.570–5.590	3.220–3.240
Lift Coefficient	0.010–0.011	0.990–1.010

Table 4.2: *Comparative analysis of drag and lift coefficients:* OpenLB *vs.* paint2sim *in the flow around a cylinder at* $Re = 20$ *(stable case).*

$\Delta x [m]$	OpenLB-1		OpenLB-2		paint2sim-1		paint2sim-2		paint2sim-3		paint2sim-4	
	Drag	**Lift**	**Drag**	**Lift**	**Drag**	**Lift**	**Drag**	**Lift**	**Drag**	**Lift**	**Drag**	**Lift**
0.010	5.820	0.015	6.116	0.016	4.040	0.046	4.015	0.046	4.015	0.046	4.040	0.046
0.006	5.689	0.008	5.785	0.008	4.915	0.058	4.899	0.058	4.900	0.058	4.915	0.058
0.005	5.632	0.012	5.796	0.012	5.282	0.064	5.268	0.064	5.269	0.064	5.282	0.064
0.004	5.624	0.009	5.740	0.009	5.882	0.074	5.871	0.074	5.871	0.074	5.882	0.074
0.003	5.601	0.010	5.505	0.010	-	-	-	-	-	-	-	-
0.002	5.593	0.010	5.628	0.024	-	-	-	-	-	-	-	-

exhibit similar challenges to those encountered in the stable scenario at $Re = 20$.

Table 4.3: *Comparative analysis of drag and lift coefficients:* OpenLB *vs.* paint2sim *in the flow around a cylinder at* $Re = 100$ *(unstable case).*

$\Delta x [m]$	OpenLB-1		OpenLB-2		paint2sim-1		paint2sim-2		paint2sim-3		paint2sim-4	
	Drag	**Lift**	**Drag**	**Lift**	**Drag**	**Lift**	**Drag**	**Lift**	**Drag**	**Lift**	**Drag**	**Lift**
0.010	-	-	-	-	3.954	2.801	4.061	3.027	4.056	2.974	3.966	2.809
0.006	3.696	1.326	3.988	1.654	3.685	1.786	3.684	1.866	3.689	1.865	3.699	1.785
0.005	3.485	1.157	3.777	1.449	3.369	1.041	3.380	1.243	3.484	1.368	3.367	0.979
0.004	3.353	1.089	3.556	1.261	3.414	0.872	3.390	0.839	3.389	0.885	3.418	0.864
0.003	3.324	1.091	3.361	1.110	-	-	-	-	-	-	-	-
0.002	3.264	0.991	3.273	1.013	-	-	-	-	-	-	-	-

4.3.4 Performance

Figure 4.8 presents the total performance achieved by PAINT2SIM measured in millions of cell updates per second (*Mega Lattice Updates per Second (MLUPS)*) across a set of mobile- and stationary test devices.

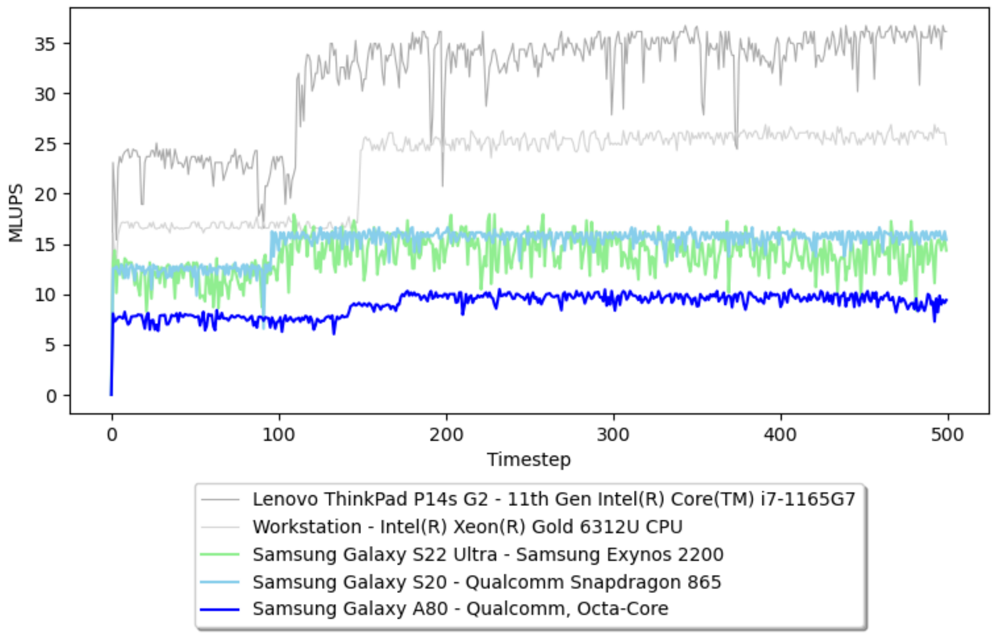


Figure 4.8: *Performance comparison of* PAINT2SIM *across different devices. Performance is measured in Mega Lattice Updates per Second (MLUPS).*

A central performance bottleneck for numerical simulations on mobile devices due to their inherent compute-heavy nature is given by heat management constraints. This is the primary explanation for visible performance fluctuations as mobile devices tend to aggressively reduce their power output beyond short performance bursts in order to prevent overheating.

While the underlying LBM library OPENLB supports various parallelization modes both on CPU and GPU targets [61, 68], PAINT2SIM explicitly only uses single-threaded, single-precision, non-vectorized execution for maximum device portability and as a heat-management trade-off. While OpenMP-based shared memory parallelization was possible, core heterogeneity and thread binding caused issues across the diverse set of test devices. Single-threaded execution provides sufficient cross-device performance for the intended two-dimensional flow simulations.

On the lowest end, we conducted a performance comparison between PAINT2SIM and the results presented in [53]. Due to hardware availability issues, we were unable to use the same experimental setup and instead relied on a low-end Huawei P8 Lite with inferior specifications compared to the initial high-end NVIDIA Shield K1 Tablet. Table 4.4 presents a comparison of the specifications obtained via Geekbench, a well-established mobile benchmark suite, and the achieved total performance in MLUPS. Despite the decision to not utilize parallelization, PAINT2SIM's performance compares favourably at a speed-up of approximately 1.45.

Among the tested mobile devices in Figure 4.8, the Samsung Galaxy A80 offers the lowest CPU performance. Despite this, the visualization remains smooth for users at an average throughput of 10 MLUPS and no noticeable lags. At the top end, the single-threaded LBM performance on a Samsung Galaxy S22 Ultra is quite close to the unvectorized performance on higher-end x86 CPU cores at approximately 24 MLUPs.

Overall, the performance characteristics exhibited by PAINT2SIM are sufficient for the intended simulation cases and are highly competitive to single cores of full-powered x86 CPUs, considering the comparably smaller power envelope.

A notable issue associated with mobile devices is their tendency to decrease the power output of their CPUs in order to prevent overheating. Figure 4.9 presents the average MLPUs of the mobile device *Samsung Galaxy S22 Ultra* during extended simulation periods. The graph illustrates a significant decline in performance over time. Moreover, the decline is not continuous; rather, the device maintains its performance until the temperature reaches a critical point, at which it then ramps down to a lower performance level.

Table 4.4: *Performance and spec comparison of Nvidia Shield Tablet K1 and Huawei P8 Lite. For the spec comparison and Single-Core Score Geekbench [69].*

Aspect	Nvidia Shield Tablet K1	Huawei P8 Lite
Processor	ARM tn8	ARM ARMv8
Base Frequenzy	**2.22** GHz	1.71 GHz
Single-Core Score	**207**	166
MLUPS	1.1	**1.6**

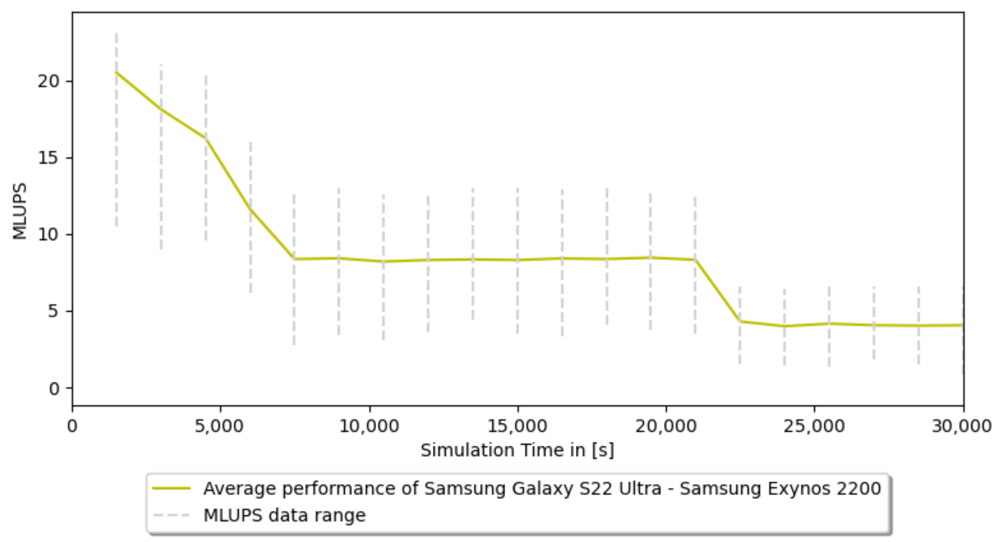


Figure 4.9: *Performance decline over time of* PAINT2SIM. *Performance is measured in Mega Lattice Updates per Second (MLUPS).*

4.3.5 Conclusions

In this paper, we present the PAINT2SIM software for LBM simulations on mobile devices and the numerical validation of the software on an example of the Schäfer test case. The simulation results are categorized into qualitative and quantitative aspects, focusing on visualization, performance, and physical accuracy. Regarding performance, we analyzed the total performance achieved by PAINT2SIM on various mobile devices. The Samsung Galaxy A80 exhibited the lowest CPU performance among the tested devices but still provided smooth visualization with an average of 10 Mega Lattice Updates per Second (MLUPS) and no noticeable lags. On the other hand, the Samsung Galaxy S22 Ultra demonstrated impressive performance, comparable to higher-end x86 CPUs, achieving the highest MLUPS of 24 under specific conditions. However, fluctuations in performance were observed due to heat management constraints, leading to an aggressive reduction in the CPU's power output beyond short performance bursts. The validation of PAINT2SIM involved qualitative and quantitative comparisons. For the qualitative validation, we replicated a well-established 2D cylinder example using a hand-drawn image and compared the results with the validated case from OPENLB. The flow patterns and characteristics exhibited by PAINT2SIM were found to be in agreement with the established findings. For the quantitative validation, we compared the drag and lift coefficients of the cylinder simulation with the results from previous studies: while the drag coefficients showed minor discrepancies of around ∼10%, the lift coefficients obtained from PAINT2SIM differed greatly. These variations can be attributed to the shape approximation of the hand-drawn cylinder and slight inaccuracies in the scanned image. Overall, the performance and accuracy of PAINT2SIM on mobile devices proved to be sufficient for the intended simulation cases and to be highly competitive with full-powered x86 CPUs within a smaller power envelope. Further enhancements can be achieved through a parallelization of the application on mobile devices, which would significantly improve performance metrics. Future work should also address the issue of performance fluctuations resulting from heat management constraints. Taking these factors into account, the digital twin and virtual laboratory features of PAINT2SIM hold the promise of offering a valuable simulation tool for 2D computational fluid dynamics applications, allowing for on-the-go simulations.

5 CFD-Based Digital Twin for Urban Environments

This chapter is based on the published article

Teutscher, D. et al. "A digital urban twin enabling interactive pollution predictions and enhanced planning." In: Building and Environment *281 (2025), p. 113093.* ISSN*: 0360-1323.* DOI*: https://doi.org/10.1016/j.buildenv.2025.113093*

The abstract has been omitted, and formatting has been adjusted to fit the thesis style. The main content remains unchanged.

5.1 Introduction

With the emergence of Industry 4.0 and the impending Industry 5.0, the term digital twin (DT) has become a cornerstone in discussions about future technological advancements. The concept of DT was first introduced by Grieves *et al.* in 2002 [18]. DTs have been successfully applied across a wide range of engineering domains, with each application offering unique advantages and presenting specific challenges. In manufacturing, DTs replicate real-time production systems, enabling continuous monitoring and optimization of processes, and improving product quality and efficiency [19, 27]. However, the main challenge in this domain is the high implementation cost and complexity in integrating DTs with existing production systems. In aerospace, DTs are used for predictive maintenance and to monitor aircraft conditions in real time, improving safety by forecasting potential failures [23]. While this application provides significant benefits in terms of increased reliability and safety, it can be limited by the availability of accurate data and the computational burden of simulating complex flight conditions. Similarly, in the automotive industry, DTs are employed for monitoring vehicle conditions and optimizing performance, with some studies focusing on autonomous driving [26].However, challenges in this area include data privacy issues and the need for high-quality sensor data, especially in dynamic traffic environments. In civil engineering, DTs play a vital role in monitoring structural health, such as in the case of bridges, buildings, and dams [70–73]. These applications enable real-time assessments of structural stability and the prediction of failures before they occur. However, the challenges here include the need for high-precision sensors and models to ensure accurate predictions, as well as the cost of deploying DT systems at large scales. In construction, DTs are used for improving project visualization, tracking construction progress, and enhancing life cycle management [74]. While these applications improve planning and decision-making, they often require significant resources and expertise to create and maintain the DT. Additionally, the integration of various software tools and databases can lead to compatibility issues. When it comes to DTs for urban environments, external factors such as air pollution and high wind speeds in narrow street canyons must be carefully considered. Recent advances have been made in developing DTs for cities, which aim to enhance urban planning and environmental management. Schrotter *et al.* [75] presented a DT for urban planning that monitors environmental factors such as noise, temperature, and pollution. However, in their work, they did not rely on computational fluid dynamics (CFD) with current environmental data to predict the distribution of these factors. Instead, they used historical data to provide a general overview of environmental distributions. For more accurate predictions, a model capable of integrating real-time data from measurement stations is necessary. In this regard, various models have been developed to simulate wind and pollution distribution, even if they are not directly updateable with real-time data. Pasquier *et al.* [76] employed the lattice Boltzmann method (LBM) with the open-source software *OpenLB* [7, 14, 61] to simulate traffic emission distribution in a city and validated the model. Similarly, Van Hooff *et al.* [77] simulated airflow around urban structures, while Jeanjean *et al.* [78] examined the effects of trees on pollution dispersion in cities, using a wind tunnel for validation. Taleb *et al.* [79] studied how trees act as windshields, reducing dust particle concentrations in desert cities.

The relevance of this research lies in the adverse health effects caused by air pollution, e.g., through exposure to particulate matter [80] and gaseous substances such as nitrogen dioxide [81], which impact respiratory and cardiovascular health. Monitoring the pollution distribution (e.g., by government-official monitoring sites) and integrating this data into urban planning is crucial for public health. Within the revision of the EU Ambient Air Quality Directive, monitoring of non-regulated "new pollutants" such as ultra fine particles or black carbon is becoming mandatory. Depending on the size and the number of inhabitants of the member state, there is an obligation to install a number of "supersites" that monitor these pollutants. However, ultra fine particles and other pollutant concentrations are known to fluctuate and local exposure depends strongly on the transmission conditions as well as the spatial and temporal behavior of pollutant sources [82]. The challenges arise from the complexity of identifying the sources of pollution and their impact on local ambient concentration levels [83]. For example, Dröge *et al.* [84] showed the significance of pollutant transmission for ultra fine particle concentrations from air traffic and the high temporal variations. Other prominent urban sources of (ultra)fine particles and air pollution are traffic [85] and public transport [86]. Furthermore, heating from wood stoves and other combustion processes, such as barbecues, can cause a significant increase in particle concentration and gaseous pollutants, especially in densely populated (residential) areas [87]. Spatially resolved measurements to investigate local particle concentrations are associated with a high amount of measurement effort [88] or limited accuracy considering the use of distributed low-cost sensors [89]. Therefore, simulations are an excellent tool to predict pollutant transmission and determine local concentration levels in the context of limited numbers of monitoring sites. Although previous studies have leveraged DTs for urban applications, they have focused primarily on structural integrity or large-scale environmental factors such as noise. However, a DT specifically designed to model wind distribution and air pollution in urban areas, while integrating real-time meteorological data and enabling interactive geometry adjustments using OpenStreetMap (OSM) [90], remains largely unexplored. To address these gaps, this paper presents a novel DT that:

- Utilizes the homogenized lattice Boltzmann method (HLBM) for accurate and computationally efficient fluid simulations in urban environments,
- Integrates real-time meteorological data to dynamically update wind and air pollution distributions, enhancing the realism of simulations,

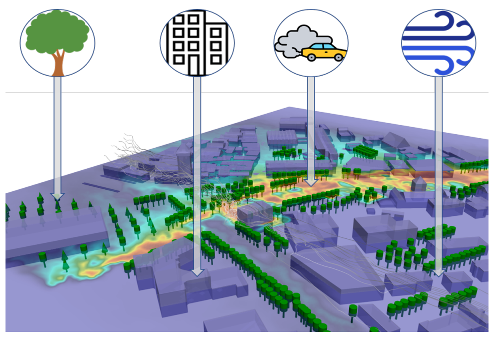
- Enables interactive urban geometry adjustments using OSM, allowing flexible adaptation to evolving city structures and planning scenarios.

By combining these elements, our DT framework extends beyond traditional static urban models, providing a dynamic and interactive tool for air quality monitoring and urban planning. The remainder of this paper is structured as follows: Section 5.2 outlines the methodology, including the DT concept, integration of OSM data, mathematical modeling, and discretization using HLBM. Section 5.3 presents the validation of the numerical approach using experimental data. Finally, Section 5.4 discusses the DT results, followed by the conclusions in Section 5.5.

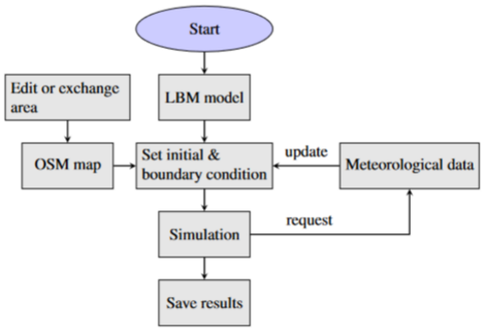
5.2 Methodology

5.2.1 Concept and Workflow of Digital Twin for Urban Areas

The proposed DT concept and workflow, illustrated in Figure 5.1, is centered on a simulation framework with two primary interfaces. The first interface enables automatic updates to the simulation using real-time data from measuring stations, while the second is a manual interface that allows users to adjust the geometry in the simulation, integrating and removing data from OSM into the simulation. Creating a DT for urban environments requires the consideration of multiple dynamic factors, especially when the objective is to assist urban planners and policymakers in making data-driven decisions. One critical application of this DT is to assess the distribution of pollutants, such as particulate matter and nitrogen dioxide, throughout the city. Identifying sources of pollution poses significant challenges, especially with particulate matter, which originates from various sources such as traffic, industrial emissions, and domestic heating. This complexity is further compounded by the limitations of current measurement infrastructure, as most urban measuring stations provide only localized point-based data. This concept is based on the idea of giving city planners and policymakers the ability to insert, for example, a new pollution source into the area and manipulate the distribution by inserting trees and buildings. To use it as a planning tool, it is significant that the performance is good enough to give reasonable, quick results. For that reason, we use the open-source software *OpenLB* which allows the use of GPUs and is highly scalable by using more GPUs.



(a)



(b)

Figure 5.1: *Concept and workflow of the urban DT. (a) Conceptual visualization of the urban DT, illustrating key data elements such as buildings and porous zones (Section 5.2.3.3), pollution sources (including vehicles and buildings) (Section 5.2.4), and meteorological factors like wind speed and direction, which dynamically update the numerical model (Section 5.2.5). (b) Workflow of the DT, highlighting continuous updates through real-time meteorological data.*

5.2.2 Interactive Data Input

To ensure that the DT accurately reflects the state of its physical counterpart, the urban area, there must be a constant exchange of data. For this purpose, we utilize publicly available measuring stations that provide real-time information on wind speed, wind direction, and the concentrations of particulate matter and nitrogen dioxide. This data is then used to update the DT at regular intervals, showcasing the current state of the monitored area. By averaging the results over a longer period, pollution hot spots are identified.

5.2.3 OpenStreeMap

In addition to data from measuring stations, information about the types of buildings and objects in the area is also crucial. For this, we use the open-source map OSM, which stores data in an XML-based format. Each object is marked with a specification of what kind of object it is, allowing us to differentiate between porous objects such as trees and solid structures. Since the data is XML-based, it is easy to add or remove information, making it suitable for planning urban areas and exploring different scenarios.

5.2.3.1 Parsing OSM Data

The first step involves reading and interpreting OSM XML files using the TinyXML2 [91] library. The nodes and ways, which are the fundamental elements of OSM, are parsed for relevant geographical data.

Nodes parsing Nodes represent specific points with latitude and longitude. These are extracted as key-value pairs using attributes like lat and lon. The nodes are stored in an unordered_map to allow a quick lookup by their unique identifiers (IDs). If a node represents a tree, additional tags such as leaf_type and height are parsed. Tags are analyzed using a function that checks the presence of natural tags like tree, wood, or scrub.

Ways parsing Ways are sequences of nodes that define streets, buildings, or other linear/area features. Attributes for specific way types are filtered. Buildings are identified using a predefined set of building-related tags (e.g. residential, commercial) and are represented as a list of their footprint nodes. Roads are parsed by their name and width when available. Trees and natural features are parsed similarly, focusing on attributes like height or species.

Figure 5.2 illustrates the parsing process for nodes and ways, highlighting the extraction of essential attributes and relationships between OSM elements.

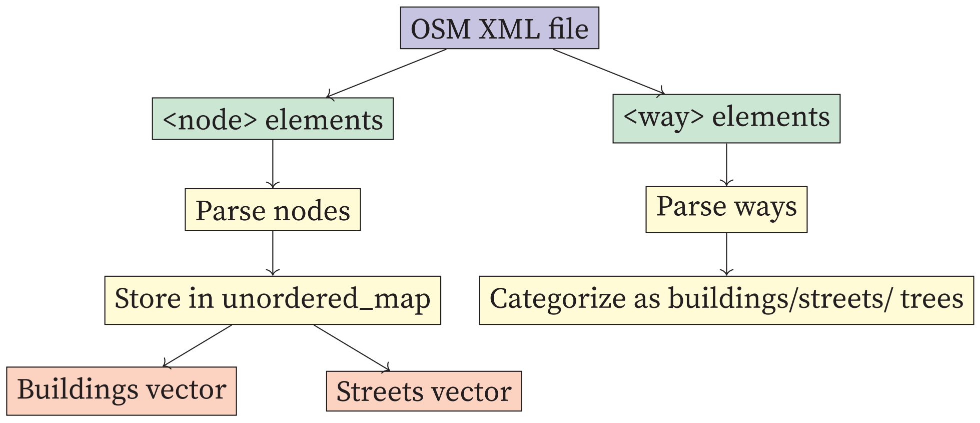


Figure 5.2: *Parsing workflow of OSM data, illustrating the extraction and categorization of nodes and ways into buildings and streets.*

5.2.3.2 Data Transformation and Filtering

Parsed data undergoes transformation to prepare it for 3D geometry creation or visualization.

Building Data Transformation Building footprints are transformed into local UTM (Universal Transverse Mercator) coordinates: The UTM zone is dynamically determined on the basis of the longitude of the first building node. Using the PROJ library, all geographical coordinates are converted into projected coordinates relative to the calculated UTM zone. To ensure compatibility with local systems, a translation is applied to shift coordinates into a local reference frame.

Tree and Natural Feature Processing Trees are processed similarly, with special attention to their height and species. If height is not explicitly defined, default values are assigned to avoid anomalies. Tree rows and areas (e.g., forest, scrubs) are treated as collections of nodes.

5.2.3.3 Geometry Creation

The processed data is used to create 3D geometries that represent the extracted OSM elements.

Buildings Building geometries are constructed as 3D prisms, with footprints that define the base and the height determined by the parsed or default values. Indicator polygons are created using the IndicatorPolygon3D class to encapsulate the footprint points. Figure 5.3 (a) shows the footprint of the sample building extruded into a 3D volume.

Streets Street geometries are expanded to 3D surfaces based on their width. A defined width or default value (e.g. 5 meters) is applied to calculate a polygonal representation of the street area, ensuring complete coverage beyond simple outlines.

Trees Tree geometries are created using cylindrical trunks and spherical crowns:

The trunk is represented by a cylinder with a fixed or parsed radius and height. The crown is visualized as a sphere placed at the top of the trunk.

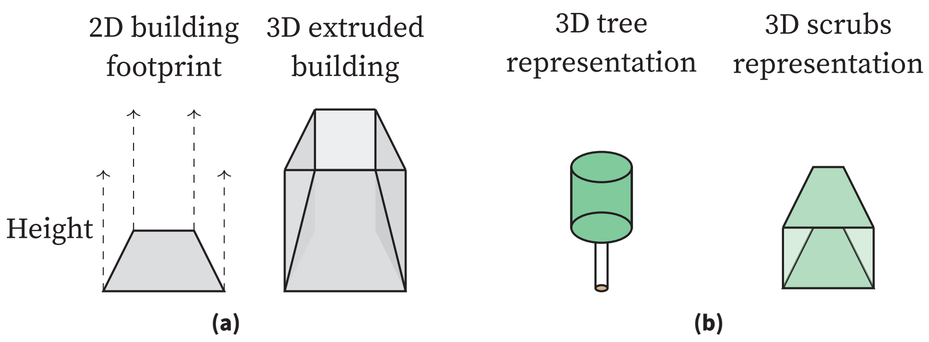


Figure 5.3: *(a) Illustration of the extrusion process, where a 2D building footprint is extended into a 3D volume based on building height, starting from the ground up. (b) 3D representations of a tree (left) and shrubs (right). The tree consists of a cylindrical trunk and crown, while shrubs are modeled as extruded volumes.*

Variations in leaf type and height are taken into account to enhance realism, as shown in Figure 5.3 (b).

Result of the Geometry Extraction The methods described above were applied to an OSM map within the bounding box defined by longitude (9.2063390–9.2118430) and latitude (48.4886460– 48.4918950). This process generated the geometry shown in Figure 5.5, which illustrates streets, buildings, and porous objects such as trees and shrubs colored green. This extracted geometry serves as the basis for the simulation setup used in the DT.

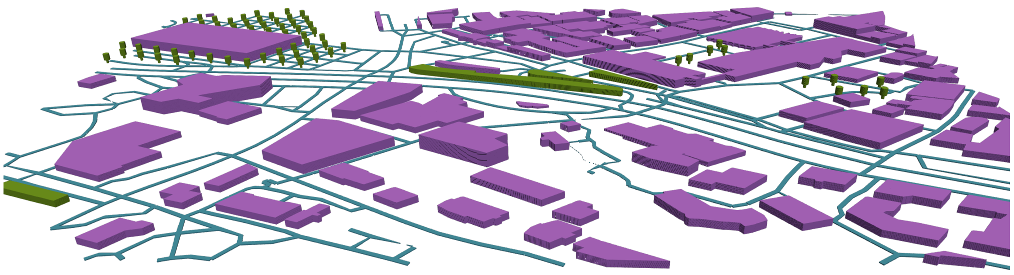


Figure 5.4: *Simulation geometry generated with OSM data. Depicted are buildings in purple, streets in gray and porous objects such as trees and scrub in green.*

5.2.4 Identification of Sources

To address the challenge of identifying sources of pollution, some assumptions must be made. Research indicates that most nitrogen dioxide pollution can be traced back to traffic emissions [92]. Although some of the pollution comes from factories, we assume that most is mainly from traffic. With this assumption, in case that no vehicle counters are present we can roughly estimate the number of cars on the street where the measurement station is located.

