VRIJE UNIVERSITEIT AMSTERDAM

UNIVERSITEIT VAN AMSTERDAM

Master Thesis

Design and Implementation of a 5G Ray-Tracing Simulation Pipeline for Dataset Generation

by

Chris Benhard Armanda

(VUnetID: car100 / UvA Studentnummer: 15998034)

First supervisor:	dr. A. S. Z. Belloum
Daily supervisor:	Z. Yang, MSc
Second reader:	dr. N. Kokash

August 22, 2025

Submitted in partial fulfillment of the requirements for the joint UvA-VU degree of Master of Science in Computer Science

"The Road goes ever on and on,
Down from the door where it began.
Now far ahead the Road has gone,
And I must follow, if I can,
Pursuing it with eager feet,
Until it joins some larger way
Where many paths and errands meet.
And whither then? I cannot say."

—J. R. R. Tolkien, *The Road Goes Ever On*

Abstract

In this work, we investigated a challenge faced by mobile operators as they upgrade their radio infrastructures to support 5G New Radio (5G NR) technology. We propose to address the challenge of predicting 5G NR coverage in diverse environments through a simulation pipeline based on Ray Tracing. The system is to support varied geographical datasets, from 2D tile maps to detailed 3D digital twins, producing numerical arrays that characterise 5G signal propagation under different transmitter configurations and environmental conditions.

The pipeline should be capable of generating thousands of datasets, enabling its integration into a larger Machine Learning (ML) training workflow for signal quality prediction. The research also examines potential improvements to the pipeline and to the third-party dependencies used in its development.

Contents

List of Figures

LIST OF FIGURES

List of Tables

LIST OF TABLES

1

Introduction

Today, mobile operators worldwide are rapidly increasing their 5G coverage. The deployment of the technology requires the construction of cell tower sites, where the operators will install radio equipments providing signal coverage for their end customers. Before the operators can proceed with the construction stage, they first have to plan the radio coverage design by asking some important questions: Which 5G-compatible frequency should the radio equipment transmit? Are there physical obstacles that may degrade the transmitted signal quality? Are the planned signal spectrum available in the area without the risk of crosstalk? In most cases, these questions can be definitively answered only by conducting site survey.

However, site surveys are costly and manpower-intensive. This is especially true for planned sites in remote areas, such as isolated island resorts or mountainous regions. For faster decision making during the planning and design stage, mobile operators may leverage simulation tools before deciding whether a physical site survey is required. Given accurate geodesic maps, a simulation tool can predict the behaviour of transmitted signal along with the important indicators, such as signal loss and Line-of-Sight (LoS).

1.1 Research Questions

The objective of this work is two-pronged. Firstly, we will study the generated datasets to understand what factors would affect signal transmission the most, so as to create a reliable dataset to improve upon the numerical analysis-based tools available in the market. Secondly, we will also benchmark the available tools in a distributed computing environment, by examining the usage of compute resources such as CPU, memory usage,

and the computing efficiency rates. To guide us in our objectives, we attempt to address the following research questions:

- **RQ1**: *What considerations should be taken into account in developing a dataset-generating pipeline for the purposes of academic research, but also extendable for use in the telecommunications industry?*
  The pipeline should have the capability to cater a wide variety of stakeholders in the 5G technology. The datasets to be produced by the pipeline will be geared for further academic research by the end of the project, but it should also lay the foundations for a functional software to be utilised by mobile operators to assist them in planning their future state of their network infrastructure. Since multiple stakeholders will introduce their own business objectives with their own constraints, we will document all of our design and architectural decisions in the pipeline's development lifecycle, after which we will analyse what consequences are produced by taking each design decision.

- **RQ2**: *To what extent can ML models and Network Digital Twin (NDT) platforms be effectively combined to leverage their complementary strengths for 5G network planning and troubleshooting, and what criteria should guide such integration?*
  Telecommunication standardisation bodies, such as ITU-T and 3GPP, have proposed integrating ML techniques into Network Digital Twin (NDT) platforms. While NDTs can continuously monitor network performance, their effectiveness depends on the quality of the datasets they consume. NDT platforms with simulation capabilities can generate physically accurate data, but often at high computational cost. ML models, once trained, can produce results quickly but risk reduced accuracy in unfamiliar scenarios. This research examines whether combining the two approaches can generate datasets that are both representative and trustworthy, and how this trustworthiness can be assessed for use in automated decision-making.

- **RQ3**: *What gaps exist in the effort of accurately capturing real-world environmental and operational factors to generate accurate datasets, and what are the implications of these gaps for downstream modelling and analysis?*
  The propagation of electromagnetic signals in urban and rural environments is influenced by a multitude of interacting variables, such as atmospheric conditions, obstructions, and material properties. These variables can change unpredictably

over time and space. This question seeks to identify which aspects of the physical and operational environment are most critical to capture, what gaps persist in current simulation and measurement techniques, and how those gaps impact the interpretability, trustworthiness, and operational relevance of generated datasets.

1.2 Work Structure

This work is structured as follows. In Chapter 2, we will provide the necessary background information on the 5G signal propagation simulation tools and the ML techniques used in this work. In Chapter 3, we will describe the methodology used to conduct our experiments, including the simulation setup, data collection, and analysis techniques. In Chapter 4, we will present the results of our experiments, including a comparison of the simulation tools and an analysis of the key parameters affecting signal propagation. In Chapter **??**, we will discuss the potential threats to the validity of our findings and how we mitigated them. In Chapter 9, we will review related work in the field of 5G signal propagation simulation and ML techniques. In Chapter 8, we will discuss the implications of our findings for mobile operators and the telecommunications industry as a whole.

Our work aims to provide a comprehensive understanding of the current state of 5G signal propagation simulation tools and their potential for improving the planning and design of mobile networks. As such, discussions on other aspects to 5G technologies will be omitted, as they are not directly related to our research questions. The topics that this work will not discuss include the 5G backhaul network and packet core network. We will focus on the ray-tracing simulation tools and the ML techniques that can be used to improve the accuracy and efficiency of these simulations. We will also discuss the potential for hybrid approaches that combine traditional simulation methods with ML techniques to leverage their complementary strengths. We hope that our findings will contribute to the ongoing development of more efficient and effective tools for mobile network planning and design, ultimately leading to better service quality for end customers.

2

Background

The 5th Generation (5G) of mobile cellular network systems is being actively deployed by Mobile Network Operators (MNOs) worldwide. Compared to its predecessors, 5G offers significantly lower latency and higher data throughput, enabling new use cases such as the Industrial Internet of Things (IIoT) and large-scale machine-to-machine communication. To facilitate widespread adoption, the 3rd Generation Partnership Project (3GPP) standardised Frequency Range 1 (FR1), which overlaps with the frequency bands previously allocated for 4G (LTE). This allows MNOs to initially deploy 5G using existing infrastructure, ensuring a smoother migration path and reducing capital expenditure.

However, co-deployment of 4G and 5G services within the same spectrum introduces trade-offs. Shared use of FR1, spanning 410 MHz to 7125 MHz, increases the risk of interference and spectral congestion. Moreover, physical constraints in these lower frequency bands limit the achievable data rates and latency improvements promised by 5G. To overcome these limitations, 3GPP introduced Frequency Range 2 (FR2), which includes bands from 24.25 GHz to 71 GHz (1). These higher-frequency signals fall in the millimeter-wave (mmWave) range, characterised by shorter wavelengths and larger available bandwidth.

mmWave is a key enabler of ultra-low-latency communication and multi-gigabit-per-second data rates. However, it also introduces new challenges. Due to their short wavelengths, mmWave signals experience greater path loss and are more susceptible to attenuation from buildings, foliage, and atmospheric conditions. As a result, mmWave deployments require a denser infrastructure of small cells and base stations to maintain reliable coverage.

Before deploying such infrastructure, MNOs must conduct detailed site surveys to assess the radio environment, including potential obstacles and diffraction patterns. These surveys are traditionally performed in the field and involve considerable operational costs,

particularly in remote or urban-dense regions. To reduce planning overhead and accelerate rollout, operators increasingly rely on simulation-based approaches. By leveraging high-resolution digital maps and propagation models, signal simulators allow operators to estimate coverage, signal degradation, and line-of-sight availability before any physical deployment occurs. This thesis investigates such simulation tools in the context of generating datasets for training machine learning models for signal quality prediction.

2.1 High-Level Architecture of 5G Radio Network

All mobile devices used by end subscribers to a MNO connect wirelessly via the MNO's Radio Access Network (RAN). In a 5G network, the base station is referred to as a *gNodeB*. Each gNodeB typically comprises multiple radio units operating on frequencies standardised by 3GPP. Wireless communication between user devices and the gNodeB is bidirectional: data sent to user equipment is termed *downlink* (DL), while data sent from the user device back to the network is called *uplink* (UL).

A single gNodeB generally covers three sectors, each serving a *cell*. These cells are often conceptualised as 120° arcs dividing the gNodeB's coverage area into three roughly circular segments. The gNodeB may transmit across multiple frequency bands simultaneously. The range and behaviour of these transmissions are frequency-dependent: signals in higher bands (*e.g.,* mmWave) suffer greater path loss and therefore cover shorter distances. This introduces an engineering and economic trade-off: achieving wide-area mmWave coverage would require deploying many additional gNodeBs. A common approach to mitigate this limitation is to deploy gNodeBs capable of dual-band operation using both FR1 and FR2 frequencies (2). This allows devices located outside the mmWave range to fall back to FR1, albeit with lower data rates and higher latency.

2.2 Radio Propagation Effects and Simulation Challenges in 5G RAN

This section outlines the physical layer phenomena relevant to signal propagation in 5G networks and their implications for simulation and coverage modelling. Since our focus is on the radio access side, backhaul and core network components are excluded.

Radio Frequency (RF) communication planning involves estimating the signal's electromagnetic power at both the transmitter and receiver ends. During its airborne transmission, a signal undergoes various propagation effects that reduce its power before reaching

the receiver. These include reflection and diffraction around obstacles, scattering from rough surfaces, and refraction due to atmospheric inhomogeneity. Collectively, these effects result in *path loss*, *i.e.,* degradation of the signal's power as it travels through the environment.

To account for these effects, RF engineers develop what is known as a *link budget*. A link budget estimates whether the received signal power exceeds the minimum required for successful demodulation and decoding. A simplified link budget equation is shown below:

$$P_{Rx} = P_{Tx} + G_{Tx} - L_{Tx} - L_{FS} - L_M + G_{Rx} - L_{Rx}$$

Here, P_{Tx} is the transmit power, and P_{Rx} is the received power. The transmitter's antenna introduces gain G_{Tx} and internal loss L_{Tx}, often due to cabling or impedance mismatches. L_{FS} represents free-space path loss, which dominates in open environments and follows an inverse square law relative to distance. L_M captures miscellaneous losses including building penetration, diffraction, and foliage attenuation. On the receiver side, G_{Rx} is the antenna gain and L_{Rx} is any internal loss before signal processing. All power gains and losses in this equation are in their logarithmic forms, defined by the units of decibels (dB).

Calculating path loss is not arbitrary. A reliable model should therefore take into account the behaviour of the transmitted signal. A commonly used theoretical model for estimating free-space path loss is the Friis transmission equation:

$$\frac{P_{Rx}}{P_{Tx}} = G_{Tx}G_{Rx}\left(\frac{\lambda}{4\pi d}\right)^2$$

Here, λ is the wavelength of the transmitted signal, and d is the distance between transmitter and receiver. This expression assumes ideal, line-of-sight (LoS) conditions with no reflections or obstructions. In practice, this model defines the baseline for free-space loss (L_{FS}) in the link budget. Since wavelength λ is inversely proportional to frequency f, higher-frequency signals (*e.g.,* FR2/mmWave) experience greater path loss over the same distance compared to lower-frequency signals (*e.g.,* FR1/sub-6 GHz). This relationship imposes a strong frequency dependence on coverage planning and is a key factor in the decision to deploy multi-band 5G systems.

The miscellaneous losses as represented by the symbol L_M in the link budget equation is the accumulation of all other losses that occur during transmission. These losses are highly dependent on the environment and the specific deployment scenario. In urban environments, buildings can cause significant diffraction and reflection, leading to additional path

loss. To estimate the effects of the building materials against the signal, simulators often based the calculations using models such as ITU-R P.1238, which provides a framework for estimating the diffraction loss due to buildings and other obstacles. This model considers the height, width, and material properties of the buildings, as well as the angle of incidence of the incoming signal. Another model that simulators use is the COST 231 model, which is based on empirical measurements and provides a more accurate estimate of the path loss in urban environments. This model takes into account the frequency of the signal, the distance between the transmitter and receiver, and the height of the buildings in the area.

The development of 5G signal standardisation, which evolved from 4G Long-Term Evolution (LTE) standards, utilised a modulation technique called Orthogonal Frequency Division Multiple Access (OFDMA). This technique divides the available bandwidth into multiple subcarriers, each carrying a portion of the data. The physics behind OFDMA provides additional challenge in estimating path losses, as the signal is transmitted over a wide range of frequencies. This means that the path loss can vary significantly depending on the frequency of the subcarrier. The mmWave frequency range, for example, is particularly susceptible to scattering and signal fading due to its shorter wavelengths. This can lead to significant variations in the received signal strength, especially in urban environments with many obstacles. To account for these variations, simulators often use models such as the ITU-R P.1411 model, which provides a framework for estimating the path loss in urban environments based on empirical measurements. This model takes into account the frequency of the signal, the distance between the transmitter and receiver, and the height of the buildings in the area.

In rural areas, foliage and terrain can introduce further attenuation. The modelling of these losses is complex and often requires empirical measurements or advanced simulation techniques. The ITU-R P.833 model is commonly used to estimate the effects of foliage on signal propagation. This model considers the type and density of vegetation, as well as the frequency of the signal. The model provides a framework for estimating the attenuation caused by foliage, which can be significant in rural areas with dense vegetation.

Accurately modelling the individual components of this link budget is critical for simulation tools aiming to reflect realistic 5G coverage in urban, rural, or industrial deployments. It is especially important for the ML models to take into account the various factors of power losses introduced by the transmission distance and environment-specific causes. In particular, ray-tracing simulators must account for site-specific propagation factors, which vary significantly depending on frequency band (*e.g.,* FR1 vs FR2), antenna configuration, and terrain complexity.

2.3 Challenges in Site Planning and RF Surveys

The deployment of 5G networks requires careful planning and site surveys to ensure optimal coverage and performance. However, traditional RF surveys can be time-consuming and costly, particularly in urban environments where access to rooftops and other elevated locations may be limited. According to a survey conducted by the United States Federal Communications Commission (FCC) in 2023, the average cost of an in-situ survey of potential radio station in the contiguous United States site can range from US$ 945 to US$ 25,000. This range may increase up to 50% for remote areas outside the contiguous United States, such as Alaska, Hawaii, and Guam (3). A trained ML model can help reduce the time and cost of RF surveys by predicting the signal quality and coverage based on data, either existing or obtained from simulations and open-sourced datasets. This can help MNOs identify the best locations for new base stations and optimise their existing infrastructure.

Simulation sweeps can be used to generate datasets for training ML models. These sweeps involve simulating the signal propagation in a specific area using a ray-tracing simulator, which takes into account the terrain, buildings, and other obstacles with slight granular differences to parameters such as antenna height, frequency, and power. The simulator generates a large number of samples, each representing a different scenario, which can be used to train the ML model. By using simulation sweeps, MNOs can generate large datasets without the need for extensive field measurements. This can significantly reduce the time and cost of RF surveys and improve the accuracy of the predictions made by the ML model.

2.4 Active Simulation in Network Digital Twin

Network Digital Twin (NDT) are systems designed to provide virtualised, software-based representations of physical communication networks. Traditionally, these systems relied on passive telemetry collection from deployed infrastructure to track performance metrics and support diagnostics. However, the developments of 5G technologies and the research on 6G technologies have extended the role of NDTs to include predictive and prescriptive capabilities.

Standardisation bodies such as the ITU and the European Telecommunications Standards Institute (ETSI) have proposed incorporating Machine Learning (ML) techniques

within NDT architectures. These techniques enable estimation of key performance indicators (KPIs), such as signal strength or throughput, at locations or times where direct telemetry is unavailable. In this capacity, the NDT evolves into a tool for forward planning and optimisation, supplementing real-time measurements with inference-based projections.

Rather than relying solely on data collected from operational networks, synthetic datasets generated through large-scale signal propagation simulations can also be used to train ML models. These simulations model signal degradation across varying terrains, frequencies, and deployment scenarios, and offer fine-grained control over transmitter parameters such as height, power, and antenna orientation. Once trained, the resulting ML models can approximate signal quality metrics in previously unsurveyed areas, helping operators identify potential coverage gaps or capacity bottlenecks prior to deployment.

This approach extends the functional range of NDTs beyond traditional monitoring, enabling the estimation of radio conditions in spatial regions not yet instrumented with physical infrastructure. Combined with digital mapping data and RF propagation models, simulation-enhanced NDTs provide a low-cost method to support RF planning, assess environmental impact on signal quality, and reduce the need for costly field surveys in early-stage deployment planning.

3

Overview of Third-Party Dependencies

This chapter outlines the experimental setup of the pipeline, including some descriptions on the pipeline's third-party dependencies such as the simulation platform and input dataset. We also describe the computing environments where the pipeline was developed and tested, along with the key configuration parameters used in the simulations.

3.1 Nvidia Sionna

Sionna is an open-source library developed by Nvidia Labs for simulating 5G signals and channels. Sionna includes channel models based on stochastic and deterministic ray tracing, as well as PHY-layer simulation modules for Multi-Input and Multi-Output (MIMO) radio communications, OFDM, and Low-Density Parity Check (LDPC) decoding (4). Sionna is implemented in Python using TensorFlow, thus enabling hardware acceleration via CUDA and simplifies integration with deep learning models. Sionna takes 3D models of structures and buildings, combined with terrain and foliage data as input to generate realistic channel models. In our pipeline, Sionna is the main signal ray-tracing simulation engine, to which we repackaged the input datasets through multiple processing stages to fit Sionna's requirements.

During the development phase of the pipeline, Sionna has undergone several version upgrades. At the start of this thesis project, Sionna was publicly released with the version number v0.19.2. Since the major version number was indicated with a 0, it is presumably intended to be a Beta version. At the time of writing this report, the developers of Sionna has released v1.1.0, which consequently deprecates some of the API functions used in our

pipeline. For the purposes of this project, we decided to persist with developing the pipeline around the Beta version of Sionna to limit feature creep. The consequences of this design decision will be further explored in the Evaluation chapter.

3.2 Distributed ASCI Supercomputer-6 (DAS-6)

DAS-6 is a cluster of distributed computing infrastructure maintained by the Advanced School for Computing and Imaging (ASCI) in the Netherlands (5). It is designed to support high-performance computing (HPC) applications and provides a scalable environment for running simulations and data-intensive tasks. Due to the high computational requirements of repeated 3D ray-tracing simulations, DAS-6 is a suitable environment for executing large-scale parameter sweeps with Sionna. In our experiments, we utilised the DAS-6 node hosted by the University of Amsterdam, with the following specifications:

- **CPU:** AMD EPYC 7402P 2.7 GHz 24-Core Processor
- **RAM:** 128 GB
- **GPU:** NVIDIA A10

In our prototype development, DAS-6 was the chosen environment where we mainly performed our testing and execution runs. DAS-6 architecture of separation between head node and GPU-equipped worker nodes enabled us to perform some testing for scalability and flexible resource detections.

3.3 3D Data Sources and Scene Preparation

To simulate signal propagation in realistic urban environments, we must first assemble accurate 3D models of the physical surroundings. This process involves sourcing building and terrain data from public geospatial datasets, followed by preparing the data using specialised tools to ensure compatibility with the simulation framework. This section introduces the key datasets and software tools used to construct and preprocess urban scenes. The full pre-processing and simulation workflow is described in detail in chapter 4.

3.3.1 3DBAG

3DBAG is a project maintained by the 3D Geoinformation Research Group at the Delft University of Technology, providing datasets and 3D models of buildings and structures in the Netherlands (6). Among 3DBAG's data sources was the Basisregistratie Adressen en Gebouwen (BAG), which is the publicly available Dutch national addresses and buildings register. The dataset includes detailed 3D models of urban environments, offering a source of building geometries with a varying degrees of Levels of Detail (LoD), namely LoD 1.2, 1.3, and 2.2. We chose to run all simulations during this project by standardising all input datasets to LoD 2.2. To scale the project, 3DBAG has divided the map of the Netherlands into smaller tiles, so that the user may download only the tiles corresponding to their areas of interest. After processing the geographic data to suit Sionna's compatibility, we can use Sionna to simulate the propagation of signals in urban areas, taking into account the effects of buildings on signal strength and quality.

3.3.2 Mitsuba 3

Mitsuba 3 is a physically-based differentiable rendering system designed for high-fidelity simulation of light transport in complex scenes (7). It supports various ray-tracing algorithms and material models, enabling accurate simulation of surface interactions such as reflection, refraction, and scattering. Unlike rasterization-based systems such as TensorFlow Graphics, which prioritize speed and real-time visualization, Mitsuba's ray-tracing approach is better suited for simulating the physical behaviour of waves, making it more appropriate for modeling electromagnetic wave propagation at high frequencies (8). This distinction is crucial, since rasterization determines visibility per pixel by projecting 3D geometry onto a 2D screen, often ignoring indirect light paths. Ray tracing simulates the physical travel of light or waves through space, tracing rays from the source and capturing interactions with surfaces in a physically plausible manner.