We first calculate the volume of the street by taking the length L, width W, and height H of the measuring station, which is approximately 5 meters above the ground. The volume V is given by

$$V = L \times W \times H. \tag{5.1}$$

By multiplying the measured concentration C of nitrogen dioxide with the volume V, we can calculate the total nitrogen dioxide $\mathrm{TNO_2}$ on the street:

$$\mathrm{TNO_2} = CV. \tag{5.2}$$

To estimate the number of cars N on the street, we divide the total nitrogen dioxide by the median emissions E of a car, which is $0.3\frac{\mathrm{g}}{\mathrm{km}}$:

$$N = \frac{\mathrm{TNO_2}}{E}. \tag{5.3}$$

In addition to traffic, many other sources can contribute to increased levels of air pollution. With the estimated number of cars, we can calculate the contribution of the particulate matter emitted by the cars. By subtracting that amount from the measured value, we can define the amount contributed by other sources (e.g. buildings).

5.2.5 Mathematical Modeling of Urban Areas

Modeling porous objects inside an urban area can be very challenging because of the presence of smaller elements, such as leaves on trees, which are difficult to resolve. To avoid the computational expense of resolving such small objects, we model them and assume an even distribution of solid materials interspersed with empty spaces, as shown in Figure 5.5. The illustration shows a unified geometric model according to Simonis *et al.* [93] of a porous structure, where the unit cells are indicated by Y_i^ϵ with ϵ representing the side length. The overall fluid domain Ω_ϵ is obtained by removing the solids, i.e.

$$\Omega_\epsilon = \Omega \setminus \bigcup_{i=1}^{N(\epsilon)} Y_{\mathrm{So},i}^\epsilon, \tag{5.4}$$

where $Y_{\mathrm{So},i}^\epsilon$ denotes the solid part of a unit cell, and $N(\epsilon)$ counts their overall number within the porous region Ω.

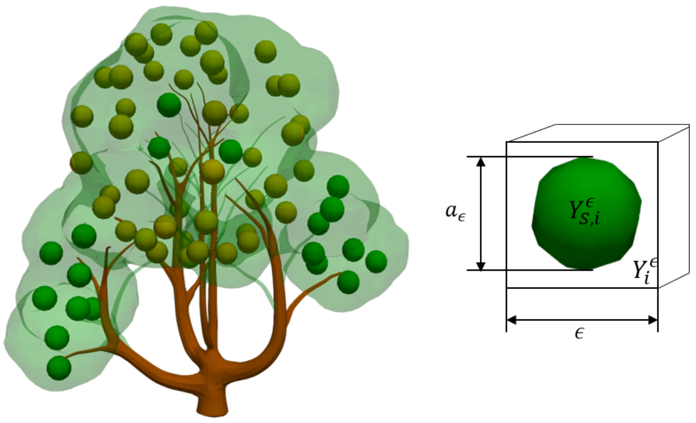


Figure 5.5: *Geometric model of a porous medium in three dimensions ($d = 3$). Left: Subvolume of the porous medium, showing the arrangement of spherical obstacles within the matrix. Right: Detailed view of the ith cell, denoted as Y_i^ϵ, containing a spherical obstacle $Y_{\mathrm{So},i}^\epsilon$, with radius a_ϵ . Each cell is a cube with side length ϵ.*

5.2.6 Homogenized Filtered Brinkman–Navier–Stokes Equation

The Navier–Stokes equations (NSE) for incompressible flow govern the motion of fluids and are fundamental in fluid dynamics. The NSE consist of two primary equations: the mass conservation equation and the momentum equation. To overcome the computational costs of resolving the Kolmogorov scale in turbulent flows, we use a spatial filtering operation that is denoted as $\bar{\cdot}$ below. The mass conservation equation ensures that the fluid density remains constant and is expressed as

$$\nabla \cdot \bar{\boldsymbol{u}}_\epsilon = 0, \quad \text{in } \Omega_\epsilon \times I, \tag{5.5}$$

where $\bar{\boldsymbol{u}}_\epsilon$ is the filtered velocity vector of the fluid and $I \subseteq \mathbb{R}_{>0}$ denotes the time horizon. The momentum equation describes the balance of forces acting on the fluid and is given by

$$\frac{\partial \bar{\boldsymbol{u}}_\epsilon}{\partial t} + \bar{\boldsymbol{u}}_\epsilon \cdot \boldsymbol{\nabla} \bar{\boldsymbol{u}}_\epsilon = -\frac{\boldsymbol{\nabla} \bar{p}_\epsilon}{\rho} + \nu_{\mathrm{mo}} \boldsymbol{\nabla}^2 \bar{\boldsymbol{u}}_\epsilon + \frac{\boldsymbol{F}}{\rho} - \boldsymbol{\nabla} \cdot \mathbf{T}, \quad \text{in } \Omega_\epsilon \times I, \tag{5.6}$$

where $\bar{p}_\epsilon$ is the filtered pressure, ρ is the fluid density, ν_{mo} is the molecular kinematic viscosity of the fluid, and $\boldsymbol{F}$ is the external force acting on the fluid. The additional term $\boldsymbol{\nabla} \cdot \mathbf{T}$ represents the effects of filtering approximated, for example, as small-scale turbulence, modeled using a subgrid-scale stress tensor in large eddy simulation (LES).

Fluid flow through porous media is strongly influenced by the porous structure, which adds additional resistance to the motion of the fluid. To account for this, the NSE on an obstacle level (resolving the porous media) are homogenized with a critical obstacle size (cf. Simonis *et al.* [93]). The homogenization limit in this case results in incorporating the Brinkman-term while also applying a filtering operation to capture large-scale turbulence effects using LES. This yields the filtered Brinkman–Navier–Stokes equations (FBNSE)

$$\begin{cases} \boldsymbol{\nabla} \cdot \bar{\boldsymbol{u}} = 0, & \text{in } \Omega \times I, \\ \frac{\partial \bar{\boldsymbol{u}}}{\partial t} + \bar{\boldsymbol{u}} \cdot \boldsymbol{\nabla} \bar{\boldsymbol{u}} = -\frac{\boldsymbol{\nabla} \bar{p}}{\rho} + \nu_{\mathrm{mo}} \boldsymbol{\nabla}^2 \bar{\boldsymbol{u}} + \frac{\nu_{\mathrm{mo}}}{K} \bar{\boldsymbol{u}} - \boldsymbol{\nabla} \cdot \mathbf{T}_{\mathrm{sgs}}, & \text{in } \Omega \times I, \end{cases} \tag{5.7}$$

where $K > 0$ is the permeability coefficient of the porous medium and is given by the Forchheimer equation

$$K = \frac{\mu_F Q}{A\left(\frac{\Delta P}{L} - \frac{\rho}{K_\beta} \frac{Q^2}{A^2}\right)}, \tag{5.8}$$

with μ_F being the dynamic viscosity of the fluid, Q the volume flow rate, L the characteristic length,A the projected area, $\triangle P$ the pressure difference and K_β the nonlinear permeability coefficient. The term $\frac{\nu_{\mathrm{mo}}}{K} \bar{\boldsymbol{u}}$ represents the additional resistance due to the porous structure, which is proportional to the viscosity of the fluid and inversely proportional to the permeability. In (5.7), we approximate $\mathbf{T} \approx \mathbf{T}_{\mathrm{sgs}}$ such that the term $\boldsymbol{\nabla} \cdot \mathbf{T}_{\mathrm{sgs}}$ accounts for the subgrid-scale turbulence modeled by the Smagorinsky LES approach

$$\mathbf{T}_{\mathrm{sgs}} = 2 \nu_{\mathrm{turb}} \bar{\mathbf{S}}, \tag{5.9}$$

$$\nu_{\mathrm{turb}} = (C_{\mathrm{S}} \triangle_x)^2 \left| \bar{\mathbf{S}} \right|, \tag{5.10}$$

where $C_{\mathrm{S}} > 0$ is the Smagorinsky constant, $\triangle_x$ is the filter width, and $\bar{\mathbf{S}}$ is the filtered strain rate tensor:

$$\bar{S}_{\alpha\beta} = \frac{1}{2} \left(\frac{\partial \bar{u}_\alpha}{\partial x_\beta} + \frac{\partial \bar{u}_\beta}{\partial x_\alpha} \right). \tag{5.11}$$

In particular, in the simulations conducted in this work, we make use of a composition of pure fluid regions and homogenized porous media regions in the overall model such that K in the Brinkman-term becomes space-dependent.

However, it is essential to account for not only advection effects but also the influence that porous media have on diffusion. The advection–diffusion equation (ADE) for porous media based on the formulation according to Lasaga *et al.* [94] is given as

$$\frac{\partial(\Phi C)}{\partial t} + \boldsymbol{\nabla}\cdot(\Phi C \bar{\boldsymbol{u}}) = D\boldsymbol{\nabla}\cdot(\Phi\boldsymbol{\nabla}C), \quad \text{in } \Omega\times I, \tag{5.12}$$

where C is the concentration, D is the diffusion coefficient, and Φ represents the local porosity. Together with the initial and boundary conditions considered in Section 5.3.2.1, (5.7) and (5.12) form the mathematical model considered in this work.

5.2.7 Numerical Methods and Discretization

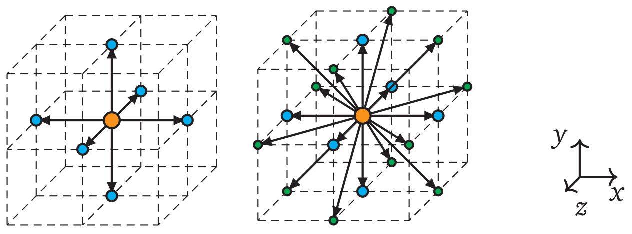
5.2.7.1 The Homogenized Lattice Boltzmann Method for Fluid Flows

Due to the high computational costs required for our DT framework, we use the HLBM to discretize the FBNSE (5.7) on a space-time grid, using the $D3Q19$ velocity stencil illustrated in Figure 5.6. The HLBM inherits the classical advantages of LBM in terms of high parallelizability and efficient scaling with increasing hardware capabilities [95]. In this work, we extend the HLBM initially proposed by Krause *et al.* [96] with the hybrid third-order recursive regularized collision model proposed by Jacob *et al.* [97]. First, we summarize the resulting space-time evolution equation and, afterwards, specify the individual features. Note that, for compactness of notation, we abandon the $\bar{\cdot}$-notation of the filtered quantities below.

The filtered and homogenized lattice Boltzmann equation approximating the FBNSE (5.7) is given by

$$f_i(\boldsymbol{x}+\boldsymbol{c}_i\triangle t, t+\triangle t) = f_i^{\mathrm{h,eq}}(\boldsymbol{x},t)+\left(1-\frac{1}{\tau_{\mathrm{eff}}(\boldsymbol{x},t)}\right)\tilde{f}_i^{(1)}(\boldsymbol{x},t), \quad \text{in } \Omega_{\triangle x}\times I_{\triangle t}, \tag{5.13}$$

where $f_i : \Omega_{\triangle x}\times I_{\triangle t}\to\mathbb{R}_{\geq 0}$ represents the filtered distribution functions, $\boldsymbol{c}_i$ denote the q mesoscopic velocities in $D3Q19$, where $i=0,1,\ldots,q-1$, $\Omega_{\triangle x}\subset\Omega\subseteq\mathbb{R}^3$ is the discretized position space domain with voxel size $\triangle x$, and $I_{\triangle t}\subset I\subseteq\mathbb{R}_{\geq 0}$ is the discrete time horizon with timestep size $\triangle t$. Moreover, τ_{eff} is the space-



(a) *D3Q7* **(b)** *D3Q19*

Figure 5.6: *A schematic illustration of the discrete velocity sets (a) D3Q7 and (b) D3Q19. Coloring refers to energy shells: orange, cyan, green denote zeroth, first, second order, respectively. Figure from [98].*

time adaptive, effective relaxation time of the naive Smagorinsky Bhatnagar–Groos–Krook (BGK) model, given by

$$\tau_{\text{eff}}(\boldsymbol{x},t) = \frac{\nu_{\text{eff}}(\boldsymbol{x},t)}{c_{\text{s}}^2}\frac{\triangle t}{\triangle x^2} + \frac{1}{2}, \tag{5.14}$$

with $\nu_{\text{eff}} = \nu_{\text{mo}} + \nu_{\text{turb}}$ representing the combined molecular and turbulent viscosity (5.10) and c_{s} is the lattice speed of sound. The regularization in (5.13) is based on the non-equilibrium function $f_i^{(1)} = f_i - f_i^{(0)}$ that is expanded as

$$f_i^{(1)}(\boldsymbol{x},t) = \omega_i \sum_{n=0}^{N=3} \frac{1}{c_{\text{s}}^{2n} n!}\mathbf{H}_i^{(n)} : \mathbf{a}_1^{(n)}(\boldsymbol{x},t), \tag{5.15}$$

where ω_i are the lattice weights and we denote by $\mathbf{H}_i^{(n)}$ the nth order Hermite polynomial with the ith discrete velocity $\mathbf{c}_i$ as an argument. The Hermite coefficient for the non-equilibrium is defined as

$$\mathbf{a}_1^{(n)}(\boldsymbol{x},t) = \sum_{i=0}^{q-1} \mathbf{H}_i^{(n)} f_i^{(1)}(\boldsymbol{x},t), \tag{5.16}$$

Note that according to [97], we use Hermite polynomials that have correct orthogonality for *D*3*Q*19 only. Then we add the hybridization of rate of strain via

$$\tilde{f}_i^{(1)}(\boldsymbol{x},t) = f_i^{(1)}(\boldsymbol{x},t)\sigma - (1-\sigma)\frac{\rho\tau}{c_{\text{s}}^2}\mathbf{H}_i^{(2)} : \mathbf{S}^{\text{FD}}(\boldsymbol{x},t), \tag{5.17}$$

where $0 \le \sigma \le 1$ and

$$S_{\alpha\beta}^{\text{FD}} \approx S_{\alpha\beta} + \frac{1}{2}\left[\frac{1}{6}\triangle x^2\left(\frac{\partial^3 u_\beta}{\partial x_\alpha^3} + \frac{\partial^3 u_\alpha}{\partial x_\beta^3}\right)\right] \tag{5.18}$$

is the finite difference (FD) strain rate tensor. Notably, $\sigma = 1$ switches off the FD contribution and reduces the model to a recursively regularized, homogenized Smagorinsky BGK collision. Further, extending the method to homogenized fluid flow, we define the homogenized equilibrium Hermite coefficients $\widehat{\mathbf{a}}_0^{(n)} = \mathbf{a}_0^{(n-1)}\widehat{\boldsymbol{u}}$ with $\widehat{\mathbf{a}}_0^{(0)} = \rho$. Then, the homogenized equilibrium distribution function is defined as

$$f_i^{\mathrm{h,eq}}(\boldsymbol{x},t) = \omega_i \left(\rho(\boldsymbol{x},t) + \frac{\boldsymbol{c}_i \cdot (\rho\, \widehat{\mathbf{u}}(\boldsymbol{x},t))}{c_s^2} + \frac{\mathbf{H}_i^{(2)} : \widehat{\mathbf{a}}_0^{(2)}(\boldsymbol{x},t)}{2c_s^4} + \frac{\mathbf{H}_i^{(3)} : \widehat{\mathbf{a}}_0^{(3)}(\boldsymbol{x},t)}{2c_s^6} \right), \tag{5.19}$$

where ρ is the zeroth order density moment and $\widehat{\boldsymbol{u}}$ is the homogenized macroscopic velocity field that incorporates the permeability effects in (5.7) from the partial homogenization of the domain. In the general case (moving solid objects [96]) the homogenized velocity is defined as a convex combination of the fluid velocity moment $\boldsymbol{u}$ and the solid object velocity $\boldsymbol{u}^{\mathrm{B}}$, given by

$$\widehat{\boldsymbol{u}}(\boldsymbol{x},t) = (1 - d(\boldsymbol{x},t))\boldsymbol{u}(\boldsymbol{x},t) + d(\boldsymbol{x},t)\boldsymbol{u}^{\mathrm{B}}(\boldsymbol{x},t), \tag{5.20}$$

where d is the so-called lattice porosity. In the present case (rigid obstacles, i.e. $\boldsymbol{u}^{\mathrm{B}} = 0$), (5.20) reduces to

$$\widehat{\boldsymbol{u}}(\boldsymbol{x},t) = (1 - d(\boldsymbol{x},t))\boldsymbol{u}(\boldsymbol{x},t), \tag{5.21}$$

where

$$d(\boldsymbol{x},t) = 1 - \frac{\triangle x^2 \nu \tau_{\mathrm{mo}}}{K(\boldsymbol{x},t)}, \tag{5.22}$$

the permeability given in (5.7) is identified with K and τ_{mo} is the molecular relaxation time, defined similarly as in (5.14) but with ν_{mo}.

In [93], a standalone derivation and analysis of the continuous homogenized kinetic model with a BGK collision for fluid flow in porous media is proposed. For unfiltered fields, a formal estimate for the order of approximation is derived in [98]. In general, the fluid velocity moment $\boldsymbol{u}$ from (5.13) is expected to provide a second-order approximation in space of a filtered velocity solution of (5.7).

5.2.8 The Homogenized Lattice Boltzmann Method for Advection–Diffusion Processes

So far, HLBM (5.13) recovers the flow in and around porous media modeled by (5.7). When coupled with a discretization of the ADE (5.12), the overall numerical method is able to reflect the unique behavior of the particles within a porous environment. To incorporate these effects into the HLBM, a second lattice Boltzmann equation is coupled to (5.13) that includes a correction term $\mathscr{R}$. This term is derived below and accounts for temporal and spatial variations in porosity, enabling an accurate reflection of the local concentration within the porous media. The correction term is derived by expanding each component of (5.12) separately and isolating the temporal and spatial variations of Φ. The first term of (5.12) is expanded using the time derivative as

$$\frac{\partial(\Phi C)}{\partial t} = \Phi \frac{\partial C}{\partial t} + C \frac{\partial \Phi}{\partial t}. \tag{5.23}$$

Similarly, the advective term is expanded as

$$\boldsymbol{\nabla} \cdot (\Phi C \boldsymbol{u}) = \Phi \boldsymbol{u} \cdot \boldsymbol{\nabla} C + C \boldsymbol{u} \cdot \boldsymbol{\nabla} \Phi + C\Phi(\boldsymbol{\nabla} \cdot \boldsymbol{u}), \tag{5.24}$$

and the diffusive term expands to

$$D\boldsymbol{\nabla} \cdot (\Phi \boldsymbol{\nabla} C) = D(\boldsymbol{\nabla}\Phi) \cdot \boldsymbol{\nabla} C + D\Phi \boldsymbol{\nabla}^2 C. \tag{5.25}$$

Combining the temporal and spatial variations from (5.23) to (5.25), we obtain the correction term

$$\mathscr{R} = -C \frac{\partial_t \Phi}{\partial t} \frac{1}{\Phi} - C\boldsymbol{u}\,(1 + D\boldsymbol{\nabla} C)\, \frac{\boldsymbol{\nabla}\Phi}{\Phi}. \tag{5.26}$$

With a finite difference approximation of (5.26) we get

$$\begin{aligned}
\mathscr{R}(\boldsymbol{x}_j, t_n) &\approx \mathscr{R}_{\triangle x, \triangle t}(\boldsymbol{x}_j, t_n) \\
&= -C_j^n \frac{\Phi_j^n - \Phi_j^{n-1}}{\Phi_j^n \triangle t} \\
&\quad - C_j^n \Bigg(u_x \frac{\Phi^n_{j+(1,0,0)} - \Phi^n_{j-(1,0,0)}}{2\Phi_j^n \triangle x} + u_y \frac{\Phi^n_{j+(0,1,0)} - \Phi^n_{j-(0,1,0)}}{2\Phi_j^n \triangle x} \\
&\qquad + u_z \frac{\Phi^n_{j+(0,0,1)} - \Phi^n_{j-(0,0,1)}}{2\Phi_j^n \triangle x} \Bigg) \\
&\quad + D \Bigg(\frac{\left(C^n_{j+(1,0,0)} - C^n_{j-(1,0,0)}\right)\left(\Phi^n_{j+(1,0,0)} - \Phi^n_{j-(1,0,0)}\right)}{4\Phi_j^n \triangle x^2} \\
&\qquad + \frac{\left(C^n_{j+(0,1,0)} - C^n_{j-(0,1,0)}\right)\left(\Phi^n_{j+(0,1,0)} - \Phi^n_{j-(0,1,0)}\right)}{4\Phi_j^n \triangle x^2} \\
&\qquad + \frac{\left(C^n_{j+(0,0,1)} - C^n_{j-(0,0,1)}\right)\left(\Phi^n_{j+(0,0,1)} - \Phi^n_{j-(0,0,1)}\right)}{4\Phi_j^n \triangle x^2} \Bigg)
\end{aligned} \tag{5.27}$$

for $\cdot(\boldsymbol{x}_j, t_n) = \cdot_j^n$ at grid nodes $\boldsymbol{x}_j \in \Omega_{\triangle x}$ and time steps $t_n \in I_{\triangle t}$, which is second order in space and first order in time. Asymmetric stencils of the same order are used at the domain boundaries. The LBM discretization of the advection–diffusion process, including a modified collision operator to account for spatial porosity variations through homogenization, is expressed as

$$g_i(\boldsymbol{x}+\boldsymbol{c}_i\triangle t, t+\triangle t) - g_i(\boldsymbol{x}, t) = \frac{1}{\tau_D}\left(g_i^{\mathrm{eq}}(\boldsymbol{x}, t) - g_i(\boldsymbol{x}, t)\right) + J^{\mathrm{por}}(\boldsymbol{x}, t), \quad \text{in } \Omega_{\triangle x} \times I_{\triangle t}, \tag{5.28}$$

where τ_D is connected to the diffusivity D similarly as in (5.10). The equilibrium distribution function g_i^{eq}, representing the equilibrium concentration distribution, is given by

$$g_i^{\mathrm{eq}}(\boldsymbol{x}, t) = w_i C(\boldsymbol{x}, t)\left(1 + \frac{\boldsymbol{c}_i \cdot \boldsymbol{u}(\boldsymbol{x}, t)}{c_s^2}\right), \tag{5.29}$$

where $\boldsymbol{u}$ is the velocity moment from (5.13) and the concentration C is calculated as the zeroth moment of the populations g_i with an additional term derived from the second-order finite difference approximation of (5.26) named $\mathscr{R}_{\triangle x, \triangle t}$ and defined in (5.27), i.e.

$$C(\boldsymbol{x}, t) = \sum_i g_i(\boldsymbol{x}, t) + \frac{1}{2}\mathscr{R}_{\triangle x, \triangle t}(\boldsymbol{x}, t). \tag{5.30}$$

The additional collision term, J^{por} in (5.28), is introduced to adjust the transport dynamics in response to spatial porosity variations and reads

$$J^{\mathrm{por}}(\boldsymbol{x}, t) = \left(1 - \frac{1}{2\tau_D}\right) w_i \mathscr{R}_{\triangle x, \triangle t}(\boldsymbol{x}, t). \tag{5.31}$$

The term (5.31) modifies the overall collision operator to ensure that the equilibrium distribution and the transport properties adapt to local porosity changes. The porosity Φ is incorporated in (5.28) by using the lattice porosity $d(\boldsymbol{x}, t)$ from (5.22) in the correction term $\mathscr{R}_{\triangle x, \triangle t}$. Due to the linearity of (5.29), we use the $D3Q7$ velocity set (see Figure 5.6) to reduce computational effort. The second order in space of approximations by LBMs for advection–diffusion equations has been proven among others by Simonis *et al.* [98–100].

5.3 Validation

In this section, we present the validation of our numerical approach by comparing simulation results against analytical solutions and experimental data. First, we validate the proposed double-distribution HLBM with a correction term using an analytical benchmark, ensuring the accuracy of the method in solving coupled FBNSE and ADE systems in porous media. Second, we assess the simulation results against wind channel experiments to confirm the applicability of the model for real-world urban flow scenarios.

5.3.1 Validation of HLBM with Correction Term

To validate the effectiveness of the HLBM ((5.13) and (5.28)) with the correction term (5.27) introduced to approximate the FBNSE (5.7) coupled to the ADE (5.12) in porous media, the numerical results are compared with an analytical solution. The analytical solution serves as a benchmark and is given by the following equations:

$$\Phi(\boldsymbol{x},t) = 0.5 + 0.4\sin(2\pi(x - u_{\mathrm{p}}t)), \tag{5.32}$$

$$C(\boldsymbol{x},t) = \frac{1}{\Phi(\boldsymbol{x},t)}, \tag{5.33}$$

$$\boldsymbol{u}(\boldsymbol{x},t) = D\Phi(\boldsymbol{x},t)\boldsymbol{\nabla}C(\boldsymbol{x},t). \tag{5.34}$$

The analytical solution represents a sinusoidal scalar field $\Phi(\boldsymbol{x},t)$ that propagates in the x-direction with a propagation speed u_{p}. This form is commonly used in theoretical studies, as it satisfies the ADE under idealized conditions, incorporating both advection (through u_{p}) and diffusion (through D). To evaluate the performance of the HLBM approach, the error was analyzed in terms of different norms (L^1, L^2 and L^∞) across a range of resolutions and are defined as

$$L^1 = \sum_{i=1}^{n} |C_i - C_i^{\mathrm{true}}| \tag{5.35}$$

$$L^2 = \sqrt{\sum_{i=1}^{n} (C_i - C_i^{\mathrm{true}})^2} \tag{5.36}$$

$$L^\infty = \max_{1\le i\le n} |C_i - C_i^{\mathrm{true}}| \tag{5.37}$$

The experimental order of convergence (EOC) in Figure 5.7 illustrates the convergence of the error to the analytical solution as the resolution improves. The consistent decrease in error for all norms confirms the stability and accuracy of the method. Specifically, the second-order observed EOC matches the theoretical expectations, demonstrating that the correction term successfully enhances the numerical approach.

The results validate the applicability of the HLBM with the correction term for modeling ADEs in porous media. This is particularly relevant for complex porous structures, such as urban environments, where accurate simulations of pollutant transport are essential. The observed convergence trends highlight the robustness of the method.