In this project, Mitsuba is used as a plugin within Blender to export annotated 3D scenes in a format that Sionna's ray tracer can parse. Specifically, Mitsuba generates auxiliary metadata files (also commonly referred to as sidecar files) describing the relationships between meshes, materials, and scene geometry. These sidecar files are essential for ensuring that Sionna interprets object boundaries and surface properties correctly, thereby enabling realistic urban channel simulations that take into account occlusions, multipath effects, and diffraction.

4

Implementation

This chapter presents the implementation of our pipeline, from processing raw geographic and building data to preparing simulation-ready 3D environments for ray tracing. The pipeline is designed around a distributed *Producer–Consumer* architecture, enabling parallel execution of tasks across multiple nodes while maintaining precise control over stage dependencies and job tracking.

4.1 Pipeline Architecture

The core of the implementation follows a Producer-Consumer model, coordinated through Apache Kafka, which serves as the message broker between processing stages. This design ensures:

- **Scalability** — Additional consumer nodes can be deployed to handle larger workloads without altering the pipeline logic.
- **Fault isolation** — Failures in one stage do not halt the entire pipeline; instead, they are reported and can be retried independently.
- **Stage decoupling** — Each stage operates on its own queue, enabling independent scaling and flexible orchestration.

All stages exchange status updates via Kafka topics. A sample of such message is shown below.

```
{
    "<tile-id>": {
        "status": "success" | "failed",
        "stage": 2,
        "error": "Exception message here"
    }
}
```

Messages with ‘"status": "failed"‘ also include the last thrown exception for debugging purposes. At the start of a run, the Producer generates a `Manifest.json`, a dictionary listing all detected input files along with their current processing stage. The manifest is updated whenever a stage reports completion, and it provides both a recovery point for interrupted runs and a real-time overview of pipeline progress.

The pipeline is divided into sequential **stages**, each consuming the output of its predecessor and producing input for the next. The roles of the stages as listed below are further expanded in section 4.4. In the current architecture, the pipeline comprises of 5 stages:

1. **Stage 1 — Input Parsing:** Checks input file integrity and normalises metadata for downstream processing.

2. **Stage 2 — Geometric Extraction:** Extracts and crops CityJSON tiles, preserving the original EPSG:7415 coordinates for spatial consistency.

3. **Stage 3 — 3D Conversion & Assignment of Radio Materials:** Converts intermediate CityJSON to OBJ format, applies ITU-compatible material mapping, and exports Mitsuba-compatible XML with corresponding PLY meshes.

4. **Stage 4 — Signal Simulation:** Runs Sionna ray-tracing simulations and outputs SINR, path loss, and coverage maps.

5. **Stage 5 — Ground Truth Creation:** Produces images for ground truth baseline comparison for downstream ML training pipeline. The images consist of 2D sliced projection of the tile and satellite view of the tile.

4.2 Producer Node

The Producer is the control point for initiating jobs, assigning work to consumers, and managing the manifest. Its responsibilities include:

- **Input discovery** — Scans the input directory for valid raw files and records them in `Manifest.json`.
- **Queue management** — Publishes work items to the appropriate Kafka topic for the next stage.
- **Progress tracking** — Updates the manifest upon receiving ‘"success"‘ messages from consumers, or logs errors upon ‘"failed"‘ messages.

For example, when tile `nl-3dbag-12345` completes Stage 1 successfully, the Producer:

1. Updates the manifest to mark Stage 1 as complete.
2. Places the tile into the Stage 2 queue.

4.3 Consumer Nodes

Each consumer node processes messages from its assigned stage's Kafka topic. Tasks vary depending on the stage (see section 4.4), but all consumers share:

- A loop for retrieving jobs from the Kafka topic.
- A try–except block for task execution, reporting success or failure.
- Emission of a completion message back to the Producer's status topic.

Consumers are stateless with respect to pipeline history; they operate only on the files and metadata available to them when the job arrives. This design reduces complexity and makes it easier to replace or scale stages independently.

4.4 Stages Performed by the Consumer Nodes

In the following subsections, we detail the tasks assigned to each stage of the pipeline, including their inputs, processing steps, and expected outputs. This breakdown follows the execution order within the Producer-Consumer architecture described earlier, following the transformation of raw input data into assets ready for future ML model training.

4.4.1 Input Parsing

The input files are obtained from 3DBAG, as explained in chapter 3. The 3DBAG project offers three formats for downloading 3D building models, namely the GeoPackage (GPKG), the Wavefront OBJ format, and CityJSON. The 3DBAG project published a map of the Netherlands that has been divided into rectangular tiles. Each tile represents a defined bounding box, which refers to the Cartesian axis-aligned rectangle containing the tile's geometry. A user can download the building models that are located in the tiles representing the geographical area where the buildings reside in the physical world. We chose CityJSON for its structured representation and flexibility during preprocessing.

For example, in the case of the University of Amsterdam's Science Park campus, 3DBAG's tiling system divides Building 904 across two separate tiles. In OBJ format, the building is completely retained, protruding the borders of the rectangular area previously defined by 3DBAG. The CityJSON format, by contrast, removes all buildings that are not completely within the defined bounding box. However, the missing building can be restored with the help of CJIO tool, by merging two neighbouring tiles together. In the scope of this project, we have not attempted to use the GPKG format since there is no obvious file conversion path to suit the dataset into Sionna's supported XML file.

The CJIO tool, a Python-based library developed by the same team of researchers maintaining the 3DBAG project, is useful to pre-process the input datasets for improved fidelity of the simulation. The datasets released by 3DBAG has up to 10 different sizes, which are inversely proportional with the density of the buildings located in the area the corresponding tile represents. As such, a tile covering a densely built residential area in Rotterdam represents an area of 250000 m^2, whereas a tile that partially covers the sparsely populated Hoge Veluwe National Park covers 64 km^2. For the purposes of the dataset, we use the CJIO tool to clip all tiles to fit a uniform size of 128m $\times$ 128m. This size is chosen since the area roughly corresponds to coverage area of an FR2 cell in an urban environment. Additionally, 128m $\times$ 128m area in a densely built urbanised region covers enough geometric diversity to simulate realistic multipath effects, but small enough to run thousands of simulations within the bounds of compute resource limits.

The CityJSON files include information such as the location descriptions, land terrain of the selected area, and physical structures present within the area. 3DBAG uses the 'Amersfoort RD New + NAP height' standard for its geospatial reference system, instead of the more generalised longitude/latitude coordinates system. The standard, referred to as EPSG:7415 by geographic surveyors, placed the origin coordinates in the city of

Amersfoort, Netherlands, and its use is limited only for surveying within the Netherlands. The coordinates along the X- and Y-axes of the Cartesian plane-like in the map of the Netherlands follow the distance from the origin in metres. This provides an easy method of checking the correctness of all maps when measured with other 3rd party tools such as Blender, since the representation of distances between two objects can be scaled following the real-world distance, unlike the longitude/latitude coordinate system that is more useful when looking at a global-scale map but not intuitive for determining locations in a city-wide map.

4.4.2 Geometric Extraction and Metadata Creation

With the CityJSON files loaded as inputs, the pipeline creates a Metadata file for each input tile. The Metadata file include information such as the bounding box, which are the 2 pairs of coordinates denoted as *(xmin, ymin)* and *(xmax, ymax)* representing the southwest and northeast corners of the bounding box respectively. The bounding box coordinates are listed following the EPSG:7415 format, along with the EPSG:4326, which is the standard longitude and latitude coordinates system more commonly used by the general users. Using the bounding box information, the pipeline searches for antenna sites that are within the borders of the bounding box, with a further tolerance of 200m, to accommodate capturing 5G signal transmitted from further away and may possibly find its way into the bounding box. In the scope of this project, We retrieved a public list of licensed antenna sites from Antennebureau, the Dutch government's information agency responsible to educate the public on the laws and regulations of antenna placements.

4.4.3 3D Conversion and Assignment of Radio Materials

This stage converts the preprocessed 3D building geometry into a Mitsuba-compatible XML scene. A scene is the complete 3D environment derived from a tile that has undergone the preprocessing steps, and is ready to be loaded onto Sionna for ray-tracing simulation. The conversion include assigning ITU-compliant material properties to each surface. The choice to use Blender at this point in the pipeline follows the Sionna developers' recommended approach for preparing scene inputs. Sionna's ray-tracing engine requires Mitsuba XML, but neither Sionna nor CJIO offers a direct CityJSON-to-XML conversion. To bridge this gap, the pipeline first exports the preprocessed CityJSON tiles from CJIO into Wavefront OBJ format. While 3DBAG already provides pre-generated OBJs, those files cannot be

easily modified; merging tiles or cropping them to fit the chosen size would require invoking Blender much earlier in the process, introducing unnecessary GPU load during bulk preprocessing. In contrast, CityJSON is highly malleable; as a Python-based library, CJIO can efficiently merge, crop, and simplify geometry while retaining semantic attributes, and only once the data is finalised is it exported to OBJ.

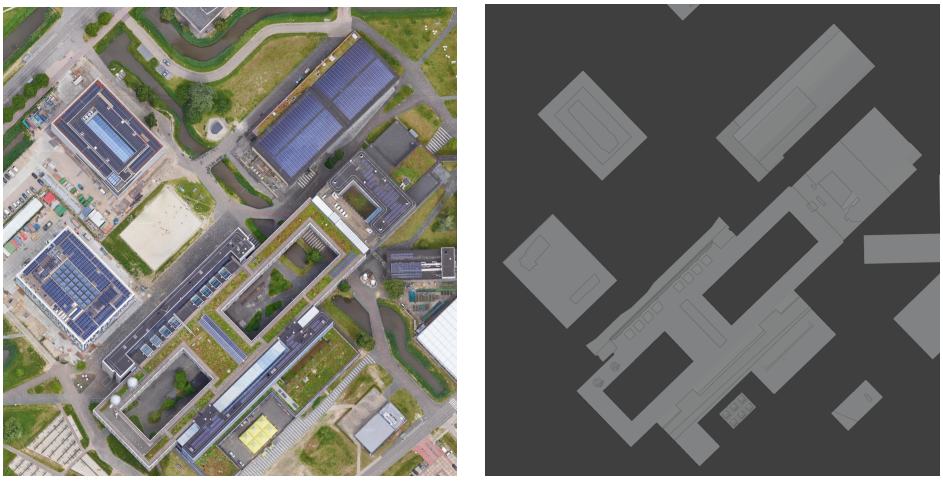

Figure 4.1: Side-by-side comparison of the Satellite View of the University of Amsterdam Science Park Campus (*left*), and the Blender 3D rendering of the tile (*right*). This illustration captures Building 904, Lab42, and Universum Sport Hall.

Since we use an input with an LoD of 2.2, the rendered OBJ file models building geometries with relatively high shape accuracy. However, this does not necessarily include surface material data. Since no comprehensive material database exists for all Dutch building surfaces, and the LoD specification does not distinguish different materials on the same building, any material assignment would be inherently approximate. We chose to assign all building surfaces the ITU-R P.2040 standard material concrete, on the rationale that concrete is structurally present in all buildings, even if not always externally exposed. As shown in Figure 4.1, the rendered 3D tile has a uniform building material, despite the University of Amsterdam's Science Park campus features a grid of solar panels on the rooftop, which could be more accurately assigned the ITU-R P.2040 glass material instead. To represent terrain, we add a flat plane covering all building footprints, positioned at the lowest footprint elevation in the tile. This mesh is assigned the ITU material medium-wet ground, reflecting typical Dutch soil and climate conditions.

After assigning all polygon meshes their respective radio materials, Blender exports the OBJ file into XML files with the associated .ply files using the Mitsuba-Blender library.

This step is essential, since Sionna can only load scenes from XML files. All of the steps where Blender is involved are executed programmatically, leveraging Blender's own bundled Python virtual environment, separate from the main pipeline's Python environment. This enables the entire stage to run in headless mode without a graphical interface, reducing GPU overhead and making it suitable for HPC or other server-based execution.

4.4.4 Signal Simulation

With the geographical scene packaged into Mitsuba-compatible XML and .ply files, the pipeline is ready to load the scene into Sionna for wireless ray tracing simulation. The pipeline focuses on three parameters that are particularly relevant for FR2 propagation analysis and machine learning-based path prediction:

1. **Received Signal Strength (RSS):** The expected signal power at each point in the tile, measured in decibel-milliwatts (dbm).
2. **Signal-to-Noise Ratio (SINR):** An estimate of link quality considering both environmental multipath effects (*e.g.,* reflections, scattering) and thermal noise.
3. **Path Loss:** The reduction in signal power over the line-of-sight path, isolating the geometric and environmental attenuation from other impairments

The simulation setup is aligned with Sionna's ray-tracing workflow as follows:

- **Scene description:** A Mitsuba XML scene is generated by Blender, with building geometry and material properties derived from 3DBAG and ITU-compliant material mappings.
- **Transmitter/receiver configuration:** The transmitter devices are positioned according to coordinates suppplied by the Antennebureau. The frequency of the signal transmitted by the devices is configured per scene. In the scope of this project, we randomise the exact frequency from the valid list of FR2 frequencies. The receiver devices are randomly positioned within the bounds of the scene, at a height of $\pm$ 1 metre from the terrain to emulate the behaviour of the typical mobile phone user.
- **Antenna model:** We assign a random antenna type to each transmitter devices, from the available choices of isotropic (#TODO). We also randomise the antennae's transmission patterns to introduce a variety of signal conditions in the dataset. The

antenna type and transmission patterns are then recorded back into the tile's respective metadata. The antennae are then configured to face random, various angles in the scene.

The simulation produces NumPy arrays for each parameter, which serve as the ground-truth data for ML training. To aid human inspection and preliminary validation, the pipeline also generates heatmaps of RSS and SINR over the tile, allowing quick identification of shadowing regions, strong-reflection zones, and coverage boundaries.

By the completion of this stage of the pipeline, a user may obtain the visual aids illustrating the heatmaps of perceived RSS and SINR for easier troubleshooting. The Figure 4.3 below is a sample of the heatmap we obtained when we simulated a tile covering the Weesperplein neighbourhood in Amsterdam, the Netherlands.

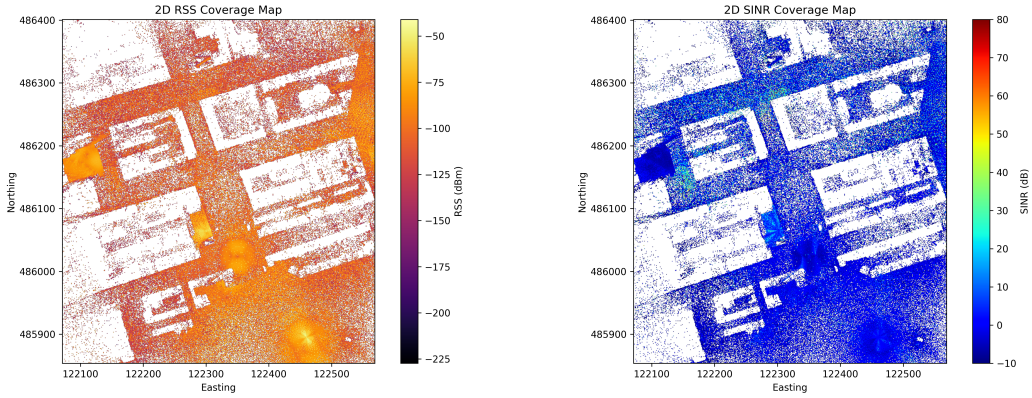

Figure 4.2: Sample of RSS (*left*) and SINR (*right*) coverage maps obtained at the end of the Signal Simulation stage.

4.4.5 Ground Truth Creation

To facilitate the training of a Machine Learning model from the .npy objects generated in the previous stage, the pipeline produces a cross-sectional 2D horizontal slice of each scene, highlighting potential obstructions such as buildings and other large structures. These are the dominant contributors to signal reflection, diffraction, and attenuation. The slice is taken at a user-specified height above local terrain and compared against the distribution of building heights in the tile. This slice is compared to the distribution of building heights in the tile. If the target height is below or equal to the 75^{th} percentile of building heights, the projection includes the full building footprints from the terrain up to the target height. Otherwise, the projection captures only the cross-sections of buildings intersecting that

height, with a ±2 m tolerance to account for modelling precision. This approach ensures that the resulting binary projection image represents relevant obstructions at the chosen altitude without including irrelevant structures above or below the slice.

The generated 2D images serve as geospatial ground truth for future ML training. However, neither the original input files nor the intermediate outputs include certain landscape features. Most notably, the omitted features include bodies of water such as rivers, canals, and coastlines. Without these, a model might incorrectly infer that the Dutch landscape consists exclusively of built-up and flat terrain. To address this gap, the pipeline includes an optional augmentation step that queries the Mapbox API to obtain unlabelled high-resolution satellite imagery for the scene's bounding box. These images provide visual context and improve completeness of the ground truth dataset. This step is disabled by default due to Mapbox's usage-based pricing, for which a user is to obtain their own API token key to enable this feature. In addition, we made an assumption consistent with ITU guidance, that FR2 cell coverage areas are small enough that water bodies have negligible impact on path loss. This contrasts with long-distance microwave backhaul links, where propagation over water can introduce refractive effects and additional losses over distances of up to 10 km.

Below is a sample of the satellite image view obtained from Mapbox and the 2D projection slice of the Weesperplein neighbourhood tile, showing consistency of the shapes of structures in a tile across different stages of the pipeline.

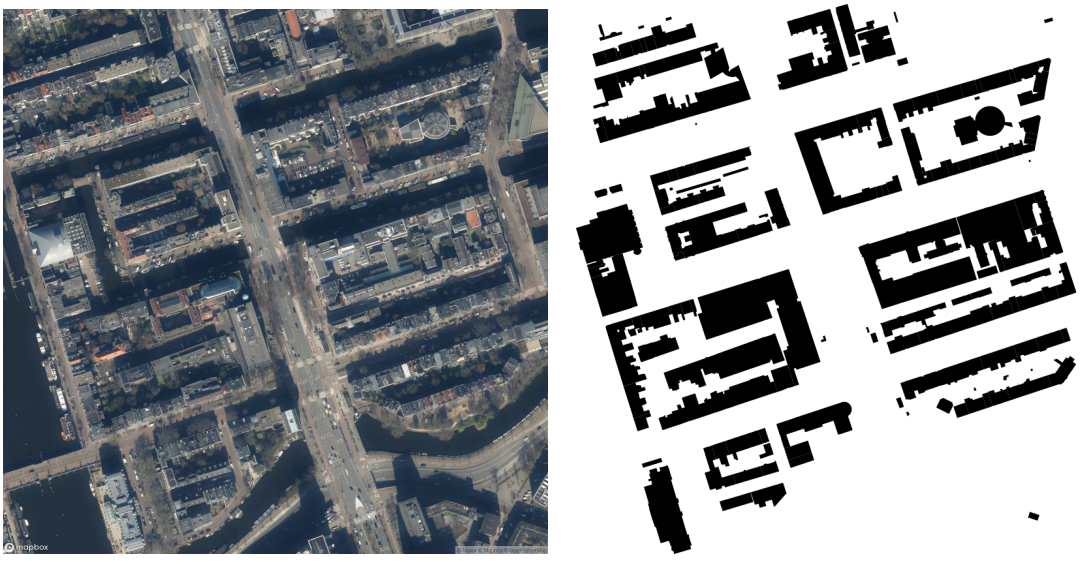

Figure 4.3: Sample of satellite view image (*left*) and 2D projection slice at a targeted height of 8 metres above the terrain (*right*) obtained at the completion of the pipeline.

5

Analysis of Pipeline Prototype Development

In this chapter, we recount the development lifecycle of the prototype, focusing on the design choices and decisions that shaped its current form. A key milestone in this process was the creation of an early implementation in which the completion of one stage triggered the execution of the next through an API call. We refer to this initial implementation as the **Alpha-version**. Components of the Alpha-version were subsequently refactored and integrated into the current iteration of the prototype, hereafter referred to as the **Kafka-version**. We shall begin with an evaluation of the Alpha-version, highlighting its characteristics, limitations, and lessons learned. This provides the basis for understanding the motivation behind the migration to the Kafka-version, which is then assessed in detail.

5.1 The Alpha-version

5.1.1 Batch-oriented File Processing

The Alpha-version was built wholly on Python without implementing any sort of message broker and queueing system. Each stage was assigned a dedicated directory to monitor, and is responsible to produce intermediate files into its corresponding directory. Each stage ends with checking whether there is an equal number of files between the directory to which it is assigned, and the directory which was assigned to the previous stage. This conditional check served as the completion signal to indicate the termination of the stage, ready to call for the next stage.

This design was simple to implement and to debug, but its scalability was quickly limited. With small inputs (*e.g.,* one CityJSON file), throughput was acceptable: processing a single

file into `.npy` objects took 39 seconds on DAS-6. However, the architecture forced each stage to wait until the entire batch of files was completed before proceeding. Consequently, no intermediate results were available until the full input set had finished, even though resources were actively being used. For larger inputs (tens to hundreds of CityJSON tiles), this batch-orientated architecture created the perception of inactivity and, more critically, withheld partial results that could otherwise have been exploited downstream. This behaviour conflicted with the broader design goal of integrating the pipeline into an ML training workflow. In iterative ML development, early access to even partial datasets is valuable: model training can begin on smaller batches while the remainder of the data continues to be processed. Alpha-version's architecture fundamentally delayed this process, and hence was the primary motivation for us to migrate the pipeline to Kafka-version.