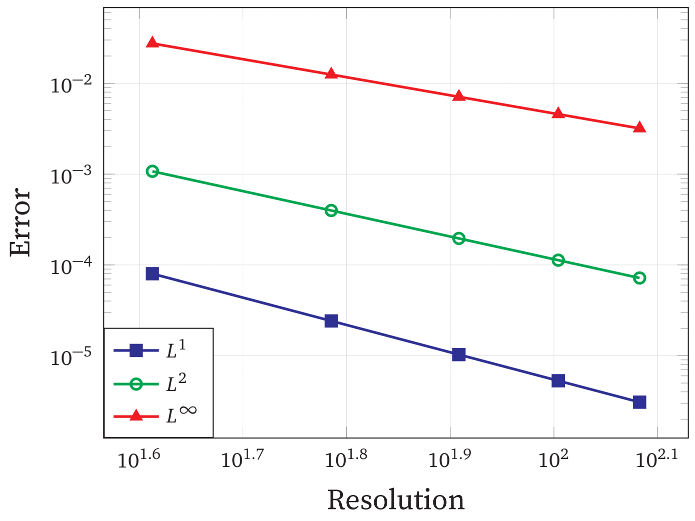


Figure 5.7: *Experimental order of convergence of the error norms L^1, L^2 and L^∞ to an analytical solution. It shows that the error gets smaller with higher resolution.*

5.3.2 Validation with Experimental Data from a Wind Channel

In order to validate the simulation part of the DT with the HLBM approach we use the work of Gromke *et al.* [101] where they validate the CFD results from the software FLUENT with measurements from a wind channel. The geometric setup is made up of two solid square blocks, which represent buildings with the between being a street canyon with a porous zone representing a tree in the middle as shown in Figure 5.8.

5.3.2.1 Setup

Two setups were used for the simulations, both following the geometrical setup of the experiment by Gromke *et al.* [101] and the simulation setup from [102], as shown in Figure 5.8. The complete boundary configuration is described as

$$\Gamma = \partial\Omega = \Gamma_{\mathrm{in}} \cup \Gamma_{\mathrm{out}} \cup \Gamma_{\mathrm{ground}} \cup \Gamma_{\mathrm{wall}} \cup \Gamma_{\mathrm{sky}} \cup \Gamma_{\mathrm{left}} \cup \Gamma_{\mathrm{right}} \cup \Gamma_{\mathrm{e}}. \tag{5.38}$$

The homogenized porous media region that models the trees is indicated as Γ_{tree} (see Figure 5.8) and is used in one of the setups. The ground Γ_{ground} and Γ_{wall} are set to no-slip Dirichlet conditions, enforcing zero velocity at these surfaces, i.e.

$$\boldsymbol{u}(\boldsymbol{x},t) = \boldsymbol{0}, \quad \text{on } \Gamma_k \times (0,T], \quad \forall \Gamma_k \in \{\Gamma_{\text{ground}}, \Gamma_{\text{wall}}\}. \tag{5.39}$$

We use a full-slip boundary condition for the fluid velocity at the open boundaries, i.e.

$$\boldsymbol{u}(\boldsymbol{x},t)\cdot\boldsymbol{n} = 0 \quad \text{on } \Gamma_k \times (0,T], \quad \forall \Gamma_k \in \{\Gamma_{\text{sky}}, \Gamma_{\text{left}}, \Gamma_{\text{right}}\}, \tag{5.40}$$

where $\boldsymbol{n}$ is the outward pointing normal vector. Specifically, a full slip boundary condition is applied at Γ_{sky}, allowing tangential flow without friction. At the inflow Γ_{in} we apply a Dirichlet condition with an interpolated inlet velocity profile, while at the outflow Γ_{out} we use a fixed pressure boundary, allowing natural flow out of the domain. For the inlet Γ_{in}, the same velocity profile as in [76] is used, depicted as

$$u(z) = U_{\text{H}}\left(\frac{z}{z_{\text{H}}}\right)^{\alpha}, \tag{5.41}$$

where $\alpha = 0.3$, U_{H} is the inflow velocity at height z_{H}, and z_{H} is the height of the walls. For the turbulence at the inlet, we used the vortex method according to Hettel *et al.* [103], where vortices are randomly predefined at the inlet. As in [101, 102] Sulfur hexafluoride (SF_6) was used as a tracer gas and is emitted from Γ_{e} with a z-velocity of U_{e}. Thus,

$$\boldsymbol{u}(\boldsymbol{x},t) = (0,0,U_{\text{e}})^{\text{T}}, \quad \text{on } \Gamma_{\text{e}} \times (0,T] \tag{5.42}$$

and

$$C(\boldsymbol{x},t) = C_{\text{SF}_6}, \quad \text{on } \Gamma_{\text{e}} \times (0,T], \tag{5.43}$$

where C_{SF_6} is the emitted concentration of SF_6. The numerical parameters for the simulation, including the number of cells, are shown in Table 5.1 where the maximum simulation time is represented by T_{tot}.

In this setup, two lattices are used: one for the airflow distribution and one for the concentration distribution of the pollution. At specific boundary regions, we collide additionally with the chosen dynamics approximating the FBNSE or the ADE, respectively. This is particularly relevant at the inflow regions of the

Table 5.1: *Physical and numerical parameters. Units are denoted in brackets.*

$Re\,[-]$	$Sc_t\,[-]$	$U_H\,[\mathrm{m/s}]$	$\triangle x\,[\mathrm{m}]$	$\triangle t\,[\mathrm{s}]$	$U_e\,[\mathrm{m/s}]$	$T_{tot}\,[\mathrm{s}]$	$H\,[\mathrm{m}]$	#cells $[-]$
37000	0.3	4.65	2×10^{-2}	1.8×10^{-5}	0.2054	25	0.12	438.705×10^{6}

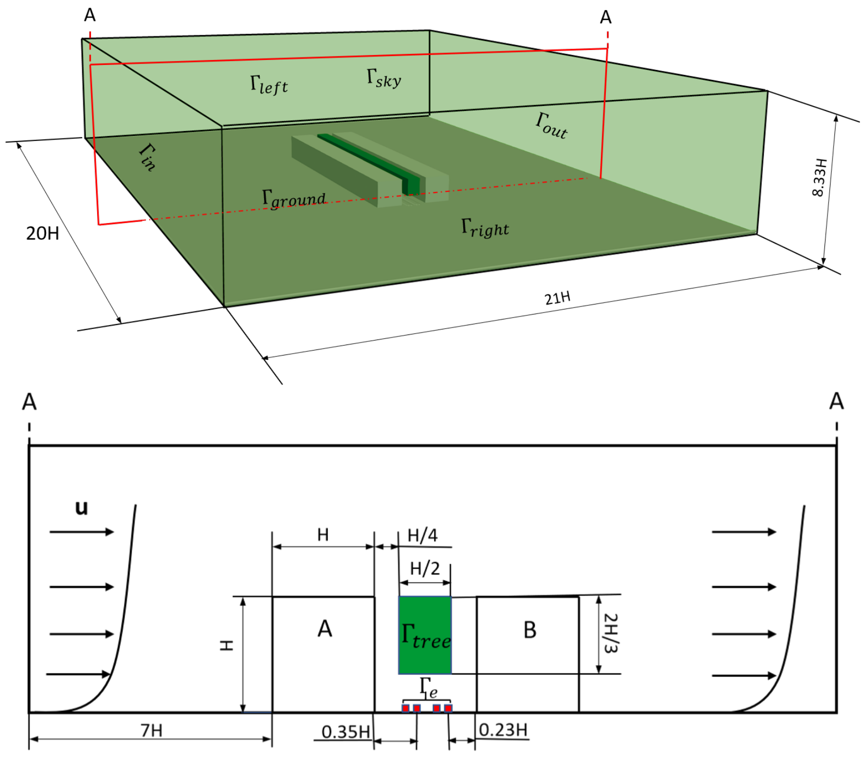


Figure 5.8: *Simulation setup for airflow and pollutant distribution in an urban environment. The 3D view (top) shows the boundary conditions (Γ) and placement of urban structures within the simulation domain. The 2D cross-sectional view (bottom) illustrates the wind profile and flow around a street canyon (Wall A and Wall B), a tree (Γ_{tree}), and four line emitters (Γ_e) within the domain.*

domain Γ_{in} and Γ_{e}. For the airflow distribution, we apply the collision according to

$$\forall \Gamma_k \in \{\Gamma_{\text{in}}, \Gamma_{\text{e}}\} : \text{FBNSE (5.7)}. \tag{5.44}$$

For the concentration distribution, we apply the collision according to

$$\Gamma_{\text{e}} : \text{ADE (5.12)}. \tag{5.45}$$

Further, for the boundary conditions for the concentration distribution are

$$\boldsymbol{g}(\boldsymbol{x}, t) = \boldsymbol{0}, \quad \text{on } \Gamma_k \times (0, T], \quad \forall \Gamma_k \in \{\Gamma_{\text{in}}, \Gamma_{\text{out}}, \Gamma_{\text{sky}}, \Gamma_{\text{left}}, \Gamma_{\text{right}}\}, \tag{5.46}$$

$$C(\boldsymbol{x}, t) = C_0, \quad \text{on } \Gamma_k \times (0, T], \quad \forall \Gamma_k \in \{\Gamma_{\text{ground}}, \Gamma_{\text{wall}}\}, \tag{5.47}$$

where $C_0 = \text{const}$ is realized with standard bounce-back.

5.3.2.2 Performance

To assess the computational efficiency of the HLBM simulations, we measure performance in terms of mega lattice updates per second (MLUPs). The performance is evaluated under different configurations: with and without trees, and with and without a wall function (WF). The results provide insight into the computational cost associated with modeling urban flow scenarios and are shown in Table 5.2. It is observed that the porous ADE correction from (5.31) has

Table 5.2: *Computational performance in MLUPs for the configurations used in this work.*

Configuration	Without WF [MLUPs]	With WF [MLUPs]
Tree-less canyon	~ 4600	~ 4100
Tree canyon	~ 4100	~ 2500

a big impact on performance, which results in a loss of ~ 500 MLUPs, by adding the WF it is reduced even further. During this work, the WF is not optimized in its implementation and is therefore rather inefficient compared to the rest of the code. The performance was investigated on a GPU node (4x NVIDIA H100 GPUs) on the Horeka cluster at KIT.

5.3.2.3 Simulation Results

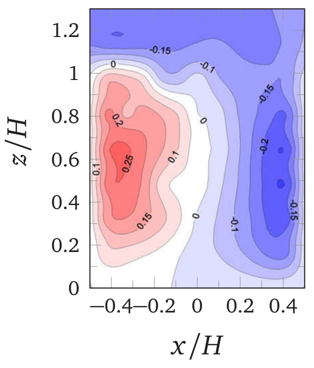
A series of simulations were performed to evaluate airflow and pollutant dispersion in a street canyon, comparing the results against experimental data from

Gromke *et al.* [101]. Two configurations were studied: a street canyon without trees and one with a porous tree. Each scenario was simulated with and without a WF, following the methodology outlined by Guo *et al.* [104, 105]. The Spalding wall function was employed to approximate equilibrium and non-equilibrium velocity profiles, with Dirichlet boundary conditions applied as described earlier. The concentration measured at the walls A and B was normalized according to

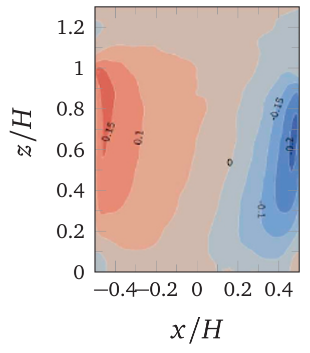
$$c_+ = \frac{c_\mathrm{m} U_\mathrm{H} H}{Q_{\mathrm{SF}_6}/l}, \tag{5.48}$$

where c_m is the concentration, $Q_{\mathrm{SF}_6} = 1.359 \times 10^{-6}\,\mathrm{m}^3/\mathrm{s}$ is the volumetric flow rate of SF_6 and $l = 1.42\,\mathrm{m}$ the length of the line sources. The objective is to evaluate the usability of the above described numerical approach as a model for a DT of a city as well as to observe the impact of the WF.

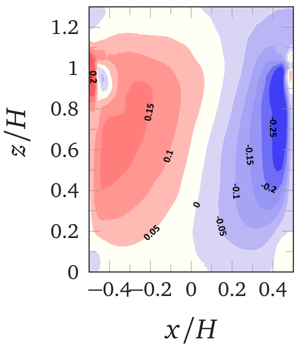
Tree-free canyon: Figure 5.9 presents the normalized vertical velocity profiles for a canyon without a tree (tree-free canyon). The subfigures compare the experimental data with numerical results from other studies and the current study, both with and without a wall function. In Figure 5.9 (a), the experimental results of Gromke *et al.* [101] provide the baseline for comparison. Subfigures (b) and (c) illustrate the results of simulations without a WF. Although the general velocity profiles are consistent with the experimental data, discrepancies are observed at the edges of the wall, where the maximum vertical velocities are overestimated. Figures 5.9 (d) and (e) incorporate a WF in the simulations. Here, the maximum vertical velocity is shifted toward the center of the canyon, aligning more closely with the experimental observations. Figure 5.9 (e) demonstrates particularly good agreement with the experimental data, validating the effectiveness of the wall model in capturing the critical flow characteristics in urban environments. The pollutant concentration distribution is further analyzed in Figures 5.10 and 5.11, which show normalized concentrations along walls A and B, respectively, in the absence of trees. In Figure 5.10 (a), experimental data from Gromke *et al.* [101] illustrate the expected pollutant accumulation patterns. Subfigures (b) and (c) present numerical results without a WF. Although the general trends and stratification of the concentration layers align with the experimental data, the pollutant accumulation zones are narrower and exhibit higher gradients. This behavior is attributed to the elevated vertical velocities near the edges of the wall, as seen in Figures 5.9 (b) and (c). The thinner pollutant layers near the walls indicate an over-prediction of upward pollutant transport. When using a WF, as shown in Figures 5.10 (d) and (e), the pollutant accumulation patterns show improved agreement with the experimental results. The concentration layers are wider and better reflect the observed stratification, particularly along the central portions of the wall. This improvement corresponds to the more realistic velocity profiles shown in Figures 5.9 (d) and (e), where the maximum vertical velocities are centered and reduced near the walls.



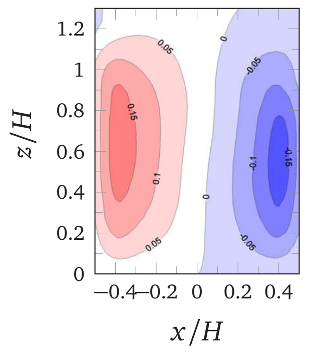
(a) *Experiment Gromke et al. [101]*



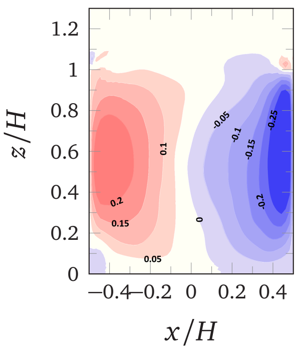
(b) *Merlier et al. [102] without WF*



(c) *Ours without WF*



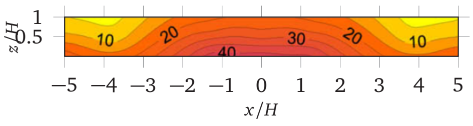
(d) *Gromke et al. [101] with WF*



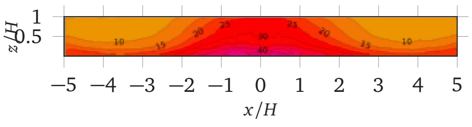
(e) *Ours with WF*

Figure 5.9: *The experimental and simulation results for the normalized vertical velocity of a tree free canyon are shown for the plane cut A-A (Figure 5.8) between Wall A and Wall B. The experimental results from Gromke et al. [101] are shown in (a), (b) shows numerical results from Merlier et al. [102] without a WF, (c) shows our results without WF, (d) shows the numerical results of Gromke et al. [101] with WF and (e) shows ours with a WF.*

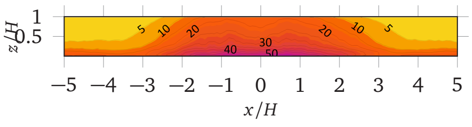
Along wall B depicted in Figure 5.11 the results without a WF differ strongly. The reference case in (b) shows stratified layers with concentrations higher than those of the experiment data in (a), while our results in (c) show strong alignment with (a). The results with a WF in (d) and (e) also agree with (a).



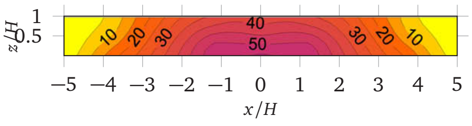
(a) *Experiment Gromke et al. [101]*



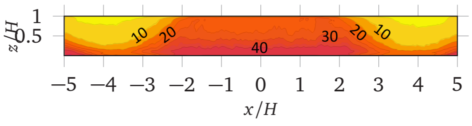
(b) *Merlier et al. [102] without WF*



(c) *Ours without WF*

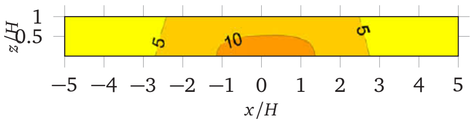


(d) *Gromke et al. [101] with WF*

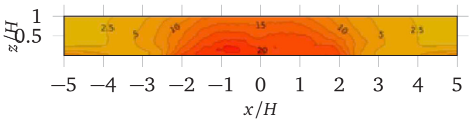


(e) *Ours with WF*

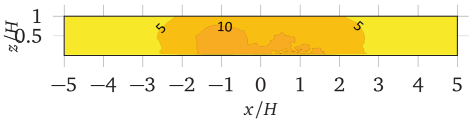
Figure 5.10: *Normalized concentration c_+ at the Wall A. (a) shows the experimental results according to Gromke et al. [101], (b) the results without a WF from Merlier et al. [102], (c) ours without a WF, (d) results from Gromke et al. [101] with a WF and (e) shows our with WF*



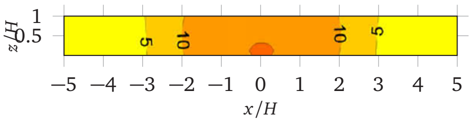
(a) *Experiment Gromke et al. [101]*



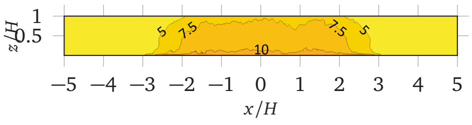
(b) *Merlier et al. [102] without WF*



(c) *Ours without WF*



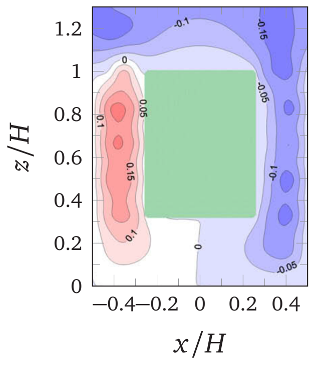
(d) *Gromke et al. [101] with WF*

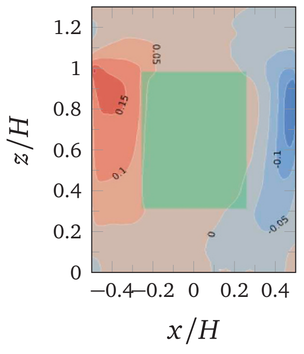


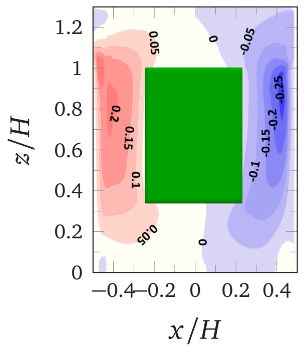
(e) *Ours with WF*

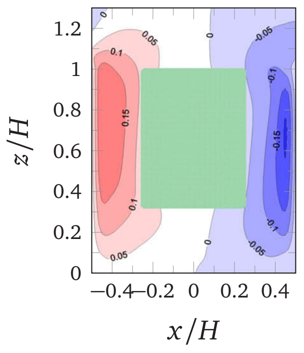
Figure 5.11: *Normalized concentration c_+ at the Wall B. (a) shows the experimental results according to Gromke et al. [101], (b) the results without a WF from Merlier et al. [102], (c) ours without a WF, (d) results from Gromke et al. [101] with a WF and (e) shows our with WF*

Tree-canyon: Similar to the tree-free canyon case, Figure 5.12 presents the simulation results for the velocity distribution in a street canyon with a porous tree. These results are compared against experimental data from Gromke *et al.* [101] (Figure 5.12 (a)) and numerical results from Merlier *et al.* [102] (Figure 5.12 (b)), as well as reference simulations with and without WF. Figures 5.12 (c) and (e), which represent our numerical results without and with WF, respectively, show a similar overall velocity distribution to the experimental data in (a). However, slight differences can be observed, particularly in the velocity magnitude near the walls. Without WF (c), the velocity distribution captures the general pattern observed in (a) but exhibits some deviations in the center and near the edges of the canyon. When WF is included (e), the results remain largely consistent with (c), suggesting that WF has a minimal impact in this setup due to the narrow spacing between the walls and the porous tree zone. Comparison with Merlier *et al.* [102] (b) and the numerical results of Gromke *et al.* [101] (d) shows that our simulations align well with previous numerical approaches, reinforcing the validity of our model. Figure 5.13 presents the normalized concentration c_+ along wall A, while Figure 5.14 shows the concentration along wall B for a canyon with a tree. The results are compared against experimental data from Gromke *et al.* [101] (Figures 5.13 (a) and 5.14 (a)) and numerical results from Merlier *et al.* [102] (Figures 5.13 (b) and 5.14 (b)), alongside our simulations with and without a WF. For wall A, Figures 5.13 (c) and (e) confirm that the general distribution of concentration aligns with the experimental results in (a). However, differences can be observed in the stratification, with our results showing higher concentration gradients, especially near the center of the wall. The comparison between our results with (e) and without WF (c) shows minimal variation, indicating that WF has little influence in this specific setup. For wall B, Figures 5.14 (c) and (e) are consistent with the experimental results in (a). The concentration remains lower compared to wall A, as expected, and the inclusion of WF (e) does not significantly alter the distribution. Our results also compare well with both Merlier *et al.* [102] (b) and Gromke *et al.* [101] (d), indicating that the general pollutant dispersion trends are well captured by our simulations. In summary, while the concentration gradients and layering differ slightly from the experimental data, our simulations effectively capture the key pollutant distribution trends in the presence of trees. The inclusion of WF enhances accuracy but is not essential for reproducing the general distribution. Given that WF increases computational costs, further optimizations are required before integrating it into large-scale urban digital twin applications without performance trade-offs.


(a) *Experiment Gromke et al. [101]*


(b) *Merlier et al. [102] without WF*


(c) *Ours without WF*


(d) *Gromke et al. [101] with WF*

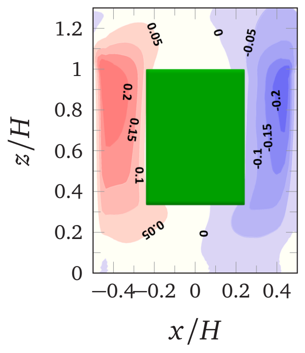
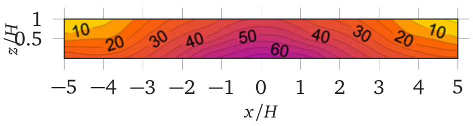
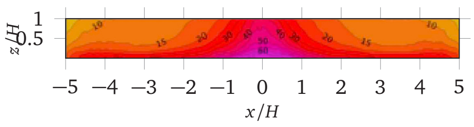

(e) *Ours with WF*

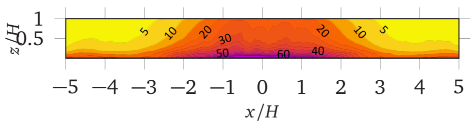
Figure 5.12: *The experimental and simulation results for the normalized vertical velocity of a canyon with a tree are shown for the plane cut A-A (Figure 5.8) between Wall A and Wall B. The experimental results from Gromke et al. [101] are shown in (a), (b) shows numerical results from Merlier et al. [102] without a WF, (c) shows our results without WF, (d) shows the numerical results of Gromke et al. [101] with WF and (e) shows ours with a WF.*



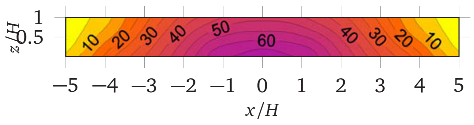
(a) *Experiment Gromke et al. [101]*



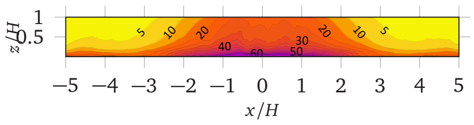
(b) *Merlier et al. [102] without WF*



(c) *Ours without WF*

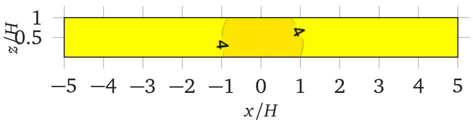


(d) *Gromke et al. [101] with WF*

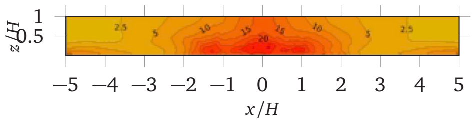


(e) *Ours with WF*

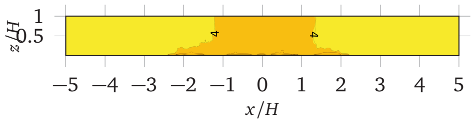
Figure 5.13: *Normalized concentration c_+ at the Wall A for a canyon with a tree. (a) shows the experimental results according to Gromke et al. [101], (b) the results without a WF from Merlier et al. [102], (c) ours without a WF, (d) results from Gromke et al. [101] with a WF and (e) shows our with WF*



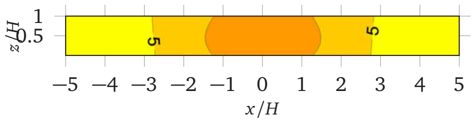
(a) *Experiment Gromke et al. [101]*



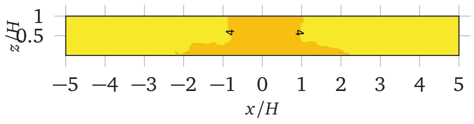
(b) *Merlier et al. [102] without WF*



(c) *Ours without WF*



(d) *Gromke et al. [101] with WF*



(e) *Ours with WF*

Figure 5.14: *Normalized concentration c_+ at the Wall B for a canyon with a tree. (a) shows the experimental results according to Gromke et al. [101], (b) the results without a WF from Merlier et al. [102], (c) ours without a WF, (d) results from Gromke et al. [101] with a WF and (e) shows our with WF*

5.4 Results of the Digital Twin

The test area for the DT prototype is a part of Reutlingen, Germany, as described in Section 5.2.3.2 and shown in Figure 5.4. This area was chosen due to the availability of publicly accessible air pollution data from two monitoring stations. These data sources are continuously integrated into the simulation, enabling real-time model updates and improving the accuracy of the results through data fusion.