5.1.2 Monolithic Execution Model

In addition to the scalability issues of the Alpha-version with respect to input size, it was also unable to efficiently leverage available computing resources. The DAS-6 environment separates the head node from the worker nodes where GPUs are installed. While processes on the head node can run for extended durations, processes on worker nodes are forcibly terminated after 15 minutes by DAS-6's job scheduling policy. This constraint imposed a fundamental testing dilemma: executing the pipeline on the head node avoided premature termination but prevented the use of GPU to accelerate 3D scene rendering and tensor computations, whereas executing on a worker node provided GPU access but with no guarantee of completing a task within the allotted time. The absence of such guarantees diminished the utility of the pipeline, as there was no assurance that any usable output would be produced before termination.

To mitigate this, we attempted to restructure the Alpha-version so that the main process ran persistently on the head node, delegating individual stages as child processes on worker nodes. While this approach initially appeared promising, it introduced race conditions: the system could not reliably ensure that an input file had fully completed one stage before being processed by the next. This arose because there was no runtime messaging mechanism between nodes to signal the start and completion of a stage. Since Python does not natively provide inter-node messaging, we briefly considered migrating the implementation to Julia, which was designed for parallel and distributed computing. However, such a migration would not resolve the more fundamental limitation of input scalability, and thus was not pursued further.

5.1.3 Reliability and Fault Tolerance Issues

Unit testing of the Alpha-version quickly revealed how the absence of recovery mechanisms translated into practical faults during execution. Individual dependencies behaved inconsistently when exposed to malformed or incomplete input: some failed gracefully by issuing termination signals, while others collapsed more severely, leaving the main pipeline in a stalled state with no indication of progress. These behaviours underscored not only the fragility of the prototype, but also how heterogeneous error-reporting conventions across the toolchain complicated fault detection.

5.1.3.1 Segmentation Faults Caused by CJIO

During our initial unit tests, we found that the CJIO tool utilised to export the JSON input files into OBJ files ready for rendering in Blender has a limitation. If the total number of vertices in the polygons of the produced OBJ file exceeds a certain threshold, the conversion could fail due to segmentation faults. We investigated further, and found the issue reproducible when the pipeline processed densely built tiles, such as those representing Rotterdam Centrum neighbourhood. Such densely built tiles could represent hundreds of distinct houses and buildings, which in turn are converted into hundreds of polygons constructed by thousands of vertices. On DAS-6's head node, we found that CJIO regularly failed when a tile has a total number of vertices exceeding a threshold of between 40 to 50 thousand. We have not tested further if the issue is reproducible in other computing environment, or whether the threshold would change.

Segmentation faults typically occur when a process attempts to access memory it does not own. In our case, we suspect the failure is related to excessive memory usage due to large polygon counts combined with Alpha-version's pattern of repetitive process spawning. Fortunately, this specific issue, when it occurs, would happen very early along the input tile's lifecycle in the pipeline prototype. A simple mitigation is to mark the tile causing segmentation error to its corresponding Metadata file, informing the next stages of the pipeline not to process the tile further. However, this mitigation would not be persistent should the pipeline is executed anew, as a new metadata would be created for all tiles in the input directory, overwriting the records of previous runs. As such, the segmentation fault would occur again, wasting the users' time and computing resources for tiles we already know will be doomed to fail. Another mitigation tactic might be to instruct the pipeline to delete the problematic tiles from the input directory altogether to avoid repeating future faults, but this comes with the cost of accountability and reproducibility.

5.1.3.2 Zombie Processes

The Blender software, used in the pipeline for 3D scene rendering, is resource-intensive and occassionally failed to terminate cleanly. Over time, "zombie" Blender processes could accumulate, consuming memory and slowing down the initiation of new Blender processes. This behaviour was observed as Alpha-version reached Stage 3, where the pipeline would read the OBJ files produced by Stage 2, initiate a Blender process, exported the input to Mitsuba-compliant XML files, and finally terminate the Blender process. We observed that as the number of input size increases, the time taken for DAS-6's head node to initiate a new Blender process for a new tile exponentially increased. Further analysis by observing `ps -aux` showed that some Blender processes initiated for previous input tiles may not be fully terminated, resulting in multiple unutilised Blender processes. As a temporary mitigation, we periodically monitored the head node's processes and issue `kill -9` commands to terminate such zombie processes. This mitigation step contradicts with our vision of a pipeline which requires minimum user supervision.

Spawning Stage 3 on GPU-equipped worker nodes could theoretically mitigate CPU stress, but the underlying root cause of failed process termination will remain unresolved. This mitigation tactic would also ultimately reintroduce the issues relating to the parallel node management as explained in subsection 5.1.2.

5.1.3.3 Process Recovery

The DAS-6's job scheduling policy highlighted another critical weakness of the Alpha-version: it had no mechanism to resume from partial progress after unexpected termination. Such recovery is essential not only for compliance with the cluster policies but also for resilience against random process failures or infrastructure downtime. Although per-file metadata tracked the last completed stage, Alpha-version's batch-oriented design meant reprocessing was required from the beginning. For small runs of 5 input files, this was manageable, but is prohibitive for a pipeline expected to process hundreds and thousands of input files for a complete simulation covering the entirety of the Netherlands. An easy workaround against this issue was to simply disable the completed stages in the main pipeline script, so the pipeline would jump into the stage where it can proceed from the last runtime. But this solution is insufficient for a self-aware pipeline to be ran with minimum user intervention.

5.2 The Kafka-version

The limitations of the Alpha-version, including lack of scalability, resilience, and recovery, motivated a fundamental redesign. The Kafka-version introduces a message broker (Apache Kafka) as the backbone of the pipeline, replacing batch file checks with asynchronous, event-driven communication between stages. The scripts for the different stages from the Alpha-version were refactored into the Kafka-version, where each stage is called upon receiving successful message from the previous stage. This migration provides message persistence, decoupled execution, and greater fault tolerance, enabling the pipeline to scale to national-level simulations.

5.2.1 Transition to File Streaming

With Kafka's message broker available as a communication channel between all stages, the conditional check of accounting files to proceed to the next stage is no longer necessary. Stages now operate on individual messages representing input tiles, allowing incremental outputs to be produced as soon as they are available. This aligns the pipeline with ML workflows, where training can begin before all data is processed.

This architectural shift, however, introduces a trade-off. In the Alpha-version's batched processing model, errors typically occurred in a uniform, "all-or-nothing" manner: if one input file triggered a fault, the entire batch would fail. While this was inefficient, it made debugging relatively straightforward, since the failing stage could be identified and addressed directly. By contrast, in the Kafka-version, faults can occur unevenly, with some tiles progressing through all stages while others remain stuck at an intermediate stage. This non-uniform behavior complicates post-mortem analysis and requires more fine-grained logging to trace silent failures at the level of individual files.

5.2.2 Distributed Processing

By decoupling the pipeline stages into independent Consumer scripts, each running as a child process, we enabled distribution across multiple worker nodes, with Kafka ensuring reliable delivery and ordered processing within partitions. This design allows the main pipeline process to run on the DAS-6 head node, while offloading compute-intensive child processes to GPU-equipped worker nodes. Each Consumer script performs a lightweight resource check at startup using TensorFlow: if a GPU is available on the assigned node, the GPU would be utilised; otherwise, the script defaults to CPU execution.

This strategy resolved two limitations of the Alpha-version. Firstly, the pipeline is no longer constrained to run entirely on either the head node or the worker nodes, eliminating the earlier deployment dilemma. Secondly, GPU-aware Consumers increase the pipeline's adaptability, allowing repackaging of the prototype for diverse compute environments, including HPC clusters, virtual machines, or containerised deployments.

5.2.3 Fault Tolerance and Recovery Mechanisms

Although the Kafka-version improves throughput and decoupling, recovery after failures remains incomplete. The current design maintains a single `Manifest.json` written by the Producer, recording for each input the last completed stage and whether processing should continue. If a stage reports `status = failed`, the input is excluded from subsequent stages. This prevents cascading failures within a run but does not persist reliably across runs. Because the tracker file is regenerated at pipeline start-up, all prior failure annotations are overwritten, leading to repeated attempts on inputs that are known to fail.

The core limitation lies in using a single JSON file as a shared job tracker. JSON offers no native support for atomic updates: a system crash during a write can produce duplicates or "stuck" jobs. Concurrent access exacerbates the problem, since read-write contention risks corruption. To mitigate this, access was restricted to the Producer alone, but this introduces indirect and delayed progress updates from other stages. A file lock mechanism was briefly considered and tested, but we eventually discarded this idea. While file locks prevent simultaneous writes, it accumulate delays when multiple nodes attempt updates, and still cannot guarantee atomicity. Consequently, the JSON-based tracker remains a bottleneck, insufficient for robust recovery in a distributed environment.

5.3 Ansible for Portability

Introducing Apache Kafka further added to the mounting list of dependencies, which could complicate portability and ease of use for future users. The current prototype was developed and tested on DAS-6, with processes executed by invoking individual Python scripts through multiple terminal sessions. While functional, this approach is not easily accessible to future users without extensive documentation. Some dependencies, which are discussed in further detail in chapter 6 require non-standard procedure to download and install. For example, obtaining the Blosm dependency involves registering through a shareware portal and receiving a unique download link, a step that cannot be reliably automated or redistributed.

To improve portability, we have added Ansible automation script to assist in provisioning the deployment environment and installing the required dependencies. This significantly reduces the installation burden for future users, with the exception of Blosm, which must still be obtained manually. The decision to use Ansible was particularly motivated by Blender's non-trivial addon installation procedure. While installing addons such as Blosm and Mitsuba-Blender is straightforward through Blender's graphical interface, our headless deployment on DAS-6 required a more complex process. Specifically, addons must be manually placed into Blender's hidden configuration directory, and additional steps must be performed via Blender's embedded Python environment to register them. The automation script encapsulates this process, making the pipeline reproducible in headless environments.

5.4 Comparative Summary

We outline the key differences between the two implementation versions, highlighting the motivations for migration and the improvements achieved.

Table 5.1: Architecture comparison of Alpha and Kafka-versions of the pipeline prototype

Metric	Alpha-version	Kafka-version
Testability	**Easy:** The batch-oriented file processing architecture means a fundamental error or fault occurring in one stage tends to propagate across all input files. This makes faults more visible and reproducible, simplifying root-cause analysis.	**Moderate:** Since files are processed independently, errors may occur silently on a per-file basis without immediately halting the pipeline. While this increases resilience, it reduces transparency.
Deployment Simplicity	**Easy:** The architecture requires minimal dependencies, with straightforward directory-based workflow.	**Medium:** The architecture requires relatively large one-time dependency installation effort, where configuring Apache Kafka and Zookeeper requires its own learning curve.

Continued on next page...