5.4.1 Setup

To construct the DT of the urban environment, a high-resolution computational model was developed using data from OSM, meteorological stations, and emission sources as shown in Figure 5.15. The selected study area includes Alteburgstraße and Lederstraße, two major roads in the region, as well as vegetation, buildings, and traffic-related emission sources. The simulation aims to model pollutant dispersion and air flow dynamics in this urban neighborhood, considering contributions from road traffic emissions, building exhaust sources, and environmental factors such as the effects of wind and vegetation. Key sources of pollution in the study area include the following:

- Road Transport Emissions: Emissions from vehicles traveling along Alteburgstraße and Lederstraße.
- Building Exhaust Emissions: Pollutants released from heating, ventilation, and industrial sources.
- Vegetation Effects: Trees and vegetation act as natural barriers that influence pollutant dispersion by affecting airflow and deposition processes.

The numerical setup is the same as in the validation from Section 5.3.2.3 with the exception being the vortex method, since it is not suitable for changing wind directions. Instead, the turbulence is induced by varying the wind direction with in a 5% margin of the current wind direction. Figure 5.16 shows the NO_2 and $PM_{2.5}$ concentrations of measured from the measuring station over the course of the week. It is observed that the NO_2 value highly correlates with the time of date, which is as expected when considering work traffic. The $PM_{2.5}$ values are more stable during the week.

Table 5.3 shows the physical and numerical parameters of the simulation. The Reynolds number was omitted because it being not practical for this simulation, because of its relation to the wind speeds. T_{up} represents the amount

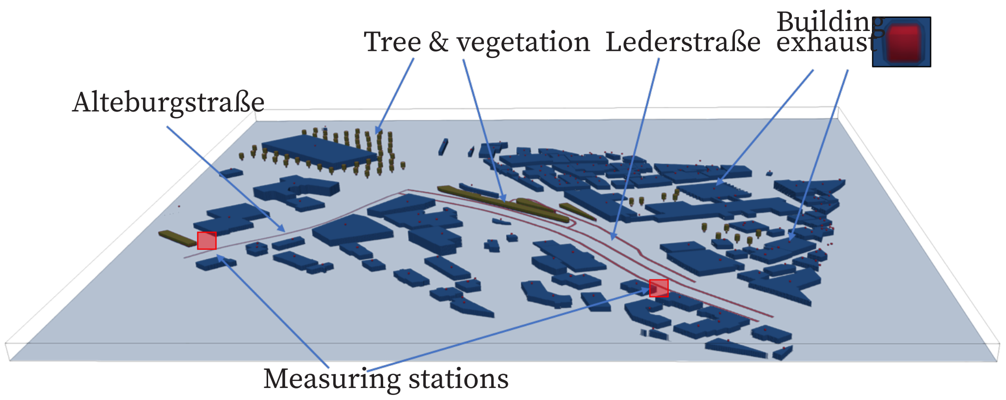
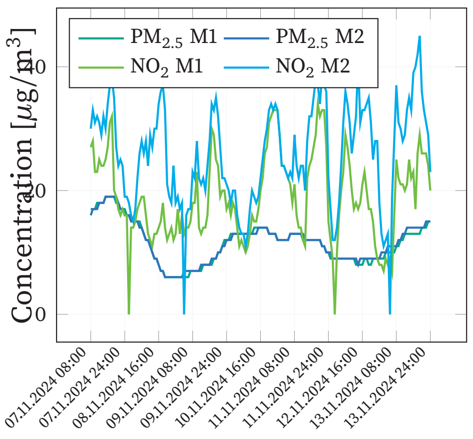
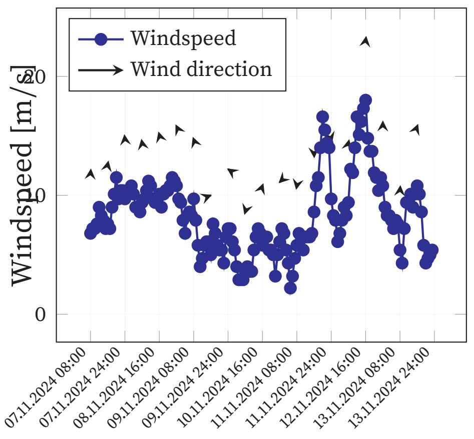


Figure 5.15: *Computational domain and measurement setup for the DT simulation. Buildings are colored blue, while trees and vegetation are colored green. Two measuring stations (red squares) are placed along Alteburgstraße (M1) and Lederstraße (M2) to monitor air pollution. The road network is colored red to indicate major traffic emission sources. Building exhaust emissions are visualized with red markers on structures, highlighting additional stationary pollution sources. The model integrates real-time data from these stations to dynamically update the simulation of pollutant dispersion and airflow behavior.*



(a) *Concentration*



(b) *Windspeed and wind direction*

Figure 5.16: *(a) Time series of pollutant concentrations, including PM$_{2.5}$ and NO$_2$ at two measurement stations (M1 and M2). (b) Time series of wind speed with overlaid arrows indicating wind direction. The arrows are oriented towards the cardinal directions (north, east, south, west) based on recorded wind direction data.*

of time until the simulation is updated with new wind and concentration data, while U_e is the emission speed of the pollution sources.

Table 5.3: *Physical and numerical parameters.*

$Sc_t\,[-]$	$\triangle x\,[\mathrm{m}]$	$\triangle t\,[\mathrm{s}]$	$U_e\,[\mathrm{m/s}]$	$T_{up}\,[\mathrm{s}]$	$T_{tot}\,[\mathrm{s}]$	#cells [−]
0.3	1	1.3×10^{-3}	0.2054	300	46000	48.25×10^{6}

The simulation achieved similar performance results as shown in Table 5.2 for the configuration with trees and without WF.

5.4.2 Simulation Results

The DT simulation was conducted from November 7 to November 13, 2024, with hourly updates, resulting in a total of 162 updates. The results presented in Figure 5.17 illustrate the velocity distribution (left column) and the $PM_{2.5}$ concentration distribution (right column) at different timestamps within the simulation period.

The velocity distributions shown in Figures 5.17 (a), (c), and (e) depict the averaged wind flow at a height of 2 meters above ground level, representing pedestrian exposure. The prevailing wind direction on November 7 (Figure 5.17 (a)) is predominantly from the northwest, leading to strong airflow through urban canyons and open spaces. By November 9 (Figure 5.17 (c)), the wind intensity appears slightly reduced, with minor directional changes. The flow near built-up areas remains relatively stable, while turbulence effects are noticeable around obstacles. On November 13 (Figure 5.17 (e)), the wind direction has changed to the northeast, affecting airflow patterns and creating lower-velocity zones where stagnation may occur.

The corresponding $PM_{2.5}$ concentration distributions in Figures 5.17 (b), (d), and (f) reveal the relationship between wind dynamics and pollutant accumulation. On November 7 (Figure 5.17 (b)), pollution hot-spots are primarily located in the southeastern part of the domain, where lower ventilation leads to increased concentration levels. This aligns with the northwest wind direction, which carries pollutants toward these regions. By November 9 (Figure 5.17 (d)), the pollution distribution has changed slightly due to changing meteorological conditions, yet high concentrations remain in areas with reduced airflow. On November 13 (Figure 5.17 (f)), the northeast wind redistributes pollutants, forming new hot-spots while decreasing concentrations in previously affected areas.

The results highlight the strong correlation between wind direction and pollutant dispersion patterns. The built-up areas and the effects of the urban canyons contribute to localized stagnation zones where pollutants remain trapped for extended periods. The formation of these hot-spots suggests the necessity for mitigation strategies such as geometrical adjustments to building layouts or the strategic placement of vegetation to enhance natural dispersion.

The findings derived from the DT provide valuable insights for both real-time air quality assessments and long-term urban planning. By integrating simulation-based predictions with real-time data, decision-makers can develop adaptive strategies to optimize airflow and reduce pollutant exposure in critical areas, ultimately improving urban air quality and public health.

5.5 Conclusion

In this work, a prototype for a DT of an urban environment was developed, utilizing HLBM to model airflow and pollutant dispersion. The methodology was validated in Section 5.3.2.3, where simulation results demonstrated strong agreement with experimental data from wind tunnel studies. The velocity and concentration distributions obtained from the DT accurately reflected expected pollutant dispersion patterns with the given geometry, confirming the feasibility of the approach. A key advantage of this DT framework is its adaptability to different urban areas. By leveraging OSM data, the automatic geometry generation allows for rapid and efficient adaptation to new cityscapes, making the approach highly scaleable. Additionally, the DT was dynamically fed with data from meteorological measuring stations and ran over a period of one week (November 7 to November 13, 2024), enabling the observation of general pollution distribution trends in the study area. This work serves as a proof of concept, demonstrating the potential of DTs in urban air quality monitoring. However, to further enhance the accuracy and reliability of the model, future work will involve the deployment of manually placed measuring stations throughout a city. These additional data sources will be used to fine-tune and validate the DT, improving its ability to capture real-world pollutant dispersion dynamics. Furthermore, while the use of a WF improves near-wall flow resolution and enhances simulation accuracy, our results indicate that it is not strictly necessary to observe the general concentration distribution. Nevertheless, future efforts will focus on optimizing WF implementations to enhance computational

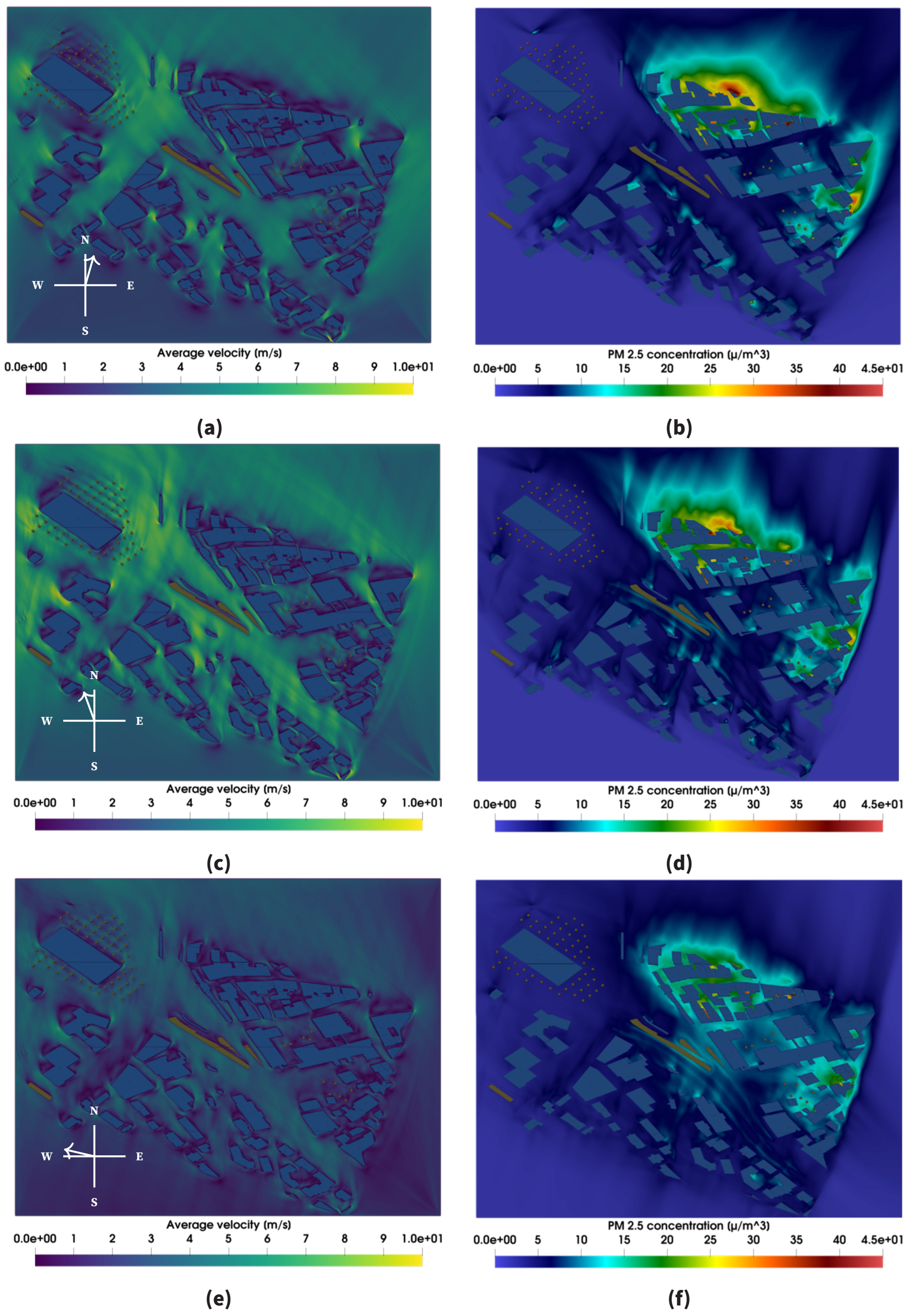

N
W
E
S
Average velocity (m/s)
0.0e+00 1 2 3 4 5 6 7 8 9 1.0e+01
PM 2.5 concentration (μ/m^3)
0.0e+00 5 10 15 20 25 30 35 40 4.5e+01
(a)
(b)
(c)
(d)
(e)
(f)

Figure 5.17: *Results of the DT from November 7 to November 13, 2024, with hourly updates. The left column (a, c and e) shows the velocity distribution, while the right column (b, d and f) presents the corresponding pollution concentration.(a) and (b) show the averaged results after running the simulation for the first day; (c) and (d) show the results on November 9th, 2024; and (e) and (f) show the results on November 13th, 2024.*

efficiency while maintaining accuracy. The WF will be further refined and used in conjunction with the DT to achieve a more precise representation of urban flow conditions. This study lays the foundation for a fully operational DT that can be continuously updated, validated, and refined using real-world sensor networks. In combination with high-performance numerical modeling, the DT framework will provide an essential tool for urban planners and environmental policymakers, supporting data-driven decisions aimed at improving air quality and public health in cities.

6 Neural Network-Based Digital Twin for a Chamber Filter Press

This chapter is based on the published article

Teutscher, D. et al. "Predicting Filter Medium Performances in Chamber Filter Presses with Digital Twins Using Neural Network Technologies." In: Applied Sciences *15.9 (2025).* ISSN*: 2076-3417.* DOI*: 10.3390/app15094933.*

The abstract has been omitted, and formatting has been adjusted to fit the thesis style. The main content remains unchanged.

6.1 Introduction

Filter presses play a critical role in numerous industries, including wastewater treatment, pharmaceuticals, and mining, by efficiently separating solids from liquids. A chamber filter press is a specific type of filter press that operates by pumping slurry into a series of recessed chambers lined with filter cloth. As pressure builds, liquid passes through the cloth and exits the system, while solids are retained to form filter cakes. This design enables high solid–liquid separation efficiency and is widely used in industrial applications due to its robustness and reliability. In the mining sector, where vast quantities of ore are processed daily and water consumption is significant [106], filter presses are indispensable for reducing fluid content in the separated material [107]. This reduction not only minimizes water usage but also mitigates the risks associated with sludge storage, such as dam failures [108]. As mining operations can have a big impact on the environment [109], it faces increasing environmental regulations and demands for sustainability. The optimization of filtration processes through innovations such as augmented reality (AR) and machine learning (ML) presents a significant opportunity to enhance automation, monitoring, and operational efficiency. Machine learning has been widely recognized for its role in predictive analytics and adaptive control in filtration systems [110, 111]. In parallel, recent work has demonstrated the potential of AR to support real-time process monitoring, operator training, and decision-making in industrial environments [112–114], suggesting further possibilities for its integration into filtration processes. By embedding predictive capabilities within filter press systems, operators can achieve higher precision in process control, leading to greater efficiency and better quality outputs. This idea has been researched through different approaches. For example, Landman et al. [115] used the theory of compressive rheology to predict the filtration time and maximize suspension throughput. Other approaches utilize neural networks (NNs) to predict metrics such as filtered volume [116], flow rate, and turbidity [117], or to predict and optimize particle counts [118]. NNs are particularly suited for modelling complex tasks due to their capability as universal function approximators, allowing them to capture intricate, non-linear relationships within diverse datasets [119]. Their adaptability and effectiveness have been demonstrated across various domains, including engineering, medical, and industrial applications, where they are used for tasks such as pattern recognition, classification, and prediction. Furthermore, recent works highlight their effectiveness in time-series forecasting and industrial prediction problems, showcasing their flexibility in real-world scenarios [120]. Another possibility is the application of computational fluid dynamics (CFD). For instance, Spielman et al. [121] employed a numerical model based on the Brinkman equation to predict pressure drops and filtration efficiency. Highly performant simulation software based on the lattice Boltzmann method (LBM) could also be utilized for this purpose [122]. However, the setup required for such simulations can be very challenging, covering aspects such as sedimentation [123–125], behaviour during the filling phase, and actual filtering processes aided by porous areas [126, 127]. Additionally, while LBM are significantly faster than other simulation approaches [128, 129], the results are still not available in real time.

Traditional filter press operations are highly dependent on manual monitoring and adjustments, which often result in inefficiencies, human errors, and increased downtime. Predicting performance parameters such as pressure and flow rates in real time is challenging, as these variables are influenced by fluctuating operating conditions and the state of the filter medium. The condition of the filter medium, particularly its level of clogging and the number of operational cycles it has undergone, significantly affects performance [130–134]. Filter press dynamics are inherently complex due to the interplay of multiple non-linear variables, including pressure, flow rate, and cake resistance, which evolve over time. In addition, fluctuations in components, such as membrane pumps, introduce noise, further complicating real-time predictive analysis. Existing data acquisition systems often lack the ability to seamlessly interface with advanced ML algorithms, limiting operational flexibility and optimization potential. According to the work of McCoy et al. [111], one problem in using ML in the mining sector is the missing amount of data to train a model. Addressing these challenges requires the integration of real-time data analytics with robust ML capabilities to enable smarter and more adaptive process control.

To address these challenges, this work proposes a machine learning-based predictive framework embedded within a digital twin (DT) architecture, a concept that refers to a virtual representation of a physical system, enabling real-time monitoring, analysis, and optimization of its operations through the integration of real-time data and predictive models [135]. The DT is designed to enable real-time estimation of key process variables, specifically pressure and flow rate, while also tracking the condition of the filter medium over successive cycles. By integrating both historical and live data, the approach mitigates the data scarcity issue and enables more robust, adaptive control strategies. The predictive model reduces reliance on manual intervention, enhances process stability, and helps determine the optimal point for filter cloth replacement, improving long-term efficiency and sustainability. By addressing both real-time and historical data, the model can assist operators in determining optimal filter medium utilization, thereby enhancing efficiency and sustainability. Key performance metrics, including root mean square error (RMSE) and mean square error (MSE), are used to evaluate the accuracy of the model in training and validation datasets. The contributions of this work can be summarized as follows. First, a machine learning-based digital twin framework is developed for a chamber filter press, enabling accurate prediction of key operational parameters such as pressure and flow rate across varying experimental configurations. Second, the proposed framework includes predictive modelling of the filter medium efficiency, allowing for proactive maintenance planning and supporting long-term resource sustainability. Finally, the architecture supports seamless data exchange and continuous model updates, ensuring adaptability to evolving operational conditions. In the remainder of the paper, first the methodology is described in Section 5.2, which contains the experimental setup, parameter selection, and the architecture of the DT, as well as the NN model. Lastly, the results are discussed in Section 5.4 and conclusions are drawn in Section 5.5.

6.2 Methodology

6.2.1 Experimental Setup of the Chamber Filter Press

The chamber filter press used in this study has a plate size of 300 mm, a compromise between typical small-scale test presses (150 mm plate size) and larger versions (up to 1000 mm). This size is compatible with existing infrastructure, requires manageable suspension quantities, and is portable. The press is equipped with a membrane pump and a manually operated hydraulic cylinder, common configurations in industry, ensuring transferability of results. The setup includes a central inlet and outlets in each corner of the plates, maintains uniform conditions across the press, and aligns with industrial applications, such as automotive paint-sludge treatment and marine scrubber systems. This design ensures leak-tightness and consistent filtrate flow, enhancing reproducibility. Air blowing for filter cake removal is controlled through adjustable valves, allowing for variable air flow. Figure 6.1 shows the experimental setup next to the filter chamber with the filter cloth.

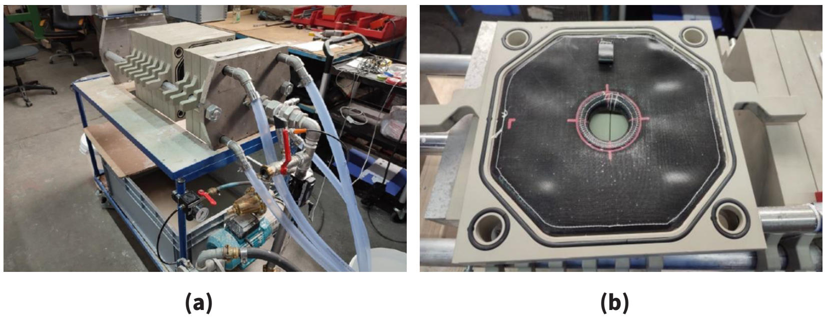

(a) (b)

Figure 6.1: *The experimental setup of the chamber filter press, where (a) shows the filter press in its entirety and (b) shows a filter chamber with the filter cloth.*

In order to ensure consistency and reproducible results that can be used to train a model, a suspension consisting of a mixture of water and perlite was used as the test material.

The filter cloth used in the experiments had a throughput capacity of $5\ \frac{l}{dm^2 \cdot min}$. During each experiment, filtration was performed under controlled conditions to evaluate key parameters such as filtrate flow rate, pressure, filter cake formation, and overall filtration performance. The filtration process continued until a specified end point, such as a defined pressure or filtrate volume. Air blowing was then applied to remove the filter cake, with the air flow rate adjusted via the system's valves.

6.2.2 Parameter Selection

Effective training of an NN for a chamber filter press requires selecting input and output variables that capture the intricate dynamics of the filtration process. The flow rate of the filtrate and the operating pressure were chosen as the primary output variables because they are key indicators of the filtration performance. As filtration progresses, the pressure gradually increases, while the flow rate decreases, eventually approaching zero. This decrease in flow, combined with an increase in pressure, signals that the filter chambers are filled with accumulated solids, indicating the end of the filtration cycle. Accurately predicting these variables allows one to monitor process efficiency in real time.

The input variables selected reflect various factors that influence the dynamics of filtration. The number of filter chambers directly impacts the filtration capacity, with a larger number of chambers enabling higher throughput. Filtration time captures the time-dependent evolution of flow and pressure, which is critical to understanding the progression of the cycle. The concentration of solids in the suspension plays a crucial role, as higher concentrations lead to a faster accumulation of solids within the chambers, affecting pressure dynamics and accelerating clogging. Another important input is the cycle count of the filter cloths, as repeated use degrades their performance, reducing the filtration efficiency over time. In addition, the maximum operating pressure sets the upper limit of the system, influencing the pressure and flow profiles throughout the process.

To enhance the generalization capability of the model, we constructed a training dataset to cover a wide range of operational conditions, including variations in chamber configurations, solid concentrations, and filter cloth usage. This comprehensive approach ensures that the model can accurately predict filtration outcomes across diverse scenarios, supporting effective, data-driven process control and optimization.

6.2.3 Digital Twin Architecture

The concept of the DT for the chamber filter press revolves around a continuously improving NN model that is updated with new training data as the filter press operates. This dynamic updating process ensures that the model predictions become increasingly reliable and accurate over time. As illustrated in Figure 6.2, the operator initiates the process by setting up a new experiment with the static parameters identified, including the number of filter chambers, the filter cloth cycle, the maximum operating pressure, and the suspension concentration. These parameters are sent simultaneously to the database and the NN model. The model uses this input to predict the expected course of pressure and flow rate over time, providing the operator with an estimated filtration time and an efficiency forecast for the process. This predictive insight helps in planning and optimizing the filtration cycle. During the filtration process, sensors installed on the filter press continuously monitor and record real-time data, specifically flow rate and pressure. This data is transmitted to the database and linked to the corresponding experiment entry, ensuring traceability and coherence between static parameters and dynamic measurements. Once the filtration cycle is complete, the recorded data can be fed back into the NN as new training data. This iterative feedback loop enhances the model's accuracy by refining its ability to predict future filtration performance based on historical patterns and real-time observations. This continuous learning approach not only optimizes the performance of the filter press, but also supports proactive decision-making, reducing downtime and improving process outcomes through more accurate predictions and insights.

This DT concept can also be expanded by integrating AR technology. AR could enable operators and maintenance personnel to visualize the state of the filter press directly on its physical counterpart, providing a real-time overlay of critical data such as performance metrics, sensor readings, or potential error states. For example, AR could highlight issues such as wear or misalignment of the filter cloth or deviations in pressure distribution, allowing immediate troubleshooting. Furthermore, AR could dynamically visualize the operational status of the filter press, including the progress of filtration cycles and system diagnostics, making complex data more intuitive and actionable. Although this combination of DT and AR has significant potential to improve system understanding, troubleshooting, and maintenance, these aspects will not be explored further in this work. However, Figure 6.3 illustrates a preliminary demonstration, from previous work, where a three-dimensional model of the

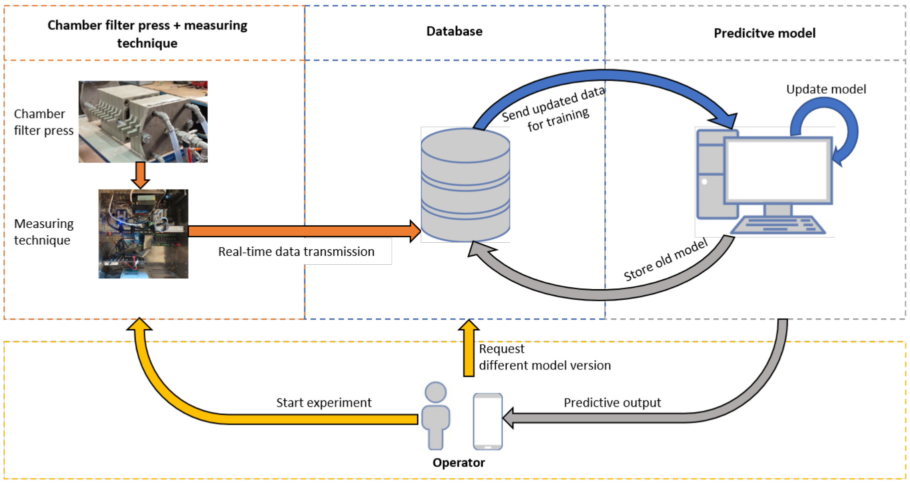


Figure 6.2: *Architecture of the DT framework for a chamber filter press, illustrating the communication flow between the filter press, real-time measuring techniques, the central database, the predictive model, and the operator.*

chamber filter press is overlaid on its real geometry, showcasing the potential for future developments in this direction [1].

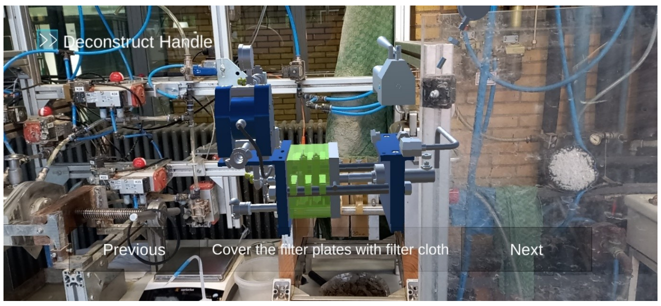


Figure 6.3: *Integration of augmented reality (AR) with the digital twin (DT) framework [1]. This figure shows a virtual 3D model of a chamber filter press overlaid onto its physical counterpart using AR. At the current stage, the AR system primarily supports visualization of the geometry and structure of the filter press. It can assist operators by highlighting specific components—such as individual filter chambers—and providing interactive maintenance instructions, for example, indicating how to access or remove parts. This lays the foundation for future functionality, such as real-time diagnostics or condition monitoring.*

6.2.4 Experiments

The experiments that serve as training and validation data for the NN are summarized in Table 6.1, with a total of 34. The selection of these experiments was based on the need to cover a wide range of operating conditions and ensure that the NN can learn patterns that generalize well to different scenarios. The configurations are designed to reflect a variety of process variables, such as concentration, filter plate number, end pressure, and filter medium cycles, which are expected to significantly influence the filtration process. By varying these parameters, we ensure that the network is exposed to a comprehensive set of inputs and can develop a robust understanding of the system's behavior.

Table 6.1: *Available training and validation data from experiments.*

Concentration [g/l]	Filter plate number	End pressure [bar]	Cycles	Frequency
6.25	2	2.0	34	1
6.25	2	4.0	32	1
6.25	2	5.0	31	1
6.25	2	6.0	30	1
6.25	2	8.0	29	1
12.50	1	10	4	1
12.50	2	10.0	2,4,5,6,7,14,23,35,36	9
12.50	2	7.0	5	1
12.50	2	8.0	6	1
12.50	2	0.2	1	1
12.50	2	0.5	10,11	2
12.50	2	0.7	12,13	2
12.50	3	10	1,2,3	3
25.00	2	10.0	24	1
25.00	2	5.0	18	1
25.00	2	6.0	19	1
25.00	2	7.0	20	1
25.00	2	8.0	21	1
25.00	2	9.0	22	1
25.00	2	10.0	23	1
25.00	3	10.0	25	1
25.00	4	10.0	26	1

6.2.5 Data Logging and Database Development

The experimental filter press setup incorporates essential hardware components powered by a 24 V DC input, including a suspension pressure sensor, a flow sensor, and a delphin data logger. The data logger, which operates at a sampling rate of 10 Hz per channel with 24-bit resolution, is suitable for capturing high-resolution data necessary for the analysis of the filtration process. It features an integrated web server and supports the open platform communications unified architecture (OPC UA) protocol, facilitating seamless data transmission and remote access (Figure 6.4). While the logger enables local data retrieval via USB using proprietary software, it does not inherently support the direct transmission of data to the model environment or provide control functionality for the active components of the filter press. Therefore, an independent control computer was introduced, equipped with a development board containing a quad-core 64-bit ARM Cortex A72 processor. This control system also includes an output module that interfaces with and manages the operation of the active elements in the filter press.

The control computer runs on a 64-bit Ubuntu Linux-based operating system and utilizes an InfluxDB time-series database for local data storage. Node-RED programming facilitates the management of local input and output (I/O) control and orchestrates the data exchange between the control computer and the data logger. Communication between the development board and the data logger is established over an Ethernet connection via the OPC UA protocol, where the data logger functions as the OPC UA server. Once data is transferred to the control system, it undergoes preprocessing, including filtering and analysis, to optimize the data storage load in the database.

The database architecture comprises two interconnected tables: one dedicated to experimental metadata and the other to measured data. The experiment table records user-defined parameters, including the experiment number, number of filter cycles, number of filter chambers, maximum operating pressure, and suspension concentration. These parameters are entered by the user via a graphical interface before initiating each experimental run. The measured data table contains time-stamped data points, such as pressure and flow rate readings, which are linked to the corresponding experiment via the experiment number as a foreign key. This relational structure allows for efficient organization and retrieval of experimental data.

Since InfluxDB utilizes time-based indexing, accurate synchronization between the control computer and the data logger is critical to maintaining data integrity. Initially, the network time protocol (NTP) is employed to synchronize the system clocks. However, given that NTP cannot achieve millisecond-level precision on the data logger, a precision time protocol (PTP) server is subsequently initiated on the control computer. This ensures precise time synchronization between the two systems, which is essential for accurate data logging and analysis.

The system architecture is designed to accommodate various data transmission strategies, depending on the operational context of the filter press. For the prototype system described, the direct transmission of measurement data from the control system to the model cloud server via LTE was selected as the preferred method. This approach facilitates real-time monitoring and data exchange with external databases for further analysis and archiving.

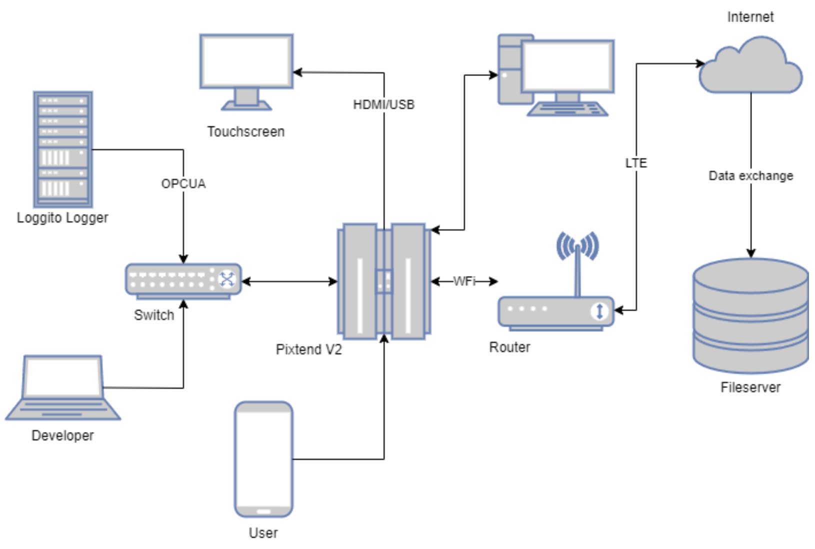


Figure 6.4: *Schematic overview of the data acquisition and communication architecture for the chamber filter press system. The setup includes a Delphin data logger connected to sensors for pressure and flow rate measurement, which communicates with a control unit via OPC UA. The control system, represented by a PiXtend V2 board, interfaces with peripheral devices such as a touchscreen and user input terminals through HDMI/USB. Data transfer and remote monitoring are facilitated through a router, enabling Wi-Fi connectivity and LTE-based transmission to a file server for storage and further analysis. Developers and operators can access the system remotely or locally for control and data export.*

6.2.6 Neural Network Model

6.2.6.1 Selection of Network Type

For the model, two NN architectures were considered, the recurrent neural network (RNN) and the feed forward neural network (FFNN), the architecture of which is shown in Figure 6.5. FFNNs are simple and effective for static data [136]; however, they lack the ability to model temporal dependencies and treat each input independently. RNNs, on the other hand, are explicitly designed to handle sequential data by maintaining an internal state that allows them to retain information from previous time steps [137]. This property makes RNNs particularly well-suited for time-series tasks, such as predicting the pressure and flow rate measurements in a filter press system. In this work, the RNN is implemented using long short-term memory (LSTM) units, which are effective at learning long-term dependencies and mitigating issues such as vanishing gradients, which can occur in standard RNNs. Its downside is that with enough accumulated data, it needs significantly more computational power to train. In this work, both the FFNN and RNN were implemented in order to compare them against each other.

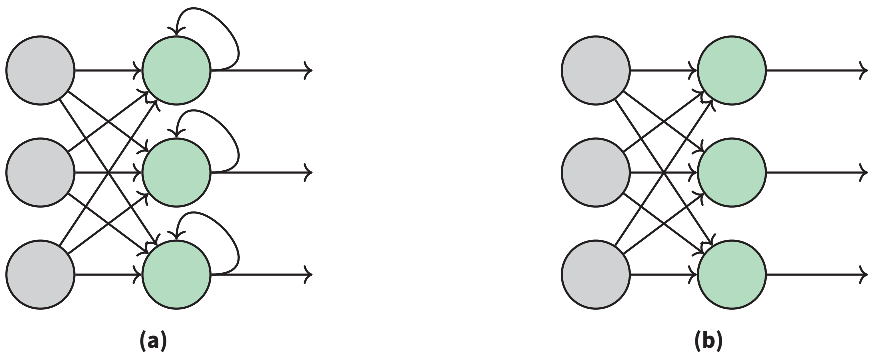


Figure 6.5: *Comparison of RNN (a) and FFNN (b). Gray nodes represent input neurons, and green nodes represent hidden neurons in both architectures. In (a), the RNN includes feedback loops, represented by the arrows looping back from the hidden neurons to themselves, enabling the processing of sequential data and temporal dependencies. The arrows pointing to the right indicate information flow to the output. In (b), the FFNN consists of direct connections between neurons, without feedback loops, illustrating a simpler network structure designed for static data processing.*

The selection of these two architectures was made to evaluate the trade-off between FFNNs' simplicity and efficiency for static data, and the temporal modelling capabilities of LSTM-based RNNs. Other architectures, such as gated recurrent unit (GRU) or convolutional neural networks (CNNs), were considered but not selected for this work. GRUs, while simpler than LSTMs, do not capture long-term dependencies as effectively, which can limit their accuracy in applications requiring extended temporal memory. However, their simpler structure, with fewer gates, results in faster training times and reduced computational overhead, making them attractive for scenarios where model efficiency and real-time responsiveness are critical [138]. Despite these advantages, in this work, LSTMs were preferred due to their superior ability to retain information over longer sequences, which is essential for accurately modelling the temporal dynamics of filter press operations. CNNs, which excel in tasks like spatial pattern recognition, are less suited for time-series forecasting tasks where sequential relationships are paramount.

6.2.6.2 Model Architectures

The FFNN consists of an input layer that accepts a normalized feature vector with five input variables. The model architecture includes two fully connected hidden layers with 64 and 32 neurons, respectively. Each hidden layer utilizes the rectified linear unit (ReLU) activation function to introduce non-linearity and enhance the model's capability to learn complex relationships. The output layer comprises a single neuron with a linear activation function, suitable for predicting continuous target variables. The FFNN is optimized using the Adam optimizer with a learning rate of 0.001.

The RNN is configured using an LSTM [139] layer to manage sequential data. The input layer receives sequences of length 10, each consisting of five features per time step. The LSTM layer contains 64 hidden units, enabling the model to capture temporal dependencies effectively. Following the LSTM layer, a fully connected output layer with a single neuron provides the final prediction, using a linear activation function for regression. Similar to the FFNN, the RNN is trained using the Adam optimizer with a learning rate of 0.001. To facilitate temporal learning, the input data is prepared by generating sequences, allowing the network to leverage internal states for improved prediction accuracy. Data normalization and sequence preparation are essential preprocessing steps that enhance the performance of both models.

6.2.6.3 Data Preparation

Before training the FFNN or RNN, the input data must undergo preprocessing [140]. The inputs are normalized for both networks using the following standardization formula:

$$x_{\text{scaled}} = \frac{x - \mu}{\sigma}, \tag{6.1}$$

where μ represents the mean of the input variables and σ denotes their standard deviation.

Although this normalization step is sufficient for training the FFNN, the RNN requires an additional preparation step: sequencing the input variables. This step structures the data into temporal sequences, enabling the RNN to capture patterns over time and develop an internal representation of the expected behavior of the experiments. This sequencing enhances the RNN's ability to model temporal dependencies and improves predictive performance.

6.2.6.4 Model Evaluation

The performance of the models was analyzed using key metrics such as the MSE, the mean absolute error (MAE), and the coefficient of determination R^2. These metrics provide complementary insights into the performance of the model. MSE quantifies the average squared differences between true and predicted values, penalizing larger errors more heavily than smaller ones. MAE measures the average absolute difference between true and predicted values, treating all deviations equally, providing a more intuitive interpretation of the average error magnitude compared to MSE. The coefficient of determination R^2 explains the proportion of variance in the true values captured by the model. The mathematical formulations of these error metrics are provided in Equations 6.2-6.4:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \tag{6.2}$$

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|, \tag{6.3}$$

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}, \tag{6.4}$$

where n is the total number of data points, y_i represents the true values, and $\hat{y}_i$ denotes the predicted values. For R^2, $\bar{y}$ is the mean of the true values.

These metrics were tracked over the training epochs to evaluate the convergence of the model and to detect potential issues such as overfitting, where the model performs well on training data but poorly on validation data, or underfitting, where the model fails to capture the underlying data patterns.

Figure 6.6 illustrates the training and validation performance of the FFNN and RNN in predicting both the pressure and the flow rate. As shown in Figure 6.6 (b), the RNN model for pressure prediction demonstrates superior convergence and generalization compared to the FFNN model (Figure 6.6 (a)). The RNN achieves training and validation MSEs of 0.0064 and 0.0093, respectively. Furthermore, MAE stabilizes at 0.04, and the R^2 score reaches 0.99, indicating strong predictive performance. In contrast, FFNN achieves a training MSE of 0.01 and a validation MSE of 0.0108, with a final R^2 score of 0.98. Although the FFNN demonstrates reasonable accuracy, the RNN consistently achieves lower errors and faster convergence compared to the FFNN. For flow rate prediction, the FFNN outperforms the RNN in terms of error metrics, achieving training and validation MSEs of 0.031 and 0.034, respectively, along with an MAE of 0.0823 and an R^2 score of 0.967. In contrast, the RNN converges to training and validation MSEs of 0.049 and 0.052, with a final MAE of 0.115 and an R^2 score of 0.9485. However, Figure 6.6 (c) reveals that the validation MAE of FFNN exhibits spikes caused by fluctuations in the validation MSE, which increases while the training MSE decreases. This instability suggests overfitting and poorer generalization. In contrast, the RNN model (Figure 6.6 (d)) achieves a final test MSE of 0.049, closely aligned with the training MSE, and an MAE of 0.1156. Unlike the FFNN, the RNN does not exhibit significant fluctuations in validation metrics, indicating better generalization and robustness. Overall, the evaluation suggests that the RNN model is better suited for this task, as it demonstrates superior generalization and stability across both prediction tasks.

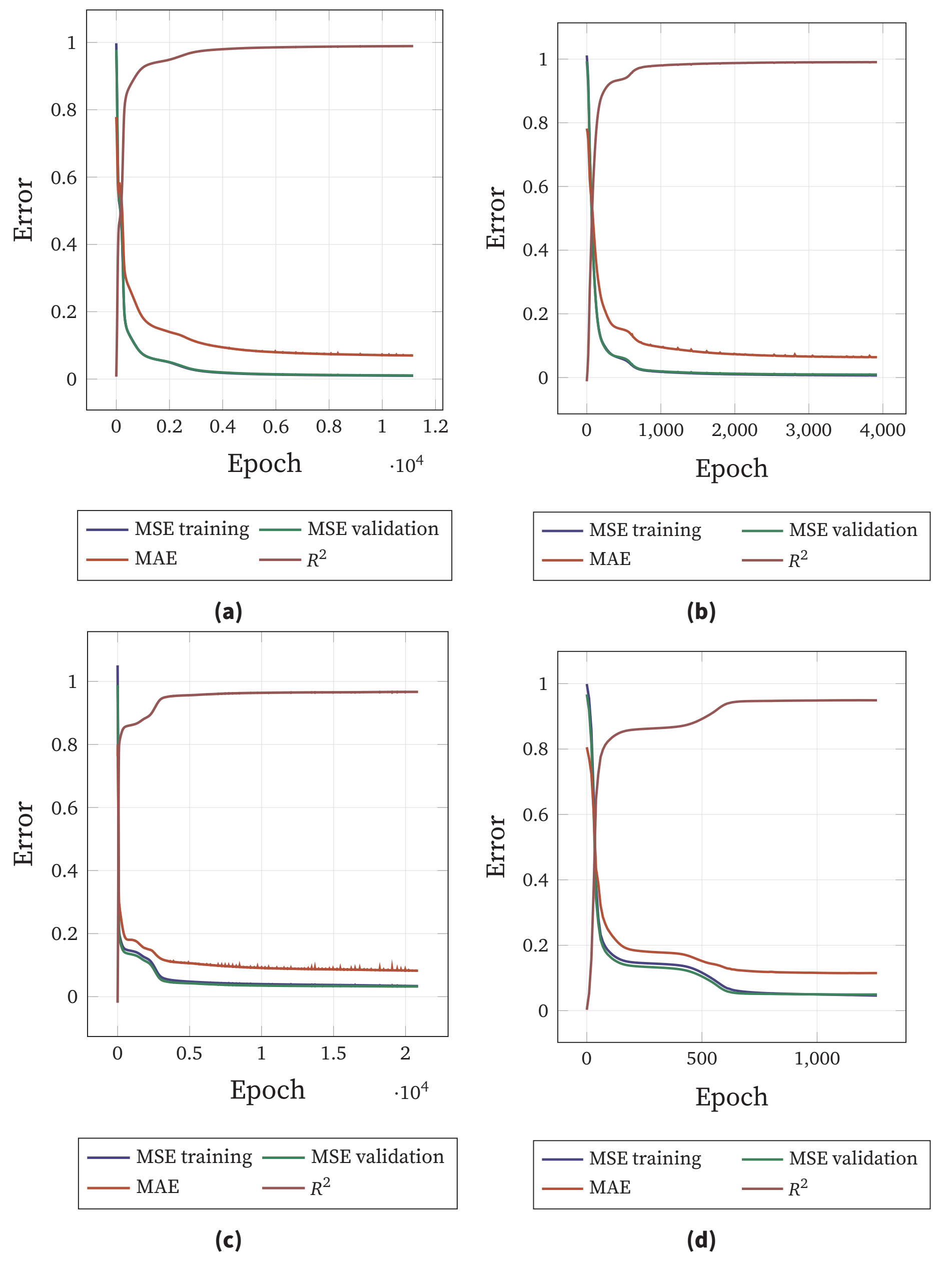


Figure 6.6: *Error metrics MSE, MAE, and R2R2 for training and validation data during training of pressure and flow models. Figure (a) and (b) represent FFNN and RNN models for pressure prediction, respectively, while (c) and (d) depict FFNN and RNN models for flow prediction*

6.3 Results

To evaluate the model, several key points must be addressed. In Figure 6.7, the true pressure values of an experiment are shown, revealing significant fluctuations in the pressure trend Figure 6.7 (a). To fairly assess the prediction error, we first calculated a moving average (MA) and standard deviation (STD) for the experiments, as described by the following equations:

$$\mathrm{MA}(t) = \frac{1}{n} \sum_{i=t-n+1}^{t} x_i, \tag{6.5}$$

$$\mathrm{STD} = \sqrt{\frac{1}{n} \sum_{i=t-n+1}^{t} (x_i - \mathrm{MA}(t))^2}, \tag{6.6}$$

where n is the size of the averaging window, x_i are the data points, and n is the number of data points.

Next, a 90% Confidence Interval (CI90%) is defined using the calculated STD:

$$\mathrm{CI90} = \mathrm{MA} \pm z \cdot \mathrm{STD}, \tag{6.7}$$

where $z = 1.645$ is the z-score corresponding to the 90% percentile. This interval is used to assess how many predictions fall within the $CI90\%$.

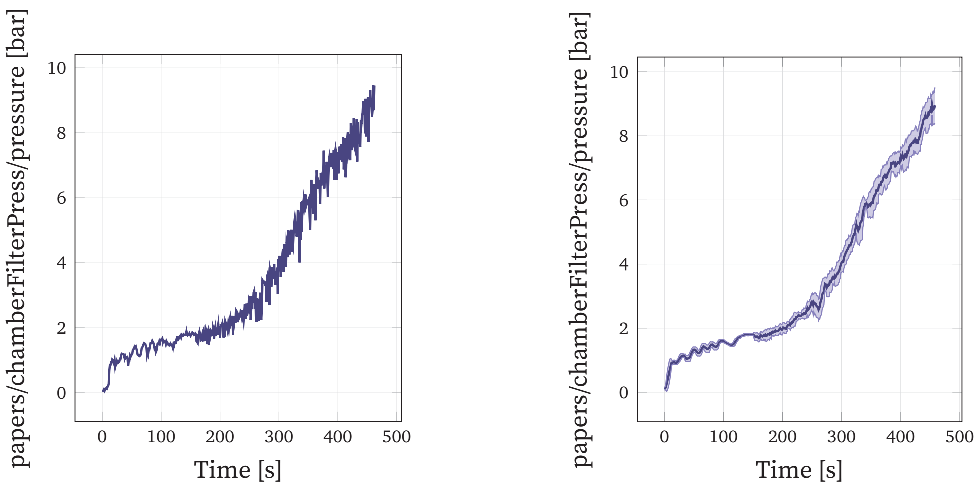
By applying these equations to the experiments, an averaged line is obtained with upper and lower bounds, which represent the fluctuations in pressure and flow rate, as shown for the pressure in Figure 6.7 (b). This was done for the flow rate in the same manner.

In addition to MSE and RMSE, the relative L^2-norm (RL2N) is also used to estimate the percentage error relative to the total magnitude of the experiments. It is defined as:

$$\mathrm{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2}, \tag{6.8}$$

$$\mathrm{RL2N} = \frac{\sqrt{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}}{\sqrt{\sum_{i=1}^{n} y_i^2}} \cdot 100, \tag{6.9}$$

where $\mathbf{y}$ represents the true values and $\hat{\mathbf{y}}$ represents the predicted values.



(a) **(b)**

Figure 6.7: *Depicted is the RAW data of a pressure trend in (a) and the MA pressure trend in (b) with CI90% bounds.*

In order to get an idea of the deviation of points in regards to the $CI90\%$ bounds the relative L^2-norm-bounds (RL2N-B) is introduced and is defined as

$$\text{RL2N-B} = \sqrt{\frac{\sum_{i=1}^{n} e^2(x_i)}{\sum_{i=1}^{n} f^2(x_i)}} \cdot 100, \tag{6.10}$$

$$e^2(x_i) = \begin{cases} 0, & \text{if } \hat{f}(x_i) \in [l(x_i), u(x_i)], \\ \min\left((\hat{f}(x_i) - l(x_i))^2, (\hat{f}(x_i) - u(x_i))^2\right), & \text{if } \hat{f}(x_i) \notin [l(x_i), u(x_i)]. \end{cases} \tag{6.11}$$

where $e^2(x_i)$ represents the squared error for the value x_i. It is zero if the predicted value $\hat{f}(x_i)$ lies within the confidence band $[l(x_i), u(x_i)]$ defined by a lower and upper bound. Otherwise, $e^2(x_i)$ is computed as the square of the minimum distance between $\hat{f}(x_i)$ and the nearest bound of the confidence interval. This formulation ensures that only deviations outside the tolerated range contribute to the total error. The RL2N-B metric thus quantifies how far and how often the prediction violates the expected confidence bounds, rather than simply measuring deviation from a reference value. This is particularly useful in scenarios where predictions are allowed to vary within a known uncertainty band and should only be penalized when they exceed those limits. The denominator $\sum_{i=1}^{n} f^2(x_i)$ serves as a normalization factor, representing the total squared magnitude of the reference values $f(x_i)$. This allows the error to be interpreted as a relative percentage of the baseline signal, enabling fair comparisons across datasets or scales while maintaining a focus on confidence-aware model performance. Last but not least the percentage of points inside the CI90% bounds (PIB) is also used for evaluation.

In this section, two experimental datasets are considered: Table 6.2, which contains experiments used for training and validation (split as 80%/20%), and Table 6.3, which contains completely unknown experiments, including one with an unknown concentration to the model.

Table 6.2: *Experiment data with diverse configurations and filter cycles. The experiment is partially known to the model since it was split into 80% training and 20% validation data.*

Experiment	Concentration [g/l]	Filter plate number	End pressure (bar)	Cycles
1	12.5	2	10	2
2	12.5	2	10	7
3	12.5	2	10	35
4	12.5	2	10	36
5	6.25	2	8	29
6	25	2	10	23
7	12.5	1	10	4
8	12.5	3	10	2
9	12.5	2	10	5
10	25	1	10	24
11	25	3	10	25
12	25	4	10	26

Table 6.3: *Experiment data with diverse configurations and filter cycles. The experiments are completely unknown to the model.*

Experiment	Concentration [g/l]	Filter plate number	End pressure (bar)	Cycles
1 val	6.25	2	10	24
2 val	12.5	2	10	30
3 val	12.5	2	10	11
4 val	12.5	2	10	10
5 val	12.5	2	10	9
6 val	12.5	3	10	6
7 val	15	2	10	7
8 val	15	2	10	8

6.3.0.1 Pressure Prediction

Partially Known Data The model demonstrated a strong reliability for pressure prediction in partially known experiments, as illustrated in Figure 6.8, which compares measured (M) against predicted (P) trends. For experiments with varying filter cycles (Exp-1 to Exp-4, Figure 6.8 (a)), the predicted pressure profiles closely matched the measured values, with minor deviations observed at higher pressures during later stages. Similarly, experiments with varying concentrations (Exp-5, Exp-3, and Exp-6, Figure 6.8 (b)) showed high accuracy, although Exp-5 exhibited less precision at maximum pressures due to the absence of training data for a concentration of 6.25 g/L. Figure 6.8 (c), (d) highlight the model's consistency across different configurations, with slight discrepancies near peak pressures. Quantitative metrics are summarized in Table 6.4. The errors for partially known experiments ranged from MSE 0.006 to 0.178, RMSE 0.103 to 0.352, and RL2N 1.7% to 9.8%. The best performance was observed in Exp-5 (RL2N 0.17%, PIB 97%), while Exp-6 had the highest error (RL2N 9.8%, PIB 64%). Overall, the model achieved an average MSE of 0.048, RMSE of 0.185, and RL2N of 5%, with 82% of the predictions falling within CI90% bounds and minor deviations for the remaining 18%. The pressure drop around 230 seconds in Exp-6 (Figure 6.8 (d)) illustrates the model's ability to follow expected trends despite limited training data for configurations with three or four plates.

Unknown Experiments For unknown experiments, the performance of the model was satisfactory, as shown in Figure 6.9. The errors for unknown experiments, detailed in Table 6.5, ranged from MSE 0.100 to 0.838, RMSE 0.317 to 0.915, RL2N 9.9% to 28.2%, and PIB 64% to 20%. Exp-6-val showed the best performance, supported by training data from a similar experiment (Exp-8). In contrast, Exp-1-val exhibited the highest errors due to limited training data at 6.25 g/L and narrow filtration cycle ranges (Figure 6.9 (b)). Despite these challenges, the model interpolated effectively in experiments with unfamiliar filtration cycles, such as Exp-4-val, achieving good results both qualitatively and quantitatively. Figure 6.9 (a) highlights deviations at the start and between 260 and 280 s, corresponding to common pressure drops during filtration. Experiments with unknown concentrations (e.g., Exp-7-val and Exp-8-val at 15 g/L, Figure 6.9 (c)) further demonstrate the generalization capabilities of the model. In general, the mean errors for unknown experiments were MSE 0.403, RMSE 0.605, RL2N 18.4%, and PIB 49.13%, with minimal deviations outside CI90%

bounds (RL2NB 8.2%).