Metric	Alpha-version	Kafka-version
Scalability	**Poor:** The architecture couples all stage scripts to the main process, preventing offloading to GPU-equipped worker nodes and limiting parallelism.	**Excellent:** Kafka enables inter-node communication, allowing stages to run on separate processes across different compute nodes. This architecture supports coherent, distributed scaling across heterogeneous resources.
Fault Tolerance	**Weak:** Unexpected system failures or incomplete runs require reprocessing of already-completed stages, wasting both compute time and user effort.	**Unclear:** Incorporating lightweight databases (*e.g.,* SQLite) for per-file stage tracking could potentially improve resilience, but this mechanism is not yet implemented.
Supervisory Overhead	**High:** Zombie processes and some faults that may cause the main process to hang means it is up to the user to monitor the pipeline's behaviour periodically.	**Low:** The pipeline is sufficiently reliable to terminate "hanging" processes and keep track of each input's progress, thereby minimising the need for user's intervention during runtime.
Portability	**Limited:** The pipeline requires execution on a single kernel environment. This rigidity leads to resource contention for large inputs, posing issues in constrained compute environments (*e.g.,* personal laptops).	**Promising but unverified:** Decoupling stages into child processes (and potentially LXC containers or cloud-deployed services) suggests high portability across environments. However, additional testing is needed to benchmark practical scalability ceilings.

As shown in Table 5.1, the Kafka-version improves scalability and reduces the need for user supervision. However, fault tolerance remains only partially addressed due to the limitations of JSON-based job tracking. These trade-offs demonstrate that while Kafka resolves the major bottlenecks observed in the Alpha-version, further work is needed to harden recovery and accountability mechanisms.

6

Analysis of External Dependencies

In this chapter, we analyse how the external dependencies utilised in our prototype helped shaped the pipeline, and discuss what limitations does each dependency bring. Where appropriate, we consider other alternatives to the dependency, while discussing what trade-offs will be involved between the chosen dependency and the suggested alternative.

6.1 Evaluation of Input Dataset Quality

6.1.1 Structural Coverage of 3DBAG and Implication for Telecommunication Research

While 3DBAG provides a highly detailed representation of Dutch buildings, it is important to note its scope limitations when applied to wireless propagation research. 3DBAG'S design focus is on buildings with registered postal addresses, ensuring a high level of consistency and reliability for urban studies and cadastral applications. As such, other civil infrastructure without registered address such as bridges, overpasses, and viaducts are excluded. This omission introduces challenges for our application, where large reflective or obstructive objects that can significantly influence radio propagation are not represented. For example, the Figure 6.1 illustrates the Erasmus Bridge in Rotterdam, which is omitted in 3DBAG but is represented in the Municipality of Rotterdam's 3D digital twin project. The absence of the bridge in 3DBAG will influence the fidelity of simulation, due to its prominent structural and electromagnetic impact on the surrounding environment.

Similarly, canals and waterways are absent from 3DBAG. Although this omission is logical given the dataset's building-centric scope, it reduces fidelity in scenarios where water bodies can affect propagation through reflection or absorption. Such missing features may lead to underestimation of multipath effects in urban simulations.

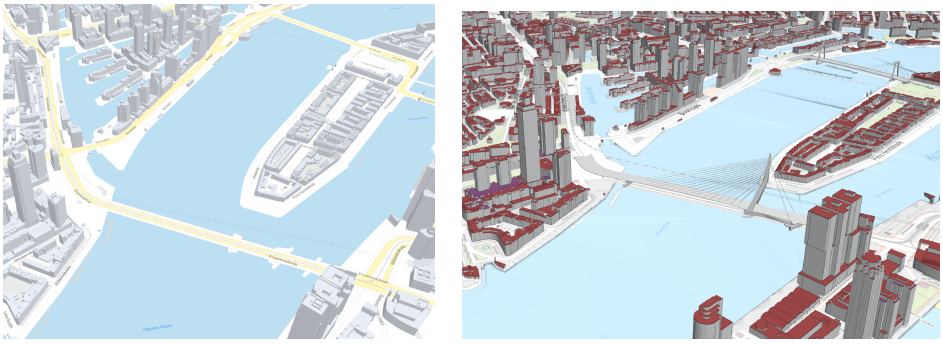

Figure 6.1: Side-by-side comparison of 3D object availability in the datasets provided by 3DBAG (*left*) and 3DRotterdam project (*right*).

6.1.2 OpenStreetMap as Alternative to 3DBAG as Input

An alternative to 3DBAG would be to use OpenStreetMap (OSM), which offers worldwide geographic coverage and includes both terrain and building geometries. This makes OSM attractive for extending the simulation beyond the Netherlands. However, OSM frequently lacks accurate or complete building height information, which can introduce errors in line-of-sight and path loss calculations.

To improve building height accuracy, OSM data can be supplemented with external sources such as the Microsoft Building Footprints dataset, which provides information on 1.4 billion building footprints and estimated heights derived from satellite imagery. While this increases global coverage, the inference-based nature of the data introduces uncertainty, particularly in dense or complex urban environments. Using OSM would also require the integration of third-party tools such as Blosm, a Blender add-on for extracting OSM data. This introduces additional dependencies into the pipeline. In contrast, 3DBAG's standardisation and consistency make it highly convenient for a prototype with a national focus.

There are other data sources available, such as CityGML format to represent major metropolitan areas. These datasets are typically maintained by the local governments and councils, and may lead to possible improvements to the prototype in the future.

6.1.3 Coverage and Completeness

As a maintained dataset covering a national scope, 3DBAG covers both densely-populated metropolitan areas and rural environments. As such, the input data set quality is guaranteed to be uniform, whether the user is simulating signal propagation in built-up areas

such as The Hague or farmlands of Het Groene Hart. This comes with the trade-off, that tiles where there are no buildings are completely omitted, unavailable to be downloaded. This makes sense given 3DBAG's primary focus, but can lead to silent absences in an automated workflow such as our prototype.

The OSM dataset is crowdsourced and has a global scope, but the quality of maps representing different regions are heterogeneous. In dense cities, completeness of the OSM dataset can rival official datasets. In less urban or less digitally active regions, buildings may be sparse, misaligned, or missing entirely. The Microsoft Building Footpritns dataset, although not crowdsourced, featured the same limitation, as the building height information is much more complete and detailed for buildings in North American and European regions as compared to Asian and African regions.

6.1.4 Missing Terrain Information

A further consideration concerns the treatment of terrain. 3DBAG employs the 'EPSG:7415 Amersfoort / RD New + NAP height' coordinate system, which provides accurate georeferencing of building footprints, including their elevation relative to sea level. In contrast, OSM data exported through Blosm is centred at the origin of a Cartesian coordinate system, complicating the alignment of additional data sources such as antenna placements.

Despite these differences, both 3DBAG and OSM share a key limitation: neither explicitly represents natural terrain features such as elevation changes, water bodies, or vegetation. This omission has minimal consequences in environments with relatively flat topography, such as most of the Netherlands, but becomes highly consequential in regions with pronounced elevation variation. In cities such as Rio de Janeiro, Brazil, or La Paz, Bolivia, steep gradients can introduce substantial elevation differences over short distances. A hilltop or a cliff could be considered as a conducive terrain choice to erect a gNodeB site, but the coverage area served by the site would be highly dependent on the antenna configuration (*e.g.,* azimuth, orientation, and terrain obstruction).

In our prototype implementation, scenes with extreme elevation changes over short distances can cause gNodeB positioning errors, resulting in transmitters that appear to float unrealistically above the built environment. Without terrain elevation data, the system unrealistically overestimates coverage areas by simulating elevated transmitters as having unobstructed 360° line-of-sight coverage. In reality, terrain irregularities would block or attenuate significant portions of these signals. Figure 6.2 illustrates this issue through a comparison of La Paz's urban contours rendered with and without terrain information.

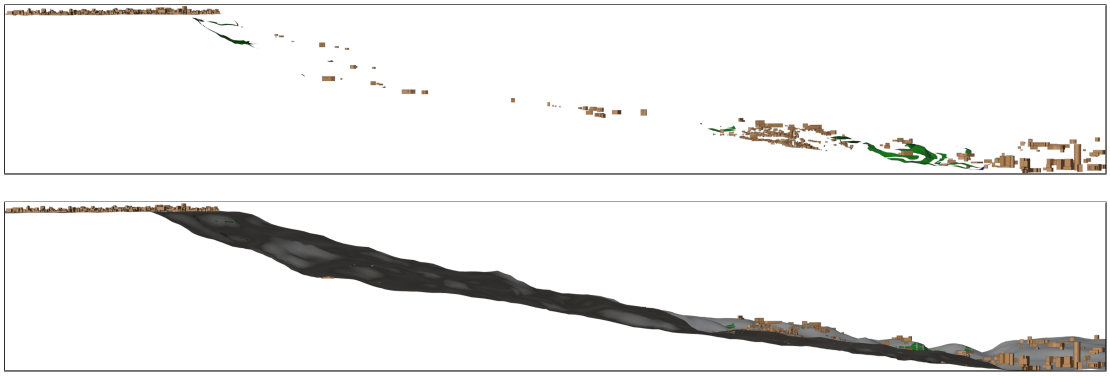

Figure 6.2: Comparison of the side-view contours of La Paz, Bolivia, where terrain information is unavailable (*top*) and where terrain information is available (*bottom*)

This rendering shows La Paz viewed from south to north, with the El Alto plateau visible on the left and the Obrajes neighbourhood on the right. Brown cubes represent buildings and structures. The comparison demonstrates how terrain integration affects visibility: when terrain data is available, natural topography obscures certain buildings from view, creating realistic line-of-sight limitations. Without this terrain information, most structures appear to have direct line-of-sight with each other, leading to overoptimistic coverage predictions.

Even when a user is simulating within the Netherlands, the absence of terrain modelling can introduce small but perceptible inconsistencies. For example, land reclaimed from the sea (referred to in Dutch as *polders*) often lies several metres below surrounding water levels and can feature subtle elevation differences. In our Stage 3 prototype, where Blender introduced a flat two-dimensional plane aligned to the lowest Z-coordinate of the tile in order to emulate the ground, building footprints did not always share a uniform base height. As a result, some buildings were rendered as if slightly elevated above the ground plane. While deviations of ± 3 metres in areas such as Almere are negligible compared to the extreme elevation changes of La Paz, they nonetheless highlight a potential misrepresentation of reality. Future applications requiring higher-fidelity datasets may therefore need to incorporate terrain data to improve accuracy.

6.2 CJIO as File Converter

As described in chapter 4, the CJIO tool was incorporated into our pipeline for two main purposes: (i) filtering 3DBAG's CityJSON files by bounding box and level of detail, and (ii) converting the CityJSON format into OBJ for downstream processing. In practice, we

observed that CJIO occasionally encountered segmentation faults during the conversion stage. Inspection of CJIO's source code revealed that the tool first enumerates all vertices prior to transforming the CityJSON structure. We discovered that failures were most likely to occur on densely built tiles containing more than ±40,000 vertices, suggesting that the conversion step is memory-intensive and prone to exceeding allocated memory resources under these conditions.