6.3.0.2 Flow Rate Prediction

The model performed well for flow rate predictions, closely aligning with the experimental setups, as illustrated in Figure 6.10. The maximum flow rates showed expected trends, such as declines with increasing filter cycles (Figure 6.10 (a)) and concentrations (Figure 6.10 (b)). The predictions for varying filter chamber numbers (Figure 6.10 (c), (d)) also matched measured values qualitatively. Quantitative errors for partially known experiments, summarized in Table 6.4, ranged from MSE 0.339 to 2.150, RMSE 0.582 to 1.466, and RL2N 6.2% to 16.7%, with 77% to 89% of predictions within CI90% bounds. Deviations were linked to specific anomalies. For example, Exp-3 showed initial discrepancies due to clogging, which normalized post-clogging. Exp-5 achieved the best results, closely followed by Exp-12. As seen in Figure 6.10 (b), the significant deviation of Exp-5 at the start reflects challenges in initial phase predictions.

Unknown Experiments For unknown experiments, the prediction errors, detailed in Table 6.5, were higher, with averages of MSE 8.229, RMSE 2.772, RL2N 15.4%, and PIB 52.25%. The worst predictions were in Exp-2-val, where initial trends deviated strongly (Figure 6.11 (a), (b)) due to limited training data for higher flow rates (7 dm^3/min). In contrast, Exp-6-val performed best, with deviations confined to the end of the filtration process, caused by sensor limitations below 5 dm^3/min. Experiments at unknown concentrations (15 g/L, Exp-2-val and Exp-3-val) achieved acceptable results, with MSE 4.876 to 9.649, RMSE 2.208 to 3.106, RL2N 12.7% to 17.9%, and PIB 63% to 45%. Figure 6.11 (c) illustrates the model's ability to approximate overall trends despite missing knowledge for certain configurations.

Table 6.4: *Prediction errors for experiments in the validation and training sets, evaluated for both pressure and flow rate. Metrics include: MSE (mean squared error), RMSE (root mean squared error), RL2N (relative L^2 norm error), RL2N-B (relative L^2 norm error with respect to CI90 bounds), and PIB (percentage of points within bounds).*

Experiment	Pressure					Flow rate				
	MSE	**RMSE**	**RL2N** [%]	**RL2N-B** [%]	**PIB [%]**	**MSE**	**RMSE**	**RL2N** [%]	**RL2N-B** [%]	**PIB [%]**
1	0.041	0.202	4.6	0.5	96	4.955	2.226	13.4	5.3	50
2	0.011	0.103	2.6	0.1	98	3.868	1.967	9.8	2.3	64
3	0.041	0.202	4.1	1.3	74	2.150	1.466	16.7	10.6	77
4	0.036	0.190	3.7	0.9	75	0.468	0.684	8.4	3.4	69
5	0.006	0.075	1.7	0.3	97	0.339	0.582	6.2	2.6	89
6	0.178	0.421	9.8	4.7	64	1.413	1.189	10.2	3.6	62
7	0.013	0.114	2.7	0.2	98	2.905	1.704	9.2	3.3	60
8	0.045	0.211	5.8	2.4	76	3.953	1.988	9.1	4.9	91
9	0.008	0.088	2.2	0.0	100	3.934	1.983	8.5	1.6	77
10	0.038	0.196	4.1	0.5	86	0.984	0.992	9.9	2.7	66
11	0.066	0.257	6.7	2.3	75	1.082	1.040	7.0	3.1	93
12	0.124	0.352	9.2	4.8	71	0.914	0.956	6.1	3.1	98
Mean	0.048	0.185	5.0	1.8	82	2.579	1.545	9.3	3.7	74

Table 6.5: *Prediction errors for experiments completely unknown to the model, evaluated for both pressure and flow rate. Metrics include: MSE (mean squared error), RMSE (root mean squared error), RL2N (relative L^2 norm error), RL2N-B (relative L^2 norm error with respect to CI90 bounds), and PIB (percentage of points within bounds)*

Experiment	Pressure					Flow rate				
	MSE	**RMSE**	**RL2N** [%]	**RL2N-B** [%]	**PIB [%]**	**MSE**	**RMSE**	**RL2N** [%]	**RL2N-B** [%]	**PIB [%]**
1-val	0.838	0.915	28.2	15.8	20.00	10.281	3.206	18.0	6.7	50.00
2-val	0.709	0.842	24.2	11.9	46.00	12.593	3.549	20.9	7.8	35.00
3-val	0.250	0.500	16.1	5.9	47.00	13.845	3.721	20.9	5.1	55.00
4-val	0.171	0.414	13.3	4.3	54.00	1.737	1.318	6.6	1.3	84.00
5-val	0.488	0.699	22.3	13.1	52.00	6.323	2.515	13.5	3.7	37.00
6-val	0.100	0.317	9.9	2.9	64.00	6.531	2.555	12.6	5.7	49.00
7-val	0.273	0.523	16.8	7.1	54.00	4.876	2.208	12.7	2.8	63.00
8-val	0.397	0.630	16.4	4.8	56.00	9.649	3.106	17.9	5.4	45.00
Mean	0.403	0.605	18.4	8.2	49.13	8.229	2.772	15.4	4.8	52.25

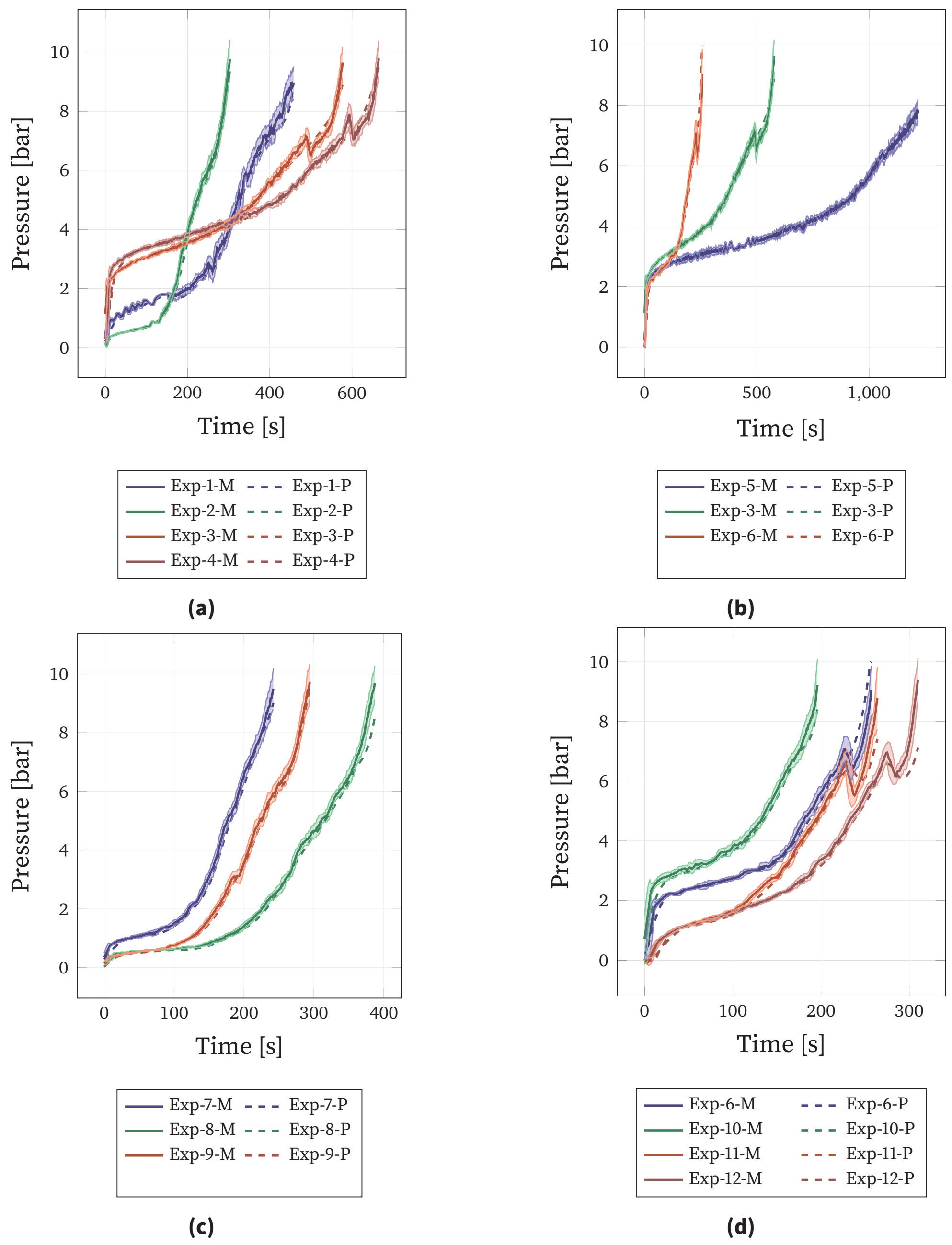
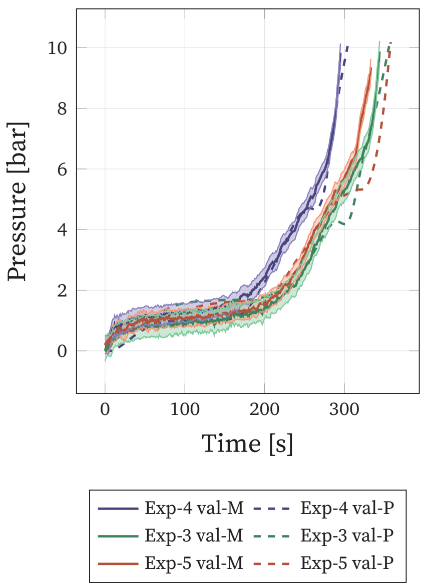
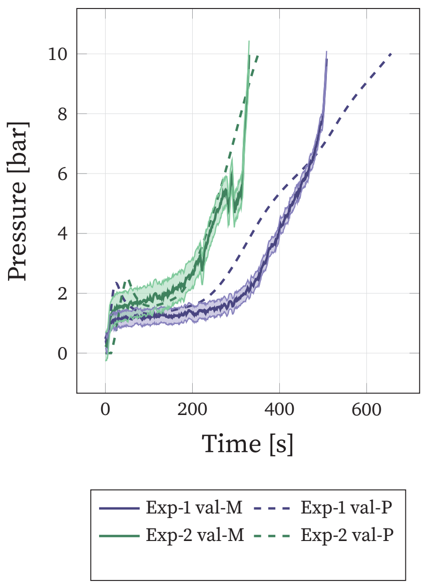


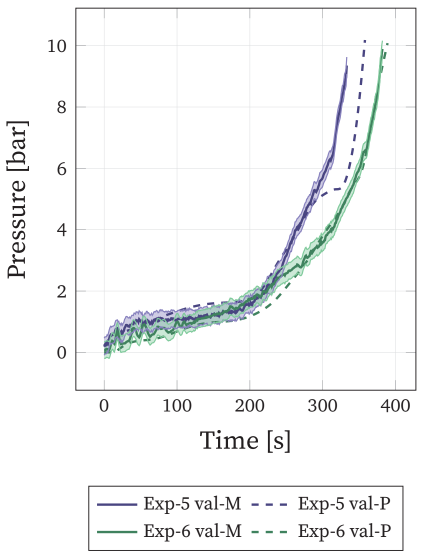
Figure 6.8: *Comparison of measured (**M**) and predicted (**P**) pressure values for the experiments from the Table 6.2. (a) shows the increase of operation time with higher cycle number. (b) shows the increase in operation time with different concentrations and similar filter cylce. (c) and (d) show the operational time with different numbers of filter chambers.*



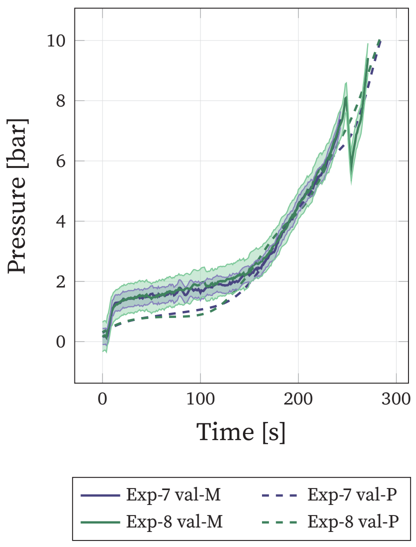
(a)



(b)

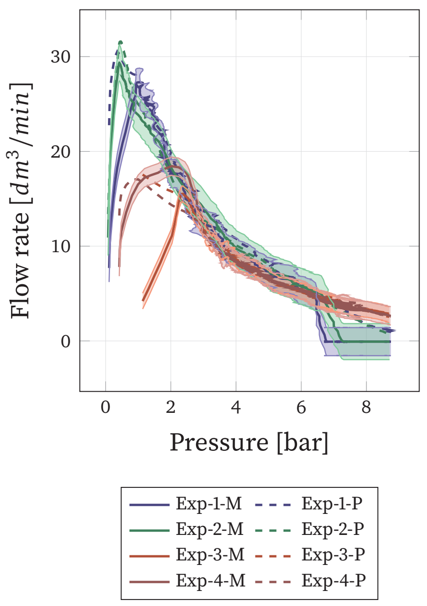


(c)

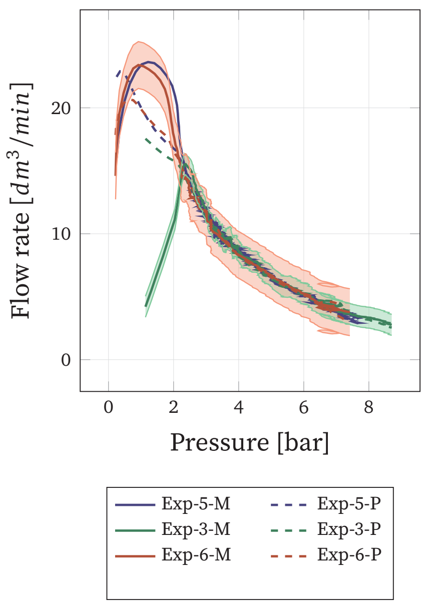


(d)

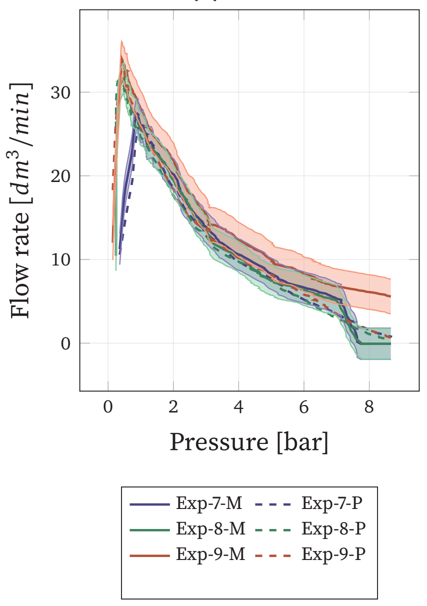
Figure 6.9: *Comparison of measured (**M**) and predicted (**P**) pressure values for the experiments listed in Table 6.3. (a) illustrates the increase in operational time with a higher number of filter cycles. (b) depicts the increase in operational time for different concentrations while maintaining similar filter cycles. (c) demonstrates the variation in operational time with differing numbers of filter chambers. (d) presents the prediction for an unknown concentration of* 15 g/L.



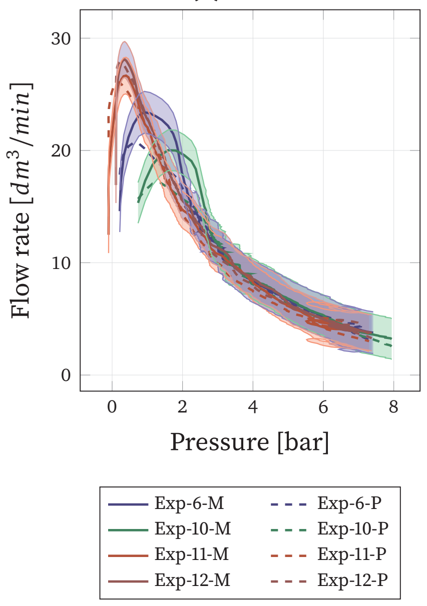
(a)



(b)

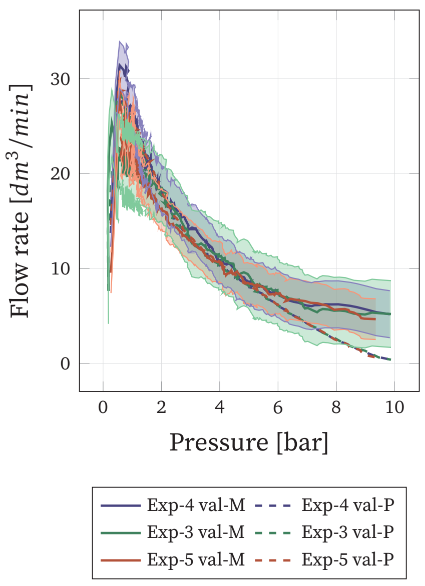


(c)

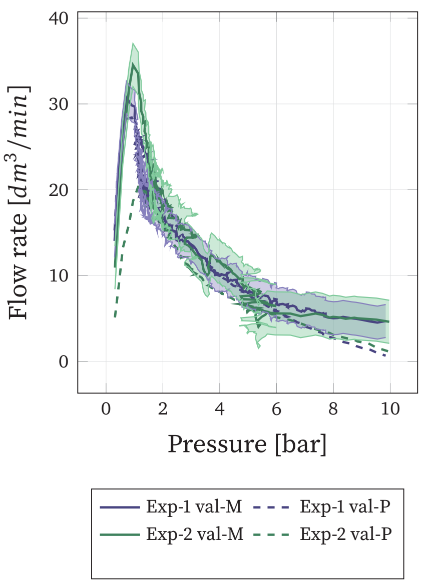


(d)

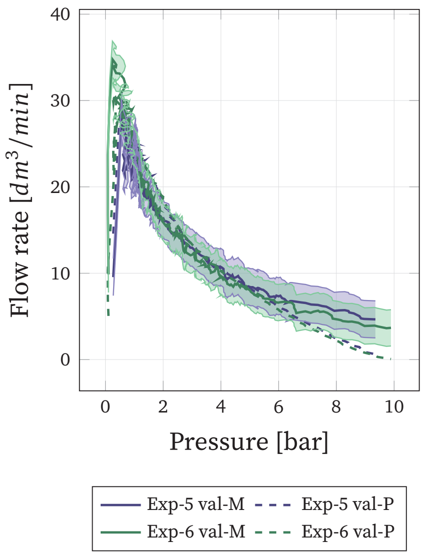
Figure 6.10: *Comparison of measured (**M**) and predicted (**P**) flow rate for the experiments from the Table 6.2. (a) shows the reduction of the flow rate with higher cycle number over the pressure. (b) shows the reduction of the flow rate with different concentrations and similar filter cylces over the pressure. (c) and (d) shows the flow rate with different numbers of filter chambers over the pressure.*



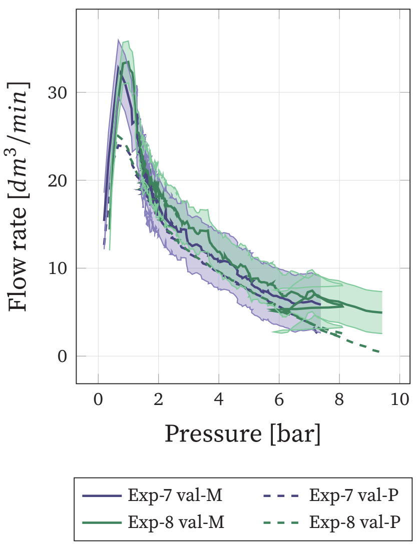
(a)



(b)



(c)



(d)

Figure 6.11: *Comparison of measured (**M**) and predicted (**P**) flow rate for the experiments listed in Table 6.3. (a) illustrates the reduction of flow rate with a higher number of filter cycles. (b) shows the reduction of the flow rate with different concentrations and similar filter cylces over the pressure. (c) shows the increase in flow rate with different number of filter chambers. (d) presents the prediction of the flow rate for an unknown concentration of* 15 g/L.

6.3.1 Current Limitations and Future Work

While the presented approach demonstrates strong performance and accurate predictions within the tested setup, several limitations remain that offer opportunities for further enhancement of the model's robustness and generalizability.

- **Material specificity:** As described in Section 6.2.1, all experiments were carried out using a single suspension, perlite, to ensure consistent results. However, other suspensions such as kieselgur exhibit significantly different behaviours due to their high porosity and fluid retention characteristics. This may affect the direct applicability of the current model to a broader range of materials.
- **Limited scalability:** The experiments were performed using a chamber filter press with a plate size of 300 mm. While the current model has shown that it can adapt to different configurations if such setups are included during training, its ability to generalize to presses of different sizes remains dependent on the diversity of the training data. Without sufficient variation, scaling the model to untested press sizes or configurations may be less reliable.
- **Hardware dependency:** The predictive accuracy of the model is influenced by the mechanical state of the system. Irregularities, such as inefficiencies in the membrane pump, leaks, or hardware degradation, can affect performance and introduce noise into the training data. If such anomalies remain undetected, they could gradually influence model performance.

To address these limitations, future work will focus on expanding the dataset to include a wider variety of suspensions, operating conditions, and press configurations. Thanks to the installed sensor setup from Section 6.2.5, the experimental data is continuously collected and stored in a central database, allowing automatic model updates as new operational scenarios arise. Incorporating anomaly detection and additional sensor types can further improve resilience to hardware-related issues. Ultimately, a more diverse and representative dataset is expected to not only improve model accuracy but also enhance its ability to generalize to unseen scenarios. In addition, we will explore whether it is possible to normalize relevant features and upscale the existing model to different filter press sizes without requiring retraining from scratch. Furthermore, the use of physics-informed neural networks incorporating domain equations such as the Darcy law will be explored, particularly for modelling porous materials such as kieselgur, where fluid–solid interactions play a more dominant role.

6.4 Conclusion

This study introduces a novel ML-based DT framework to sustainably predict key operational parameters of chamber filter presses, specifically pressure and flow rates. The framework was evaluated using two datasets: one for training and validation, and another with unknown experimental configurations.

Summary of evaluation:

- **Pressure prediction (training and validation):** Overall, MSE was 0.048, RMSE was 0.185 and RL2N was 5.0%. Deviation from the CI90% bounds was 1.8%.
- **Flow rate prediction (training and validation):** Overall, MSE was 2.579, RMSE was 1.545 and RL2N was 9.3%. Deviation from the CI90% bounds was 3.7%.
- **Pressure prediction (unknown data):** Overall MSE was 0.403, RMSE was 0.605 and RL2N was 18.4%. Deviation from the CI90% bounds was 8.2%.
- **Flow rate prediction (unknown data):** Overall MSE was 8.229, RMSE was 2.772, and RL2N was 15.4%. Deviation outside CI90% bounds was 4.8%.

The comparison of RMSE and MSE values suggests fewer significant errors and more frequent small deviations in flow rate predictions. Furthermore, the model's ability to interpolate between known configurations (e.g., 12.5 g/L and 25 g/L) to approximate unknown configurations (e.g., 15 g/L) highlights its practical utility, even in cases with limited training data. However, errors for specific configurations, such as 6.25 g/L, indicate that broader training datasets should be considered in future studies to further enhance performance in edge cases.

This DT framework, which integrates real-time data and predictive analytics, represents a significant step forward in optimizing chamber filter press operations. By accurately forecasting pressure and flow rates and estimating the lifespan of the filter medium, the system enables more efficient process control, reduces downtime, enhances resource utilization, and supports sustainable practices. The model can be adopted by operators in industrial filtration settings such as mining, wastewater treatment, and pharmaceuticals, where data-driven optimization is essential. It is particularly suited for applications involving repetitive, measurable processes that benefit from predictive maintenance and efficiency improvements. Future work will focus on expanding the diversity of operational scenarios, improving prediction accuracy for sparsely trained configurations, and further refining the model's adaptability to evolving industrial conditions.

7 Conclusion and Outlook

7.1 Conclusion

In this work, a comprehensive DT framework tailored for process engineering is presented. The framework includes a robust, modular visualization system, along with a data-driven neural network-based DT and a physics-driven DT utilizing CFD. The data-driven component introduces a general setup for collecting measurement data and storing it in a cloud database, which updates and evolves the neural network-based DT while interfacing between the DT and sensors. This approach demonstrated that accurate predictions can be achieved even with sparse data. The physics-driven DT was developed using the LBM based open-source software OpenLB. Due to its high performance on GPUs, LBM can be used to create an updatable DT model that provides results in a reasonable time. Preprocessing steps were required and implemented in OpenLB. This contribution is summarized in the following with respect to the invidual components.

Data preprocessing In Chapter 2, OpenLB was enhanced with preprocessing classes for data before the simulation begins. This enhancement allows for dynamic adjustment of simulation parameters through user interactions or automated updates from third-party software, which is crucial for real-time applications. For instance, wind speed, direction, and pollution concentration can be adjusted during run time, as demonstrated in Chapter 5. This capability is also essential for third-party interface implementation, as it requires a method to set dynamics, achievable through the XML tuple concept. Another critical preprocessing step is geometry creation, especially in large-scale simulations such as urban modeling. A parser for OSM was developed to facilitate reading and modifying OSM files. This not only addresses the challenge of assigning material numbers to objects, but also allows for adding or removing objects by manipulating the file. The latter feature is not yet implemented and is further discussed in the outlook section.

Visualization framework As a modular visualization extension for DT, either physically or data-driven, a framework was developed in Chapter 3. It enables the visualization of CFD data, as demonstrated in Chapter 4, and supports flow visualization in AR. The results presented highlight that the with the framework developed application effectively visualizes simulation data in an intuitive and interactive manner. Users can explore simulations from different angles by default in AR, zoom in to examine details, remove obstructing 3D data using a cutting plane, and utilize a color legend to interpret velocity distributions within the simulation. This application proves valuable for providing insight into complex processes that are typically invisible. In educational contexts, where theoretical concepts often lack visual representation, this tool enhances understanding by bridging theory and simulation. In addition, engineers benefit from an alternative perspective on simulations and customers can gain a clearer understanding of potential design challenges.

Virtual CFD lab on mobile devices In Chapter 4, the capabilities of the visualization framework of Chapter 3 were integrated with an adjustable LBM simulation to develop the mobile application *paint2sim*, which serves as a DT for simulations of 2D fluid flow. This application allows users to scan a hand-drawn simulation domain using a phone camera, adjust boundary conditions, and run simulations. Due to its high performance, the simulation results are generated just-in-time. The application was validated qualitatively and quantitatively against the benchmark for flow around a cylinder by Schäfer and Turek *et al.* [67]. The flow patterns and characteristics produced by *paint2sim* aligned well with the established findings. For quantitative validation, the drag and lift coefficients of the cylinder simulation were compared. Although the drag coefficients exhibited minor discrepancies of approximately $\sim 10\%$, the lift coefficients showed stronger deviations. These variations are likely due to the shape approximation of the hand-drawn cylinder and slight inaccuracies in the scanned image. This application demonstrates a physical driven DT for 2D fluid flow simulations and serves as a proof-of-concept for integrating the visualization framework with live CFD simulations.

Development of a physical driven DT As in Chapter 4 in Chapter 5, a LBM-based model for the DT is explored. The main difference is that, instead of using image and user data as input, data from measuring stations and geometric data of an urban environment are extracted, using the preprocessing detailed in Chapter 2. Here, HLBM not only accounts for solid structures like buildings, but also includes porous objects such as trees to observe their impact on pollution and wind distribution. To this end, an additional porous ADE correction term was implemented. The methodology of using HLBM with the correction term was validated through an established wind tunnel study, simulating scenarios with and without a WF for a canyon featuring a tree and without a tree. The simulation results demonstrated strong agreement with both the experimental results and the reference simulations. The DT was implemented for a section of Reutlingen a town in the south of germany, which features two measuring stations. Utilizing information from OSM, an approximation of $PM2.5$ pollution sources was carried out by evaluating the volume of observed streets, setting their height to that of the meteorological stations, and extrapolating NOx values over this volume to estimate street traffic at a given timestamp. By multiplying the median emission rate of cars with the estimated street traffic, the $PM2.5$ value was approximated. This allowed for general assumptions regarding pollution concentrations originating from both streets and buildings. By running and updating the DT for the duration of one week (November 7 to November 13, 2024), and integrating wind direction, speed, and pollution data, pollution hotspots could be identified. The findings aligned with the validation results, showing that most of the pollution concentrations were retained inside narrow canyon-like building areas, mainly due to rotating air circulation. By leveraging OSM, this approach is highly scalable and can easily be implemented for other cities. This sets the foundation for a fully operational DT that can be continuously updated, validated, and refined using real-world sensor networks. In combination with high-performance numerical modeling, the DT framework will provide an essential tool for urban planners and environmental policymakers, supporting data-driven decisions aimed at improving air quality and public health in cities. Further possible work in this area is discussed in Chapter 7.2.

Development of a data-driven DT A data-driven DT for a chamber filter press was developed in Chapter 6. This study introduced a NN-based DT framework designed to predict key operational parameters, specifically the pressure and flow rates of chamber filter presses. It consists of a setup for data exchange using sensors, data loggers, a Raspberry Pi, and network-based database cloud storage. This setup stores the experimental data, to continuously update the model, and send data for prediction to view the trends of process parameters such as flow rate and pressure over time. The framework was evaluated using two distinct datasets: one for training and one for validation consisting of unknown experiments and partially unknown configurations that achieved a relative L^2 norm error of 5% for pressure and 9.3% for the prediction of the flow rate on partially known data. For completely unknown data, the relative errors were 18.4% and 15.4%, respectively. Qualitative analysis showed strong alignment between predicted and measured data, with deviations around a confidence band of 8.2% for pressure and 4.8% for flow rate predictions. This data-driven DT framework, which integrates real-time data with predictive analytics, represents a significant advancement in the optimization of chamber filter press operations. By accurately forecasting key parameters such as pressure and flow rates while accounting for concentration and filter medium aging, the system contributes to more efficient process control, reduced downtime, enhanced resource utilization, and supports sustainable practices. Furthermore, the framework has the potential to estimate the lifespan of the filter medium, improving overall operational efficiency. The concept of this DT is applicable to be effective in various process engineering applications.

The outlined subgoals from Chapter 1.6 have been successfully achieved. A modular visualization framework was developed that is integrable with both data-driven and physics-driven DTs, enabling enhanced interpretation and interactive analysis through AR-based visualization. Furthermore, a DT for CFD simulations was implemented, featuring real-time data representation and user interaction, significantly improving accessibility and understanding of complex flow phenomena. A NN-based DT was designed for real-time monitoring and predictive analytics in process engineering, demonstrating accurate predictions of key operational parameters such as pressure and flow rates. Additionally, an updatable simulation-based DT was created, integrating real-time data updates from sensors and environmental conditions to ensure dynamic adaptability. These contributions establish a comprehensive and scalable DT framework, advancing process engineering through improved visualization, predictive analytics, and real-time system adaptability.

7.2 Outlook

This work introduced a general framework for implementing a DT for specific applications in process engineering. However, there are four adaptations and optimizations that need further development:

Validation and optimization of the urban DT As shown in Chapter 5 while the general simulation approach was validated with the help of the experimental data, the DT was not yet fully validated in itself. Future work should include validation of the DT with the help of strategically placed measuring stations in order to validate and calibrate the model, since the initializing of the pollution sources are roughly approximated and need to be refined. Furthermore, it was stated that the WF was not yet able to be deployed for the large-scale simulation because it was not optimized. Although the general distribution of pollution is the same, it can be used to get more accurate results.

Development of a user interface In Chapter 2, the core concept for the implementation of a UI in OpenLB was covered. However, the external interface itself has not yet been developed. With respect to the DT of the urban area, it should also integrate the OSM API with the ability to interact with the OSM map interactively. Although manually modifying the OSM map is possible, it is cumbersome because each object consists of a list of nodes. An interactable UI allows architects and policymakers to easily adjust the maps and observe the impact of additional objects, as well as pollution and wind distributions.

Combining physical-driven with data-driven models While in this work the framework foundation for data-driven and physics-driven DTs was laid, the combination of both was not yet investigated using a PINN as a basis. This type of model integrates known physical laws, typically in the form of differential equations, directly into the neural network training process. By embedding physical constraints, such as conservation laws or governing equations, PINNs ensure that predictions remain consistent with established physics while leveraging data to refine and adjust parameters. This approach improves generalization and prediction accuracy with limited data, and allows better extrapolation to unseen scenarios. Future research should explore how a PINN-based DT could bridge the gap between purely data-driven and physics-driven models, enabling more robust, interpretable, and computationally efficient predictions for complex process engineering applications.

Integration into a unified framework Throughout this work, individual modules were developed for various DT applications, ranging from preprocessing and visualization to modeling with LBM and NNs. While these modules were designed to be compatible and reusable across different applications, they currently exist as separate components without a unified, overarching framework. Creating a single, integrated DT platform that encompasses all modules would offer significant advantages. It would streamline the development of future applications by providing a standardized interface to combine data preprocessing, real-time visualization, and model-driven simulation or prediction. In addition, a unified framework would improve maintainability, modular scalability, and user accessibility, allowing users to switch between or combine application types, such as physical simulations and machine learning models, within a common environment. Such a platform could serve as a foundation for an extensible ecosystem, where new modules can be plugged in and reused across domains, paving the way for broader adoption and easier deployment of DTs in both research and industry.

Acknowledgment

This work was funded by the Ministry of Science, Research and the Arts of the State of Baden-Württemberg (grant number 34-7811.553-4/5), the Lattice Boltzmann Research Group, the Ramboll Foundation as part of the project "TreeCFD – Advancing Urban Sustainability through High-Fidelity Microclimate Modeling of Trees", and the Invest BW project BW1_0027/01 "Filt-AR: Entwicklung einer neuen Filt-AR-Filterpresse mit wissensbasierter Regelung anhand eines digitalen Zwillings und Augmented Reality-Visualisierung".

This work was partially performed on the following computational resources:

- HoreKa supercomputer funded by the Ministry of Science, Research and the Arts Baden-Württemberg and by the Federal Ministry of Education and Research

List of Puplications

Peer-Reviewed Publications

[1] **Teutscher, D.**, Weckerle, T., Öz, Ö. F., and Krause, M. J. "Interactive Scientific Visualization of Fluid Flow Simulation Data Using AR Technology-Open-Source Library OpenVisFlow." In: *Multimodal Technologies and interaction* 6.9 (2022), p. 81. DOI: 10.3390/mti6090081.

[2] **Teutscher, D.**, Kummerländer, A., Bukreev, F., Dorn, M., and Krause, M. J. "Just-in-Time Fluid Flow Simulation on Mobile Devices Using OpenVisFlow and OpenLB." In: *Applied Sciences* 14.5 (2024). DOI: 10.3390/app14051784.

[3] **Teutscher, D.**, Bukreev, F., Kummerländer, A., Simonis, S., Bächler, P., Rezaee, A., Hermansdorfer, M., and Krause, M. J. "A digital urban twin enabling interactive pollution predictions and enhanced planning." In: *Building and Environment* 281 (2025), p. 113093. DOI: https://doi.org/10.1016/j.buildenv.2025.113093.

[4] **Teutscher, D.**, Weber-Carstanjen, T., Simonis, S., and Krause, M. J. "Predicting Filter Medium Performances in Chamber Filter Presses with Digital Twins Using Neural Network Technologies." In: *Applied Sciences* 15.9 (2025). DOI: 10.3390/app15094933.

[5] Schutera, S., Schnierle, M., Wu, M., Pertzel, T., Seybold, J., Bauer, P., **Teutscher, D.**, Raedle, M., Heß-Mohr, N., Röck, S., and Krause, M. J. "On the Potential of Augmented Reality for Mathematics Teaching with the Application cleARmaths." In: *Education Sciences* 11.8 (2021). DOI: 10.3390/educsci11080368.

[6] Bukreev, F., Kummerländer, A., Jeßberger, J., **Teutscher, D.**, Simonis, S., Bothe, D., and Krause, M. J. “Benchmark Simulation of Laminar Reactive Micromixing Using Lattice Boltzmann Methods.” In: *AIAA Journal* (2024), pp. 1–10. DOI: 10.2514/1.J064234.

Other Publications

[7] Kummerländer, A., Bingert, T., Bukreev, F., Czelusniak, L. E., Dapelo, D., Englert, S., Hafen, N., Heinzelmann, M., ito, S., Jeßberger, J., Kaiser, F., Kummer, E., Kusumaatmaja, H., Marquardt, J. E., Rennick, M., Pertzel, T., Prinz, F., Sadric, M., Schecher, M., Simonis, S., Sitter, P., **Teutscher, D.**, Zhong, M., and Krause, M. J. *OpenLB User Guide 1.7*. Version 1.7r0. 2024. DOI: 10.5281/zenodo.13293033. URL: https://doi.org/10.5281/zenodo.13293033.

Conference Talks

[8] **Teutscher, D.**, Bukreev, F., and Krause, M. *OpenLBar: A Visualization Tool for Complex Fluid Flow*. Exhibition Stand at FILTECH 2022. Showcased as part of the Teaching4Future project. 2022.

[9] **Teutscher, D.** and Krause, M. J. *Teaching4Future Symposium: Interactive Visualization Tools for STEM Education*. Symposium organized at Karlsruhe Institute of Technology (KIT). Included demonstrations of the OpenLBar App, ARCTIC App, paint2sim App, and cleARmath App. Lattice Boltzmann Research Group (LBRG) & Forschungszentrum CeMOS, 2023.

[10] **Teutscher, D.**, Kummerländer, A., and Krause, M. J. “Simplifying Fluid Flow Simulations with paint2sim: A Just-in-Time Visualization Tool Using Lattice Boltzmann Method.” In: *DSFD 2023 – 32th International Conference on Discrete Simulation of Fluid Dynamics*. Albuquerque, USA, 2023.

[11] **Teutscher, D.**, Bukreev, F., Simonis, S., and Krause, M. “Homogenized Lattice Boltzmann Methods for City-Scale Flow Simulations in Digital Twins.” In: *9th European Congress on Computational Methods in Applied Sciences and Engineering ECCOMAS2024*. Lisbonn, Portugal, 2024.

Software Releases

[12] Kummerländer, A., Avis, S., Kusumaatmaja, H., F., B., Dapelo, D., Großmann, S., Hafen, N., Holeksa, C., Husfeldt, A., Jeßberger, J., Kronberg, L., Marquardt, J., Mödl, J., Nguyen, J., Pertzel, T., Simonis, S., Springmann, L., Suntoyo, N., Teutscher, D., Zhong, M., and Krause, M. *OpenLB Release 1.5: Open Source Lattice Boltzmann Code*. Version 1.5. 2022. DOI: 10.5281/zenodo.6469606.

[13] Kummerländer, A., Avis, S., Kusumaatmaja, H., Bukreev, F., Crocoll, M., Dapelo, D., Hafen, N., Ito, S., Jeßberger, J., Marquardt, J. E., Mödl, J., Pertzel, T., Prinz, F., Raichle, F., Schecher, M., Simonis, S., **Teutscher**, **D.**, and Krause, M. J. *OpenLB Release 1.6: Open Source Lattice Boltzmann Code*. Version 1.6. 2023. DOI: 10.5281/zenodo.7773497.

[14] Kummerländer, A., Bingert, T., Bukreev, F., Czelusniak, L. E., Dapelo, D., Hafen, N., Heinzelmann, M., Ito, S., Jeßberger, J., Kusumaatmaja, H., Marquardt, J. E., Rennick, M., Pertzel, T., Prinz, F., Sadric, M., Schecher, M., Simonis, S., Sitter, P., **Teutscher**, **D.**, Zhong, M., and Krause, M. J. *OpenLB Release 1.7: Open Source Lattice Boltzmann Code*. Version 1.7.0. 2024. DOI: 10.5281/zenodo.10684609.

[15] Kummerländer, A. et al. *OpenLB Release 1.8: Open Source Lattice Boltzmann Code*. 2025. DOI: 10.5281/zenodo.15270117.

Bibliography

[16] Velasquez, N., Estevez, E., and Pesado, P. “Cloud Computing, Big Data and the industry 4.0 Reference Architectures.” In: *Journal of Computer Science and Technology* 18 (2018), e29. DOI: 10.24215/16666038.18.e29.

[17] Yadav, M., Vardhan, A., Chauhan, A. S., and Saini, S. “A Study on Creation of industry 5.0: New innovations using big data through artificial intelligence, internet of Things and next-origination technology policy.” In: *2023 iEEE international Students’ Conference on Electrical, Electronics and Computer Science (SCEECS)*. 2023, pp. 1–12. DOI: 10.1109/SCEECS57921.2023.10063069.

[18] Grieves, M. and Vickers, J. “Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems.” In: *Transdisciplinary Perspectives on Complex Systems: New Findings and Approaches*. 2017, pp. 85–113. DOI: 10.1007/978-3-319-38756-7_4.

[19] Zhang, L., Zhou, L., and Horn, B. K. “Building a right digital twin with model engineering.” In: *Journal of Manufacturing Systems* 59 (2021), pp. 151–164. DOI: https://doi.org/10.1016/j.jmsy.2021.02.009.

[20] Wright, L. and Davidson, S. “How to tell the difference between a model and a digital twin.” In: *Advanced Modeling and Simulation in Engineering Sciences* 7 (2020). DOI: 10.1186/s40323-020-00147-4.

[21] Sjarov, M., Lechler, T., Fuchs, J., Brossog, M., Selmaier, A., Faltus, F., Donhauser, T., and Franke, J. “The Digital Twin Concept in industry – A Review and Systematization.” In: *2020 25th iEEE international Conference on Emerging Technologies and Factory Automation (ETFA)*. Vol. 1. 2020, pp. 1789–1796. DOI: 10.1109/ETFA46521.2020.9212089.

[22] VanDerHorn, E. and Mahadevan, S. "Digital Twin: Generalization, characterization and implementation." In: *Decision Support Systems* 145 (2021), p. 113524. DOI: https://doi.org/10.1016/j.dss.2021.113524.

[23] Glaessgen, E. and Stargel, D. "The digital twin paradigm for future NASA and U.S. air force vehicles." In: 2012. DOI: 10.2514/6.2012-1818.

[24] Sharma, A., Kosasih, E., Zhang, J., Brintrup, A., and Calinescu, A. "Digital Twins: State of the art theory and practice, challenges, and open research questions." In: *Journal of industrial information integration* 30 (2022), p. 100383. DOI: https://doi.org/10.1016/j.jii.2022.100383.

[25] Stark, R., Kind, S., and Neumeyer, S. "Innovations in digital modelling for next generation manufacturing system design." In: *CiRP Annals* 66.1 (2017), pp. 169–172. DOI: https://doi.org/10.1016/j.cirp.2017.04.045.

[26] Klostermeier, R., Haag, S., and Benlian, A. "Digitale Zwillinge – Eine explorative Fallstudie zur Untersuchung von Geschäftsmodellen." In: 2019, pp. 255–269. DOI: 10.1007/978-3-658-26314-0_15.

[27] Tao, F., Cheng, J., Qi, Q., Zhang, M., Zhang, H., and Sui, F. "Digital twin-driven product design, manufacturing and service with big data." In: *The international Journal of Advanced Manufacturing Technology* 94 (2018). DOI: 10.1007/s00170-017-0233-1.

[28] Jones, D., Snider, C., Nassehi, A., Yon, J., and Hicks, B. "Characterising the Digital Twin: A systematic literature review." In: *CiRP Journal of Manufacturing Science and Technology* 29 (2020), pp. 36–52. DOI: https://doi.org/10.1016/j.cirpj.2020.02.002.

[29] Thabet, S. and Thabit, T. "CFD Simulation of the Air Flow around a Car Model (Ahmed Body)." In: *International Journal of Scientific and Research Publications (IJSRP)* 8 (2018), p. 517. DOI: 10.29322/IJSRP.8.7.2018.p7979.

[30] Fujii, K. "Progress and future prospects of CFD in aerospace—Wind tunnel and beyond." In: *Progress in Aerospace Sciences* 41.6 (2005), pp. 455–470. DOI: https://doi.org/10.1016/j.paerosci.2005.09.001.

[31] Lamba, S., Rawat, A., Jacob, J., Rawat, J., Chauhan, V., and Panchal, S. "Impact of Teaching Time on Attention and Concentration." In: *IOSR Journal of Nursing and Health Science* 3 (2014), pp. 01–04. DOI: 10.9790/1959-03410104.

[32] Noghabaei, M., Heydarian, A., Balali, V., and Han, K. "Trend Analysis on Adoption of Virtual and Augmented Reality in the Architecture, Engineering, and Construction industry." In: *Data* 5 (2020), p. 26. DOI: 10.3390/data5010026.

[33] Hussain, R., Lalande, A., Guigou, C., and Bozorg-Grayeli, A. "Contribution of Augmented Reality to Minimally invasive Computer-Assisted Cranial Base Surgery." In: *IEEE Journal of Biomedical and Health Informatics* 24.7 (2020), pp. 2093–2106. DOI: 10.1109/JBHI.2019.2954003.

[34] Iatsyshyn, A., Valeriia, K., Y.O., R., I.I., D., Andrii, I., Popov, O., Kutsan, Y., Artemchuk, V., Burov, O., and Lytvynova, S. "Application of augmented reality technologies for preparation of specialists of new technological era." In: *Proceedings of the 2nd International Workshop on Augmented Reality in Education*. Vol. 2547. 2019, pp. 181–200. DOI: https://doi.org/10.31812/123456789/3675.

[35] Dalim, C. S. C., Piumsomboon, T., Dey, A., Billinghurst, M., and Sunar, S. "TeachAR: An Interactive Augmented Reality Tool for Teaching Basic English to Non-native Children." In: *2016 IEEE International Symposium on Mixed and Augmented Reality (ISMAR-Adjunct)*. 2016, pp. 344–345. DOI: 10.1109/ISMAR-Adjunct.2016.0113.

[36] Thomas, R. G., William John, N., and Delieu, J. M. "Augmented reality for anatomical education." In: *Journal of visual communication in medicine* 33.1 (2010), pp. 6–15. DOI: 10.3109/17453050903557359.

[37] Lin, J.-R., Cao, J., Zhang, J.-P., van Treeck, C., and Frisch, J. "Visualization of indoor thermal environment on mobile devices based on augmented reality and computational fluid dynamics." In: *Automation in Construction* 103 (2019), pp. 26–40. DOI: https://doi.org/10.1016/j.autcon.2019.02.007.

[38] Zhu, Y., Fukuda, T., and Yabuki, N. "Integrating Animated Computational Fluid Dynamics into Mixed Reality for Building-Renovation Design." In: *Technologies* 8.1 (2020). DOI: 10.3390/technologies8010004.

[39] Kim, M., Yi, S., Jung, D., Park, S., and Seo, D. "Augmented-Reality Visualization of Aerodynamics Simulation in Sustainable Cloud Computing." In: *Sustainability* 10.5 (2018). DOI: 10.3390/su10051362.

[40] Sanderasagran, A. and Aziz, A. "Real-Time Computational Fluid Dynamics Flow Response Visualisation and interaction Application Based on Augmented Reality." In: *Journal of information and Communication Technology* 19 (2020), pp. 559–581. DOI: 10.32890/jict2020.19.4.5.

[41] Fukuda, T., Yokoi, K., Yabuki, N., and Motamedi, A. "An indoor thermal environment design system for renovation using augmented reality." In: *Journal of Computational Design and Engineering* 6.2 (2019), pp. 179–188. DOI: https://doi.org/10.1016/j.jcde.2018.05.007.

[42] Ham, Y. and Golparvar-Fard, M. "EPAR: Energy Performance Augmented Reality models for identification of building energy performance deviations between actual measurements and simulation results." In: *Energy and Buildings* 63 (2013), pp. 15–28. DOI: https://doi.org/10.1016/j.enbuild.2013.02.054.

[43] Solmaz, S. and Van Gerven, T. "Automated integration of extract-based CFD results with AR/VR in engineering education for practitioners." In: *Multimedia Tools and Applications* 81 (2022), pp. 14869–14891. DOI: https://doi.org/10.1007/s11042-021-10621-9.

[44] Krüger, T., Kusumaatmaja, H., Kuzmin, A., Shardt, O., Silva, G., and Viggen, E. M. "The lattice Boltzmann method." In: *Springer international Publishing* 10.978-3 (2017), pp. 4–15. DOI: https://doi.org/10.1007/978-3-319-44649-3.

[45] Chapman, S. and Cowling, T. G. *The mathematical theory of non-uniform gases: an account of the kinetic theory of viscosity, thermal conduction and diffusion in gases*. Cambridge university press, 1990. DOI: https://doi.org/10.2307/3609795.

[46] Community, B. O. *Blender - a 3D modelling and rendering package*. Blender Foundation. 2018.

[47] Haas, J. K. "A History of the Unity Game Engine." Interactive Qualifying Project. Worcester Polytechnic Institute, 2014.

[48] Epic Games. *Unreal Engine*. Version 4.22.1. 25, 2019.

[49] Berger, M. and Cristie, V. "CFD Post-processing in Unity3D." In: *Procedia Computer Science* 51 (2015). International Conference On Computational Science, ICCS 2015, pp. 2913–2922. DOI: https://doi.org/10.1016/j.procs.2015.05.476.

[50] Stam, J. "Real-time fluid dynamics for games." In: *Proceedings of the game developer conference*. Vol. 18. 2003, p. 25.

[51] Zuo, W. and Chen, Q. "Real-time or faster-than-real-time simulation of airflow in buildings." In: *Indoor Air* 19.1 (2009), p. 33. DOI: 10.1111/j.1600-0668.2008.00559.x.

[52] Minichiello, A., Armijo, D., Mukherjee, S., Caldwell, L., Kulyukin, V., Truscott, T., Elliott, J., and Bhouraskar, A. "Developing a mobile application-based particle image velocimetry tool for enhanced teaching and learning in fluid mechanics: A design-based research approach." In: *Computer Applications in Engineering Education* 29.3 (2021), pp. 517–537. DOI: https://doi.org/10.1002/cae.22290.

[53] Harwood, A. R. and Revell, A. J. "Interactive flow simulation using Tegra-powered mobile devices." In: *Advances in Engineering Software* 115 (2018), pp. 363–373. DOI: https://doi.org/10.1016/j.advengsoft.2017.10.005.

[54] Liu, Z., Chu, X., Lv, X., Liu, H., Fu, H., and Yang, G. "Accelerating Large-Scale CFD Simulations with Lattice Boltzmann Method on a 40-Million-Core Sunway Supercomputer." In: *Proceedings of the 52nd international Conference on Parallel Processing*. New York, NY, USA, 2023, pp. 797–806. DOI: 10.1145/3605573.3605605.

[55] Wenhan, Z., Hong, F., and Gong, S. "improved multi-relaxation time thermal pseudo-potential lattice Boltzmann method with multi-block grid and complete unit conversion for liquid–vapor phase transition." In: *Physics of Fluids* 35 (2023), p. 053337. DOI: 10.1063/5.0147074.

[56] Kuzmin, A., Mohamad, A., and Succi, S. "Multi-relaxation time Lattice Boltzmann Model for multiphase flows." In: *international Journal of Modern Physics C* 19 (2008). DOI: 10.1142/S0129183108012571.

[57] Li, Q., Zhou, P., and Yan, H. J. "improved thermal lattice Boltzmann model for simulation of liquid-vapor phase change." In: *Phys. Rev. E* 96 (6 2017), p. 063303. DOI: 10.1103/PhysRevE.96.063303.

[58] Dietzel, M., Ernst, M., and Sommerfeld, M. "Application of the Lattice-Boltzmann Method for Particle-laden Flows: Point-particles and Fully Resolved Particles." In: *Flow, Turbulence and Combustion* 97 (2016). DOI: 10.1007/s10494-015-9698-x.

[59] Hamila, R., Jemni, A., and Perre, P. "A lattice Boltzmann source formulation for advection and anisotropic." In: *indian Journal of Physics* 97 (2023). DOI: 10.1007/s12648-023-02667-2.

[60] Bradski, G. "The OpenCV Library." In: *Dr. Dobb's Journal of Software Tools* (2000).

[61] Krause, M. J., Kummerländer, A., Avis, S. J., Kusumaatmaja, H., Dapelo, D., Klemens, F., Gaedtke, M., Hafen, N., Mink, A., Trunk, R., Marquardt, J. E., Maier, M.-L., Haussmann, M., and Simonis, S. "OpenLB—Open source lattice Boltzmann code." In: *Computers & Mathematics with Applications* 81 (2021), pp. 258–288.

[62] Li, J. *Appendix: Chapman-Enskog Expansion in the Lattice Boltzmann Method*. 2015. DOI: 10.48550/ARXiV.1512.02599. URL: https://arxiv.org/abs/1512.02599.

[63] Guo, Z., Zheng, C., and Shi, B. "Discrete lattice effects on the forcing term in the lattice Boltzmann method." In: *Physical review E* 65.4 (2002), p. 046308. DOI: https://doi.org/10.1103/PhysRevE.65.046308.

[64] Nathen, P., Gaudlitz, D., Kratzke, J., and Krause, M. "An extension of the Lattice-Boltzmann Method for simulating turbulent flows around rotating geometries of arbitrary shape." In: *21st AIAA Computational Fluid Dynamics Conference*. 2013. DOI: 10.2514/6.2013-2573.

[65] Bhatnagar, P. L., Gross, E. P., and Krook, M. "A Model for Collision Processes in Gases. i. Small Amplitude Processes in Charged and Neutral One-Component Systems." In: *Phys. Rev.* 94 (3 1954), pp. 511–525. DOI: 10.1103/PhysRev.94.511.

[66] Nordström, J., Nordin, N., and Henningson, D. "The Fringe Region Technique and the Fourier Method Used in the Direct Numerical Simulation of Spatially Evolving Viscous Flows." In: *SIAM Journal on Scientific Computing* 20.4 (1999), pp. 1365–1393. DOI: 10.1137/S1064827596310251.

[67] Schäfer, M., Turek, S., Durst, F., Krause, E., and Rannacher, R. "Benchmark Computations of Laminar Flow Around a Cylinder." In: *Flow Simulation with High-Performance Computers II: DFG Priority Research Programme Results 1993–1995*. 1996, pp. 547–566. DOI: 10.1007/978-3-322-89849-4_39.