This behaviour posed significant challenges in our Alpha-version prototype as detailed in chapter 5, where CJIO was executed as a subprocess in Stage 1. A segmentation fault in this setting would terminate the subprocess without propagating an error back to the parent process, leaving the pipeline in a indefinite stalled state. No reliable runtime mitigation was possible, which contributed to the motivation for adopting the more fault-tolerant Kafka-based architecture.

A pragmatic workaround was later introduced by preprocessing the input tiles. Whereas the smallest native 3DBAG tile covers approximately 500 m × 500 m, subdividing tiles into 128 m × 128 m bounding boxes significantly reduced the likelihood of segmentation faults by lowering the simulation's coverage area, thereby lowering the number of vertices processed in a single run. Coincidentally, this subdivision step was also required by the downstream ML training pipeline, making it a natural fit within our workflow. Nevertheless, this finding highlights an important limitation: while merging tiles into larger simulation regions may appear convenient in certain scenarios, it risks reintroducing instability during runtime and should therefore be approached with caution.

6.3 Sionna as Simulator Engine

Sionna constitutes a central component of our pipeline, providing a highly customisable framework in which users can adjust parameters to increase simulation fidelity. Nevertheless, we identified several limitations relevant to urban-scale wireless modelling.

Firstly, Sionna applies certain transmitter parameters (specifically transmitting power and frequency) uniformly across all devices in a scene. In dense urban environments, this assumption does not reflect operational practice. For example, an antenna site in Amsterdam may host multiple gNodeBs, each equipped with transmitters configured differently in terms of azimuth, orientation, power, and frequency. Our inspection of Antennebureau records revealed that a 500×500 m tile of an Amsterdam neighbourhood contained up to 23 licensed transmitters. In reality, operators rely on heterogeneous configurations to achieve

differentiated Quality of Service (QoS), mitigate path loss, or comply with regulatory spectrum allocations. By treating all transmitters as identical, Sionna risks underrepresenting interference dynamics (such as crosstalk) and over-simplifying the propagation environment. This simplification constitutes a major barrier to adoption by mobile operators, as it reduces the realism of propagation modelling and undermines the simulator's ability to reproduce conditions encountered in real-world networks.

Secondly, we observed that Sionna's material library does not include water as a predefined medium for electromagnetic propagation. This omission arises from its reliance on ITU-R P.1238, which does not prescribe conductivity or permittivity constants for water. While Sionna allows users to manually define additional materials, the absence of native support for water introduces challenges when modelling environments containing significant surface water, such as coastal areas or river-dense cities. In such cases, additional empirical calibration would be required to ensure accurate propagation characterisation. Notably, this mirrors a limitation we observed in the 3DBAG and OSM dataset, which also omits water bodies from its representation. The fact that both the geographic source and the simulator share this absence underscores a broader challenge: ensuring that the tools and data used for wireless system evaluation align with the full range of conditions present in operational networks.

The composite Figure 6.3 illustrates the effect and consequences of omitting features including bridges and bodies of water. These figures were obtained when we simulated the area near the mouth of the Rijnkanaal, featuring the section of the A10 Ring Road, where the Zeeburgerbrug bridge connects the Watergraafsmeer neighbourhood in the south and Zeeburgereiland in the north. It is noticeable that the canal, bridge, and the Buiten-IJ lake are missing from the 3D scene rendering and therefore are not taken into account during the signal propagation simulation.

In summary, while Sionna provides a flexible foundation for research-oriented simulation, its current design choices limit its fidelity in scenarios requiring heterogeneous transmitter configurations or accurate modelling of water surfaces. These issues highlight areas where further extensions would improve its alignment with real-world operator requirements.

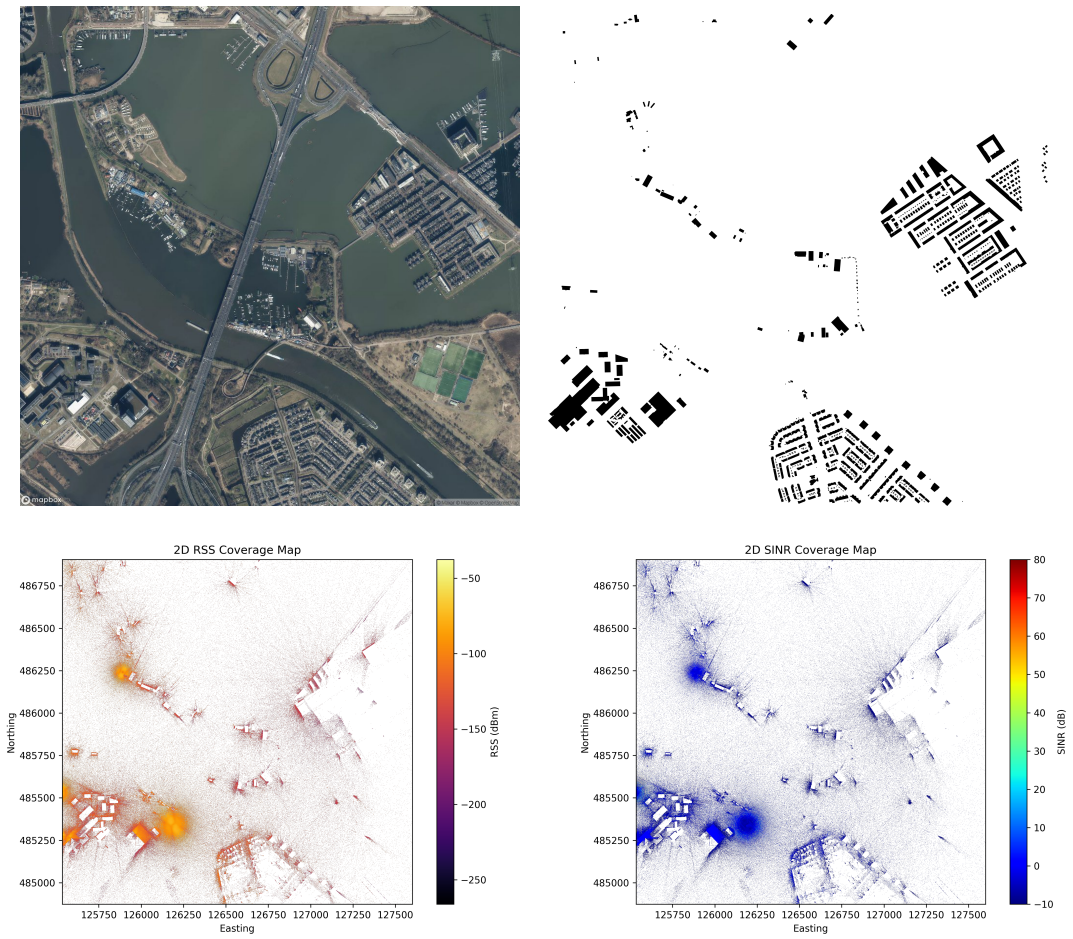

Figure 6.3: Composite of figures generated by the pipeline when simulating area around the mouth of Rijnkanaal, showing the absence of the Zeeburgerbrug and surrounding bodies of water *Clockwise from the top left: satellite image view, 2D projection slice of the area, RSS and SINR coverage maps.*

7

Future Work

This chapter outlines potential extensions and refinements to the prototype. These include mitigation strategies that are yet to be tested and implemented for the limitations identified in previous chapters, as well as new features to improve accuracy, fidelity, and scalability. We also highlight directions for extending the utility of the pipeline and the datasets it generates.

7.1 Upgrading Dependencies

Like many research prototypes, the pipeline relies on a frozen set of software versions chosen at the time of development. While this ensured stability during experimentation, it also created technical debt: several dependencies are already lagging behind their current stable releases, constraining compatibility and long-term sustainability. The current prototype relies on legacy versions of several dependencies, primarily dictated by the use of Sionna v0.19.2. This release supports only Python 3.10, which has in turn constrained the compatibility of other libraries such as TensorFlow. While this choice was necessary at the outset of the project, it is not sustainable. Migrating the pipeline to the latest stable release of Sionna (v1.1.0) is essential to ensure long-term maintainability. The urgency of this migration is underscored by the planned end-of-support (EoS) for Python 3.10 in October 2026, after which downstream dependencies may no longer provide security patches or functional updates, creating risks for reproducibility and deployment.

Another critical dependency is Apache Kafka, which serves as the message broker between the pipeline's nodes. For this prototype, Kafka v3.7.0 was adopted rather than the most recent stable release (v4.0.0). The rationale was pragmatic: v3.7.0 represents the last stable version using ZooKeeper for node membership and consensus, while subsequent

releases replace ZooKeeper with KRaft, a Raft-based consensus mechanism. Although KRaft promises improved scalability, v3.7.0 was selected for its simpler learning curve and smoother integration during development. As Kafka functions primarily as a subprocess orchestrator and does not directly intervene in data processing, upgrading to a KRaft-enabled release can be deferred to future iterations.

In addition, the pipeline currently depends on third-party Blender addons whose compatibility is limited to specific versions. For instance, as of August 2025, Blosm is supported only in Blender 4.0-4.3, while the Mitsuba-Blender exporter has been validated up to Blender 4.4. Although Blosm is optional, where it is required only when extracting geospatial data from OSM as the pipeline's input, its constraints effectively freeze the development environment to older Blender releases. Upgrading to the latest long-term support (LTS) version 4.5 is not recommended, since Mitsuba-Blender addon has not yet been tested against it. Future work should therefore include systematic migration planning for both Blender and its addons, to reduce reliance on frozen versions and enable a more flexible and sustainable software stack.

7.2 Input Format Interoperability

The current pipeline processes `CityJSON` input files through two intermediate conversions: `CityJSON` to `OBJ`, and `OBJ` to Mitsuba-compliant `XML`. A future extension would be to implement direct ingestion of intermediate formats. For instance, if an `OBJ` or `XML` file is provided as input, the pipeline could bypass the preceding stages and enqueue the file directly to Stage 2 or Stage 3. This would reduce redundant processing and increase flexibility for users who already maintain assets in these formats.

7.3 Input Source Interoperability

A natural extension of the pipeline concerns interoperability with additional input sources. While the current prototype has focused on publicly available outdoor geodata, future applications will require the ability to ingest both urban-scale digital twins and highly specialised indoor environments. These two domains present different challenges and opportunities, which are outlined below.

7.3.1 Outdoor Scenes

Beyond format handling, a key challenge for future work lies in supporting additional sources of urban geodata. Several municipalities are actively developing 3D digital twins, such as Rotterdam's *3DRotterdam* project, or London's *London Datastore*, and similar projects across Europe. Incorporating these datasets would extend the relevance of the pipeline beyond the Netherlands.

However, such integration poses two primary challenges. Firstly, digital twins are published in a wide variety of file formats, ranging from geospatial standards such as CityGML and ArcGIS to general-purpose formats such as OBJ. While OBJ is widely adopted, its lack of semantic richness makes it unsuitable as a native representation of urban environments. Conversions from semantically rich geospatial formats to OBJ may result in loss of information, such as vegetation or land-use attributes. To support more formats beyond CityJSON, OBJ, and XML as inputs, additional dependencies would need to be incorporated into the pipeline. Tools such as the CityGML2OBJ library, developed at the Technical University of Munich (9), provide one path toward bridging this gap.