[68] Kummerländer, A., Dorn, M., Frank, M., and Krause, M. J. "Implicit propagation of directly addressed grids in lattice Boltzmann methods." In: *Concurrency and Computation: Practice and Experience* 35.8 (2023), e7509. DOI: https://doi.org/10.1002/cpe.7509.

[69] Geekbench. *Geekbench Browser*. https://browser.geekbench.com/. Accessed: June 22, 2023. 2023.

[70] Airaudo, F. N., Löhner, R., Wüchner, R., and Antil, H. "Adjoint-based determination of weaknesses in structures." In: *Computer Methods in Applied Mechanics and Engineering* 417 (2023), p. 116471. DOI: https://doi.org/10.1016/j.cma.2023.116471.

[71] Honghong, S., Gang, Y., Haijiang, L., Tian, Z., and Annan, J. "Digital twin enhanced BIM to shape full life cycle digital transformation for bridge engineering." In: *Automation in Construction* 147 (2023), p. 104736. DOI: https://doi.org/10.1016/j.autcon.2022.104736.

[72] Gao, Y., Li, H., Xiong, G., and Song, H. "AIoT-informed digital twin communication for bridge maintenance." In: *Automation in Construction* 150 (2023), p. 104835. DOI: https://doi.org/10.1016/j.autcon.2023.104835.

[73] Ye, C., Butler, L., Bartek, C., iangurazov, M., Lu, Q., Gregory, A., Girolami, M., and Middleton, C. "A digital twin of bridges for structural health monitoring." In: *12th international Workshop on Structural Health Monitoring 2019*. Stanford University. 2019. DOI: 10.12783/shm2019/32287.

[74] Zribi, H., Abid, T. B., Elloumi, A., Hani, Y., Graba, B. B., and and, A. E. "Industry 4.0: digital twins characteristics, applications, and challenges in-built environments." In: *Production & Manufacturing Research* 13.1 (2025), p. 2456277. DOI: 10.1080/21693277.2025.2456277.

[75] Schrotter, G. and Hürzeler, C. "The digital twin of the city of Zurich for urban planning." In: *PFG–Journal of Photogrammetry, Remote Sensing and Geoinformation Science* 88.1 (2020), pp. 99–112. DOI: 10.1007/s41064-020-00092-2.

[76] Pasquier, M., Jay, S., Jacob, J., and Sagaut, P. "A Lattice-Boltzmann-based modelling chain for traffic-related atmospheric pollutant dispersion at the local urban scale." In: *Building and Environment* 242 (2023), p. 110562. DOI: https://doi.org/10.1016/j.buildenv.2023.110562.

[77] van Hooff, T. and Blocken, B. "Coupled urban wind flow and indoor natural ventilation modelling on a high-resolution grid: A case study for the Amsterdam ArenA stadium." In: *Environmental Modelling & Software* 25.1 (2010), pp. 51–65. DOI: https://doi.org/10.1016/j.envsoft.2009.07.008.

[78] Jeanjean, A., Hinchliffe, G., McMullan, W., Monks, P., and Leigh, R. "A CFD study on the effectiveness of trees to disperse road traffic emissions at a city scale." In: *Atmospheric Environment* 120 (2015), pp. 1–14. DOI: https://doi.org/10.1016/j.atmosenv.2015.08.003.

[79] Taleb, H. M. and Kayed, M. “Applying porous trees as a windbreak to lower desert dust concentration: Case study of an urban community in Dubai.” In: *Urban Forestry & Urban Greening* 57 (2021), p. 126915. DOI: https://doi.org/10.1016/j.ufug.2020.126915.

[80] Kim, K.-H., Kabir, E., and Kabir, S. “A review on the human health impact of airborne particulate matter.” In: *Environment international* 74 (2015), pp. 136–143. DOI: https://doi.org/10.1016/j.envint.2014.10.005.

[81] Boningari, T. and Smirniotis, P. G. “impact of nitrogen oxides on the environment and human health: Mn-based materials for the NOx abatement.” In: *Current Opinion in Chemical Engineering* 13 (2016). Energy and Environmental Engineering / Reaction engineering and catalysis, pp. 133–141. DOI: https://doi.org/10.1016/j.coche.2016.09.004.

[82] Dall’Osto, M., Querol, X., Alastuey, A., O’Dowd, C., Harrison, R. M., Wenger, J., and Gómez-Moreno, F. J. “On the spatial distribution and evolution of ultrafine particles in Barcelona.” In: *Atmospheric Chemistry and Physics* 13 (2013), pp. 741–759. DOI: 10.5194/acp-13-741-2013.

[83] Garcia-Marlès, M. et al. “Source apportionment of ultrafine particles in urban Europe.” In: *Environment international* 194 (2024), p. 109149. DOI: 10.1016/j.envint.2024.109149.

[84] Dröge, J., Klingelhöfer, D., Braun, M., and Groneberg, D. A. “influence of a large commercial airport on the ultrafine particle number concentration in a distant residential area under different wind conditions and the impact of the COViD-19 pandemic.” In: *Environmental Pollution* 345 (2024), p. 123390. DOI: 10.1016/j.envpol.2024.123390.

[85] Mohan, V., Soni, V. K., and Mishra, R. K. “Analysing the impact of day-night road traffic variation on ultrafine particle number size distribution and concentration at an urban site in the megacity Delhi.” In: *Atmospheric Pollution Research* 15 (4) (2024), p. 102065. DOI: 10.1016/j.apr.2024.102065.

[86] Samad, A., Arango, K., Florez, D. A., Chourdakis, i., and Vogt, U. “Assessment of Coarse, Fine, and Ultrafine Particles in S-Bahn Trains and Underground Stations in Stuttgart.” In: *Atmosphere* 13 (11) (2022), p. 1875. DOI: 10.3390/atmos13111875.

[87] Bächler, P., Weis, F., Kohler, S., and Dittler, A. “Exploratory measurements of ambient air quality in a residential area applying a diffusion charge based UFP monitor.” In: *Gefahrstoffe* 84 (2024). DOI: 10.37544/0949-8036-2024-01-02-17.

[88] Bächler, P., Müller, T. K., Warth, T., Yildiz, T., and Dittler, A. “impact of ambient air filters on PM concentration levels at an urban traffic hotspot (Stuttgart, Am Neckartor).” In: *Atmospheric Pollution Research* 12 (6) (2021), p. 101059. DOI: 10.1016/j.apr.2021.101059.

[89] Kaur, K. and Kelly, K. E. “EPerformance evaluation of the Alphasense OPC-N3 and Plantower PMS5003 sensor in measuring dust events in the Salt Lake Valley, Utah.” In: *Atmospheric Measurement Techniques* 16 (10) (2023), pp. 2455–2470. DOI: 10.5194/amt-16-2455-2023.

[90] OpenStreetMap contributors. *Planet dump*. Data retrieved from Open-StreetMap and available under the Open Database License. 2025. URL: https://planet.openstreetmap.org/.

[91] Thomason, L. *TinyXML-2*. https://github.com/leethomason/tinyxml2.

[92] Rijnders, E., Janssen, N. A., Vliet, P. H. van, and Brunekreef, B. “Personal and outdoor nitrogen dioxide concentrations in relation to degree of urbanization and traffic density.” In: *Environmental Health Perspectives* 109.suppl 3 (2001), pp. 411–417. DOI: 10.1289/ehp.01109s3411.

[93] Simonis, S., Hafen, N., Jeßberger, J., Dapelo, D., Thäter, G., and Krause, M. J. “Homogenized lattice Boltzmann methods for fluid flow through porous media – part i: kinetic model derivation.” In: *ESAiM M2AN* (2025). DOI: 10.1051/m2an/2025005.

[94] Lasaga, A. *Kinetic theory in the earth sciences*. Princeton University Press, 2014, pp. 1–811.

[95] Kummerländer, A., Bukreev, F., Berg, S. F. R., Dorn, M., and Krause, M. J. “Advances in Computational Process Engineering using Lattice Boltzmann Methods on High Performance Computers.” In: *High Performance Computing in Science and Engineering ’22*. Cham, 2024, pp. 233–247. DOI: 10.1007/978-3-031-46870-4_16.

[96] Krause, M. J., Klemens, F., Henn, T., Trunk, R., and Nirschl, H. “Particle flow simulations with homogenised lattice Boltzmann methods.” In: *Particuology* 34 (2017), pp. 1–13. DOI: 10.1016/j.partic.2016.11.001.

[97] Jérôme Jacob, O. M. and Sagaut, P. “A new hybrid recursive regularised Bhatnagar–Gross–Krook collision model for Lattice Boltzmann method-based large eddy simulation.” In: *Journal of Turbulence* 19.11-12 (2018), pp. 1051–1076. DOI: 10.1080/14685248.2018.1540879.

[98] Simonis, S. "Lattice Boltzmann Methods for Partial Differential Equations." URL: https://publikationen.bibliothek.kit.edu/1000161726. Doctoral thesis. Karlsruhe institute of Technology (KiT), 2023. DOI: 10.5445/iR/1000161726.

[99] Simonis, S., Frank, M., and Krause, M. J. "On relaxation systems and their relation to discrete velocity Boltzmann models for scalar advection–diffusion equations." In: *Philosophical Transactions of the Royal Society A* 378 (2020), p. 20190400. DOI: 10.1098/rsta.2019.0400.

[100] Simonis, S., Frank, M., and Krause, M. J. "Constructing relaxation systems for lattice Boltzmann methods." In: *Applied Mathematics Letters* 137 (2023), p. 108484. DOI: 10.1016/j.aml.2022.108484.

[101] Gromke, C., Buccolieri, R., Di Sabatino, S., and Ruck, B. "Dispersion study in a street canyon with tree planting by means of wind tunnel and numerical investigations – Evaluation of CFD data with experimental data." In: *Atmospheric Environment* 42.37 (2008), pp. 8640–8650. DOI: https://doi.org/10.1016/j.atmosenv.2008.08.019.

[102] Merlier, L., Jacob, J., and Sagaut, P. "Lattice-Boltzmann Large-Eddy Simulation of pollutant dispersion in street canyons including tree planting effects." In: *Atmospheric Environment* 195 (2018), pp. 89–103. DOI: https://doi.org/10.1016/j.atmosenv.2018.09.040.

[103] Hettel, M., Bukreev, F., Daymo, E., Kummerländer, A., Krause, M. J., and Deutschmann, O. "Calculation of Single and Multiple Low Reynolds Number Free Jets with a Lattice-Boltzmann Method." In: *AIAA Journal* 63.4 (2025), pp. 1305–1318. DOI: 10.2514/1.J064280.

[104] Guo, Z., Zheng, C., and Shi, B. "An extrapolation method for boundary conditions in lattice Boltzmann method." In: *Physics of Fluids* 14.6 (2002), pp. 2007–2010. DOI: 10.1063/1.1471914.

[105] Zhao-Li, G., Chu-Guang, Z., and Bao-Chang, S. "Non-equilibrium extrapolation method for velocity and pressure boundary conditions in the lattice Boltzmann method." In: *Chinese Physics* 11.4 (2002), p. 366. DOI: 10.1088/1009-1963/11/4/310.

[106] International Council on Mining and Metals (ICMM). *Water Management in Mining*. 2012. URL: http://hdl.handle.net/20.500.12424/185794.

[107] Gunson, A., Klein, B., Veiga, M., and Dunbar, S. “Reducing mine water requirements.” In: *Journal of Cleaner Production* 21.1 (2012), pp. 71–82. DOI: https://doi.org/10.1016/j.jclepro.2011.08.020.

[108] Baker, E. *Mine Tailings Storage: Safety is No Accident.* GRiD-Arendal, 2017.

[109] Matschullat, J. and Gutzmer, J. “ Mining and Its Environmental Impacts.” In: *Encyclopedia of Sustainability Science and Technology*. 2012, pp. 6633–6645. DOI: 10.1007/978-1-4419-0851-3_205.

[110] Tran, T., Truong, T., Tran, T., Hài, N., and Dao, Q. “An overview of the application of machine learning in predictive maintenance.” In: *Petrovietnam Journal* 10 (2021), pp. 47–61. DOI: 10.47800/PVJ.2021.10-05.

[111] McCoy, J. and Auret, L. “Machine learning applications in minerals processing: A review.” In: *Minerals Engineering* 132 (2019), pp. 95–109. DOI: https://doi.org/10.1016/j.mineng.2018.12.004.

[112] Anagnostopoulos, C., Mylonas, G., Fournaris, A. P., and Koulamas, C. “A Design Approach and Prototype Implementation for Factory Monitoring Based on Virtual and Augmented Reality at the Edge of Industry 4.0.” In: *2023 IEEE 21st International Conference on Industrial Informatics (INDIN)*. 2023, pp. 1–8. DOI: 10.1109/INDIN51400.2023.10218239.

[113] Alpha Wastewater. *Wastewater Treatment and Augmented Reality: Visualizing Processes and Optimizing Operations*. Accessed: 2025-04-16. 2023. URL: https://www.alphawastewater.com/wastewater-treatment-and-augmented-reality-visualizing-processes-and-optimizing-operations.

[114] Alatawi, F. et al. “An AR-Assisted Deep Reinforcement Learning-Based Model for Industrial Training and Maintenance.” In: *Sensors* 23.13 (2023), p. 6024. DOI: 10.3390/s23136024.

[115] Landman, K. A. and White, L. R. “Predicting filtration time and maximizing throughput in a pressure filter.” In: *AiChE Journal* 43.12 (1997), pp. 3147–3160. DOI: https://doi.org/10.1002/aic.690431204.

[116] Puig-Bargués, J., Duran-Ros, M., Arbat, G., Barragán, J., and Ramírez de Cartagena, F. “Prediction by neural networks of filtered volume and outlet parameters in micro-irrigation sand filters using effluents.” In: *Biosystems Engineering* 111.1 (2012), pp. 126–132. DOI: https://doi.org/10.1016/j.biosystemseng.2011.11.005.

[117] Hawari, A. H. and Alnahhal, W. “Predicting the performance of multimedia filters using artificial neural networks.” In: *Water Science and Technology* 74.9 (2016), pp. 2225–2233. DOI: 10.2166/wst.2016.380.

[118] Griffiths, K. and Andrews, R. “Application of artificial neural networks for filtration optimization.” In: *Journal of Environmental Engineering* 137.11 (2011), pp. 1040–1047. DOI: 10.1061/(ASCE)EE.1943-7870.0000439.

[119] Khan, W. A., Chung, S.-H., Awan, M. U., and Wen, X. “Machine learning facilitated business intelligence (Part I): Neural networks learning algorithms and applications.” In: *Industrial Management & Data Systems* 120.4 (2020), pp. 657–698. DOI: 10.1108/IMDS-07-2019-0361.

[120] Khan, W. A., Chung, S.-H., Eltoukhy, A. E., and Khurshid, F. “A novel parallel series data-driven model for IATA-coded flight delays prediction and features analysis.” In: *Journal of Air Transport Management* 114 (2024), p. 102488. DOI: 10.1016/j.jairtraman.2023.102488.

[121] Spielman, L. and Goren, S. L. “Model for predicting pressure drop and filtration efficiency in fibrous media.” In: *Environmental Science & Technology* 2.4 (1968), pp. 279–287. DOI: 10.1021/es60016a003.

[122] Krause, M. J., Kummerländer, A., Avis, S. J., Kusumaatmaja, H., Dapelo, D., Klemens, F., Gaedtke, M., Hafen, N., Mink, A., Trunk, R., Marquardt, J. E., Maier, M.-L., Haussmann, M., and Simonis, S. “OpenLB—Open source lattice Boltzmann code.” In: *Computers & Mathematics with Applications* 81 (2021), pp. 258–288. DOI: https://doi.org/10.1016/j.camwa.2020.04.033.

[123] Krause, M. J., Klemens, F., Henn, T., Trunk, R., and Nirschl, H. “Particle flow simulations with homogenised lattice Boltzmann methods.” In: *Particuology* 34 (2017), pp. 1–13. DOI: https://doi.org/10.1016/j.partic.2016.11.001.

[124] Trunk, R., Bretl, C., Thäter, G., Nirschl, H., Dorn, M., and Krause, M. J. “A Study on Shape-Dependent Settling of Single Particles with Equal Volume Using Surface Resolved Simulations.” In: *Computation* 9.4 (2021). DOI: 10.3390/computation9040040.

[125] Stipić, D., Budinski, L., and Fabian, J. “Sediment transport and morphological changes in shallow flows modelled with the lattice Boltzmann method.” In: *Journal of Hydrology* 606 (2022), p. 127472. DOI: https://doi.org/10.1016/j.jhydrol.2022.127472.

[126] Hafen, N., Dittler, A., and Krause, M. J. “Simulation of particulate matter structure detachment from surfaces of wall-flow filters applying lattice Boltzmann methods.” In: *Computers & Fluids* 239 (2022), p. 105381. DOI: https://doi.org/10.1016/j.compfluid.2022.105381.

[127] Zhang, X., Zhang, Z., Zhang, Z., and Zhang, X. “Lattice Boltzmann modeling and analysis of ceramic filtration with different pore structures.” In: *Journal of the Taiwan Institute of Chemical Engineers* 132 (2022), pp. 104–113. DOI: 10.1007/s11814-022-1329-3.

[128] Haussmann, M., Ries, F., Jeppener-Haltenhoff, J. B., Li, Y., Schmidt, M., Welch, C., illmann, L., Böhm, B., Nirschl, H., Krause, M. J., and Sadiki, A. “Evaluation of a Near-Wall-Modeled Large Eddy Lattice Boltzmann Method for the Analysis of Complex Flows Relevant to iC Engines.” In: *Computation* 8.2 (2020). DOI: 10.3390/computation8020043.

[129] Wichmann, K.-R. “Evaluation and Development of High Performance CFD Techniques with Focus on Practice-Oriented Metrics.” PhD thesis. Technische Universität München, 2014.

[130] Callé, S., Bémer, D., Thomas, D., Contal, P., and Leclerc, D. “Changes in the performances of filter media during clogging and cleaning cycles.” In: *The Annals of Occupational Hygiene* 45.2 (2001), pp. 115–121. DOI: 10.1093/annhyg/45.2.115.

[131] Kandra, H. S., McCarthy, D., Fletcher, T. D., and Deletic, A. “Assessment of clogging phenomena in granular filter media used for stormwater treatment.” In: *Journal of Hydrology* 512 (2014), pp. 518–527. DOI: https://doi.org/10.1016/j.jhydrol.2014.03.009.

[132] Kehat, E., Lin, A., and Kaplan, A. “Clogging of filter media.” In: *Industrial & Engineering Chemistry Process Design and Development* 6.1 (1967), pp. 48–55. DOI: 10.1021/i260021a009.

[133] Fränkle, B., Morsch, P., and Nirschl, H. “Regeneration assessments of filter fabrics of filter presses in the mining sector.” In: *Minerals Engineering* 168 (2021), p. 106922. DOI: https://doi.org/10.1016/j.mineng.2021.106922.

[134] Callé, S., Contal, P., Thomas, D., Bémer, D., and Leclerc, D. “Description of the clogging and cleaning cycles of filter media.” In: *Powder Technology* 123.1 (2002), pp. 40–52. DOI: https://doi.org/10.1016/S0032-5910(01)00430-2.

[135] Lyu, J. et al. *Handbook of Digital Twins*. Routledge, 2024.

[136] Sandberg, i. W., Lo, J. T., Fancourt, C. L., Principe, J. C., Katagiri, S., and Haykin, S. *Nonlinear dynamical systems: feedforward neural network perspectives*. Vol. 21. John Wiley & Sons, 2001.

[137] Sherstinsky, A. “Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network.” In: *Physica D: Nonlinear Phenomena* 404 (2020), p. 132306. DOI: 10.1016/j.physd.2019.132306.

[138] Waqas, M. and Humphries, U. W. “A critical review of RNN and LSTM variants in hydrological time series predictions.” In: *MethodsX* 13 (2024), p. 102946. DOI: 10.1016/j.mex.2024.102946.

[139] Mienye, i. D., Swart, T. G., and Obaido, G. “Recurrent Neural Networks: A Comprehensive Review of Architectures, Variants, and Applications.” In: *information* 15.9 (2024). DOI: 10.3390/info15090517.

[140] McClarren, R. “Feed-Forward Neural Networks.” In: 2021, pp. 119–148. DOI: 10.1007/978-3-030-70388-2_5.

Appendix

A List of Application Cases

Application Cases 1 and 2 are available in the master branch of the OpenVisFlow repository, hosted at https://gitlab.com/Dteutscher/openvisflow. Application Case 3 can be found in the master branch of the OpenLB repository at https://gitlab.com/openlb/olb. The source code for Application Case 4 is available only upon reasonable request and with explicit permission from Simex Filterpressen und Wassertechnik GmbH & Co. KG.

1. OpenVisFlow – Chapter 3: Visualization module

   Description: see Chapter 3
   Commit hash:
   `4a4702968e06d4c6a3da298898692d3023b04b3c`
   Case folder: `unity/Assets/Scenes/MenueOpenLBar`
   Application: https://www.openlb.net/ar/
   Date: 27/03/2022
   Released state: Version 0.2r1

2. Paint2Sim – Chapter 4: Virtual CFD Lab

   Description: see Chapter 4
   Commit hash:
   `4a4702968e06d4c6a3da298898692d3023b04b3c`
   Case folder: `unity/Assets/Scenes/PaintToSim`
   Application: https://www.openlb.net/paint2sim/
   Date: 16/02/2023
   Released state: Version 0.1r1

3. Urban DT – Chapter 5: CFD-Based Digital Twin for Urban Environments

   Description: see
   Commit hash:
   `08d7c1b5e479c8587448c0301386fe4b4be25f9a`
   Case folder: `apps/dennis/urban_dt`
   Date: 24/04/2025
   Released state: Version 1.8

4. FiltAR – Chapter 6: Neural Network-Based Digital twin for a Chamber Filter Press

   Description: see Chapter 6
   Link: https://bwsyncandshare.kit.edu/apps/files/files/4570688644?dir=/lbrg/projects/FiltAR
   Date: 14/03/2025

B Acronyms

AI artificial intelligence

AR augmented reality

CFD computational fluid dynamics

CI90% 90% Confidence Interval

DT digital twin

DTE digital twin environment

DTI digital twin instance

DTP digital twin prototype

FFNN feed forward neural network

GLB binary GL transmission file format

IoT internet of things

LBM lattice Boltzmann method

ML machine learning

NN neural network

NSE Navier-Stokes equation

OSM OpenStreetMap

PINN physics-informed neural network

PLY polygon file format

RL2N relative L^2-norm

RL2N-B relative L^2-norm-bounds

RNN recurrent neural network

VR virtual reality

VTK visualization toolkit

C User Guide for *paint2sim*

The application *paint2sim* is available for download at www.openlb.net/paint2sim/. This section provides a detailed description of how to use the application. The forward/backward buttons are used to navigate between different steps, while the cross can be used to reset the application. All steps are shown in the remaining part of this section. It is advisable to use a thick fine-tip marker to draw the domain for effective domain recognition.

Selecting Resolution: Begin by selecting your preferred resolution using the slider, as illustrated in Figure C.1a. Options range from 100 to 300 in 50-unit increments, representing the number of cells across the screen width on your mobile device. Note that higher resolutions may impact user experience on certain devices, potentially causing performance issues.

Scanning the Domain: Utilize the scan button (Figure C.1b) to initiate the domain scanning process. Once completed, input the scale of the domain in meters (Figure C.1c).

Setting Characteristic Length: Set the characteristic length for the Reynolds number estimation either through the input field or by adjusting the scale via touch (Figure C.1d).

$$Re = \frac{u \cdot l_c}{\nu}$$

Edit the Domain: The domain can be edited using various touch-based options as shown in Figure C.1e. Edit function are: Add walls; Remove existing walls; Declare particles for drag and lift calculations (currently limited to one particle); Define mandatory inlets and outlets

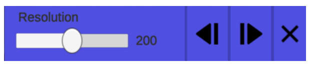
Fluid Parameters: Choose fluid parameters from the dropdown menu, selecting Water, Oil, or Air. Alternatively, input custom kinematic viscosity and density values (Figure C.1f).

Reynolds Number: Enter the Reynolds number for the simulation. (Figure C.1h).

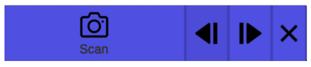
Inflow Direction: If not opting for a Poiseuille inflow, enter the desired inflow direction (Figure C.1g).

Simulation: The final step initiates the simulation, which runs continuously until manually canceled using the back button or the cross button. During the simulation, toggle between velocity and pressure visualization methods, and view simulation parameters as needed.

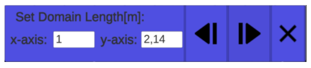
If every step was done correctly, the simulation should be running *just-in-time* on the mobile screen. In Figure C.2, an example simulation is shown.


Resolution 200

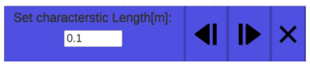
(a) *Selecting Resolution*



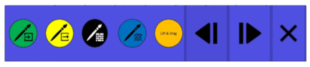
(b) *Scanning Domain*


Set Domain Length[m]:
x-axis: 1 y-axis: 2,14

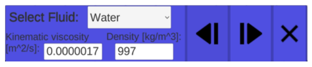
(c) *Input Domain scale*


Set characterstic Length[m]:
0.1

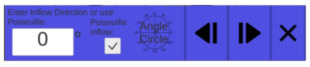
(d) *Characteristic length*



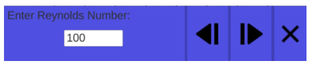
(e) *Edit domain*


Select Fluid: Water
Kinematic viscosity [m^2/s]: 0.0000017
Density [kg/m^3]: 997

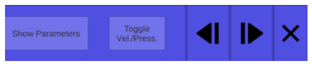
(f) *Select fluid*


Enter Inflow Direction or use Poiseuille: 0 °
Poiseuille inflow: ✓

(g) *Choose inflow*

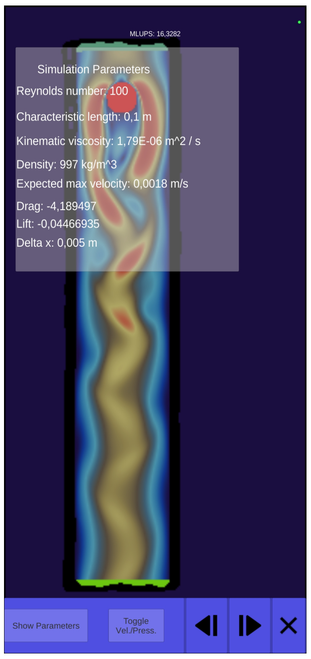

Enter Reynolds Number:
100

(h) *Reynolds Number*



(i) *Toggle vizualization*

Figure C.1: *The figures show the input menus of paint2sim to scan and set up a simulation*


MLUPS: 16,3282

Simulation Parameters
Reynolds number: 100
Characteristic length: 0,1 m
Kinematic viscosity: 1,79E-06 m^2 / s
Density: 997 kg/m^3
Expected max velocity: 0,0018 m/s
Drag: -4,189497
Lift: -0,04466935
Delta x: 0,005 m

Figure C.2: *Resulting example simulation after following each step (Figure C.1a - C.1i).*