Secondly, coordinate system heterogeneity introduces alignment risks when merging datasets across regions. For example, the 3DBAG project uses EPSG:7415, which is specific to the Netherlands, whereas other digital twins may adopt the more globally recognized EPSG:4326 (latitude/longitude). Conversions between these systems are supported by libraries such as `Pyproj`, but must account for differences in underlying datums. Neglecting datum shifts can cause geographic drift on the order of $\pm$10m, potentially leading to misaligned structures in scenarios where users would like to simulate a scene in an area across the borders of the Netherlands and Germany, where the latter does not use EPSG:7415 in their geographical datasets. There are more extreme cases arising from different coordinate systems: China mandates the use of the GCJ-02 coordinate system, which introduces offsets of up to 700m relative to EPSG:4326. This discrepancy, historically visible in the misalignment of map labels and satellite imagery in China, underscores the importance of carefully managing coordinate transformations in any globally interoperable simulation pipeline.

7.3.2 Indoor Scenes

Signal propagation in indoor environments has received comparatively little attention in the research community, despite its growing importance in the telecommunications industry. A key driver is the emergence of *Private 5G as a Service* (P5GaaS), offered by mobile

operators to enterprises operating in enclosed or expansive facilities such as manufacturing plants, mining sites, harbours, or military bases (10). In such contexts, conventional outdoor gNodeBs may provide insufficient coverage or raise concerns over data privacy. P5GaaS addresses this by deploying dedicated gNodeBs that operate on isolated frequency bands, with users receiving devices locked to those bands. This ensures both improved coverage and a degree of separation from the public mobile network.

Extending the prototype to cover such scenarios requires the generation of indoor-specific datasets. However, publicly available indoor datasets are scarce and often unsuitable: most consist of LIDAR scans of residential houses, whereas industrial-scale sites, which are the primary targets of P5GaaS service offerings, remain inaccessible. Privacy and security considerations largely explain this absence, as operators of sensitive facilities are unlikely to release detailed scans, regardless of nondisclosure agreements. Some curated datasets do exist, but these are typically gated behind administrative restrictions and therefore unavailable for incorporation within the timeframe of this project.

That said, it is possible for dedicated users to obtain their own input representations of indoor scenes. One approach is to employ a robot equipped with LIDAR to scan the desired facility, producing point clouds that can be converted into an `.obj` file. Alternatively, computer-aided design (CAD) software such as AutoCAD or SketchUp can be used to manually construct a 3D representation of a facility and export it into a compatible format. While these approaches lie beyond the scope of the present work, they illustrate potential pathways for future users who wish to simulate custom indoor environments.

7.4 Input Transience

While the current pipeline focuses on spatially accurate simulations, it implicitly assumes static conditions for both transmitters and receivers. In real-world deployments, however, environmental and user-related factors often exhibit temporal variability that can significantly influence propagation characteristics. For instance, human users frequently act as receiver devices yet may move continuously across streets or within buildings, creating dynamic propagation paths. Similarly, vehicles or temporary structures can introduce additional sources of reflection or obstruction over time. Sionna as our simulation engine already provides support for time-evolution parameters (*e.g.,* velocity and orientation) for both nodes and scatterers, enabling the modeling of Doppler effects and time-varying multipath conditions. Incorporating these temporal parameters would bridge the gap between

the current static modeling assumptions and the dynamic nature of real-world wireless environments.

A second limitation concerns the temporal validity of geospatial input datasets themselves. The 3DBAG input datasets used in this pipeline provide accurate structural representations at the time of data collection but remain inherently frozen until the next release cycle. Ongoing construction or demolition projects may thus create discrepancies between the modeled environment and the real world, particularly in rapidly evolving urban areas. This limitation underscores the need for input-source interoperability: the pipeline should support seamless integration of alternative or real-time data sources when primary datasets become outdated or unavailable. Such flexibility would enable the simulation framework to remain operationally relevant even as environmental conditions evolve over time.

7.5 Incorporating Database as Stage Progress Tracker

Reliable tracking of pipeline progress is crucial for correctness and recovery. In the current design, progress is logged through a shared file (`Manifest.json`), which introduces both performance and fault-tolerance limitations, as we have previously discussed in chapter 5. Relying on constant status updates by reading and writing the tracker file `Manifest.json` introduces risk to the file's correctness and integrity, especially when it is accessed by multiple nodes simultaneously. At the moment, the tracking file's safety is guaranteed owing to the fact that only the Producer node has access to the file. Regular documentation on JSON file works under a single-writer and low-scale conditions, hence the tracker has a single-point of failure in an otherwise parallelisable workload. Databases on the other hand allows safe concurrent writes, transactional integrity, and better recovery.

Migrating the tracker to a database can also help solve the issue of recovering the pipeline's in the event of system failure. If the pipeline has been forcefully terminated, for example due to power failure, in the middle of processing thousands of files, the Producer will have to restart thousands of file processing, each at different stages. A single node executing multiple I/O read operations to relaunch stages is a bottleneck, but at the same time, the Consumer nodes cannot be trusted to read the tracker file and relaunch their own processes. The atomic behaviour of SQLite, which rejects read and update wholesale if one operation is found to be invalid, will help creating a more robust recovery system.

7.6 Pipeline Packaging for Reusability and Reproducibility

At its current shape, executing the prototype involves launching multiple scripts over multiple terminal windows manually. This deployment is neither user-friendly nor scalable. A more robust packaging solution could be implemented to streamline the deployment process. An easy first step may be to create a single wrapper script to launch the necessary components with a customisable flag to determine the number of Consumer nodes to be launched. Further development could involve containerisation using Docker, providing operating system agnosticism and a simplified, single entry point deployment. The container could include all dependencies, precluding the need of configuring Ansible for each new machine. The container could then be deployed on a public cloud for an even larger scale of simulation inputs.

8

Discussion

In this chapter, we shall revisit the Research Questions (RQs) outlined in chapter 1, discussing how our findings provide answers to these questions. We will also reflect on the implications of our results, the limitations of our study, and potential avenues for future research.

To address **RQ1**, we have implemented a prototype pipeline and documented the rationale behind key design decisions, along with the lessons learned during development, as presented in Chapters 5 and 6. Building on these insights, we now frame our recommendations for a pipeline suitable for both academic and professional use by evaluating its desired characteristics against the ISO 25010 software quality attributes. This standard provides a structured framework for assessing software quality, enabling us to highlight where the current implementation meets essential requirements for generating 5G signal-propagation datasets and where future iterations can be strengthened.

While the prototype has reached a stage of minimum viability, all future improvements as outlined in chapter 7 must be iteratively tested against the criteria in Table 8.1 to ensure both resilience and adaptability. With iterative feedback, continuous integration, and continuous deployment (CI/CD) of the underlying dependencies, such as those mentioned in chapter 6, we anticipate steady improvements in both the pipeline and the resulting ML models over time, enabling the pipeline to serve a wide variety of purpose and users with diverse objectives.

Table 8.1: Pipeline architecture considerations defined as per ISO 25010 software quality characteristics

ISO 25010 Sub-characteristic	Explanation to Consideration & Assessment Criteria
Functional Suitability: This characteristic evaluates whether the pipeline provides the essential functionalities required for generating signal characterization datasets from geospatial inputs.	
Functional Completeness	The pipeline implements the following core stages: input parsing, pre-processing, simulation, ground-truth generation for baseline comparison, and dataset export. Future implementations targeting similar objectives may adapt these criteria to suit their domain-specific needs.
Functional Correctness	The pipeline's outputs should remain consistent with real-world measurements where applicable. They should also exhibit deterministic behavior across different runtime environments, avoiding variability caused by non-deterministic factors.
Performance Efficiency: This characteristic considers the pipeline's responsiveness and resource usage under varying execution environments.	
Time Behaviour	Processing times for input files should remain within reasonable bounds and degrade gracefully as input sizes grow, without introducing significant bottlenecks during batch processing.
Resource Utilization	The pipeline should demonstrate measurable performance gains when granted access to more powerful hardware, such as multi-node HPC systems or GPUs with higher compute capacity.
Compatibility: This characteristic measures how effectively the pipeline integrates within larger workflows and interacts with heterogeneous data sources or tools.	
Co-existence	The pipeline should integrate seamlessly into downstream workflows (*e.g.,* ML training pipelines, simulation frameworks, or NDT platforms) without causing failures or requiring substantial rework.
Interoperability	The pipeline should accept diverse input sources (e.g., multiple geospatial formats, antenna databases) and produce outputs in formats suitable for varied downstream objectives.

Continued on next page...

Table 8.1 (continued): Pipeline architecture considerations

ISO 25010 Sub-characteristic	Explanation to Consideration & Assessment Criteria
Reliability: This characteristic reflects the pipeline's ability to maintain correct and consistent operation over time, including its resistance to crashes, handling of runtime faults, and overall stability under varying workloads.	
Maturity	The pipeline is expected to achieve consistent performance over extended runs. Metrics such as Mean-Time-to-Failure (MTTF) and Mean-Time-Between-Failure (MTBF) can serve as indicators of robustness and resilience. Future iterations could incorporate automated logging of these metrics to guide improvements.
Fault Tolerance	The pipeline should detect and handle runtime faults (*e.g.,* missing or corrupted inputs) without silently failing or halting the entire workflow. Partial results generated from valid inputs should remain valid, and users should be notified of any issues for follow-up.
Flexibility: This characteristic captures the ease with which the pipeline can be deployed, configured, and operated across diverse environments without requiring extensive customisation or manual intervention.	
Adaptability	The pipeline should be installable and executable across different environments (*e.g.,* from an HPC cluster to a public cloud server) with minimal changes to configuration or dependencies. Ideally, installation steps remain consistent across platforms.
Scalability	The pipeline should maintain functional correctness and stable performance when processing large-scale inputs (*e.g.,* millions of files) as well as small ones, with output quality unaffected by input size.

Understanding what existing gaps in NDT platform are expected to be bridged by ML models is key to address **RQ2**. When assisting engineers to plan networks and troubleshoot faults, NDT platforms are capable of providing what-if scenarios through simulations and poll network devices at regular intervals for keep-alive messages, the device performance metrics, and the other analytics involving the data they are transmitting and receiving. At a glance, this is similar to our pipeline prototype, which includes a simulation module, with

a downstream ML pipeline architecture featuring a feedback training loop to iteratively train with incoming dataset on top of existing dataset. Some crucial distinctions between the two frameworks is that the NDT polls query for real-time, just-in-time status, whereas ML pipelines learn from historical and simulated observations, producing predictive models than real-time snapshots.

The differences of objectives between the two frameworks are:

- **Unknown unknowns:** ML models may fail when encountering operational conditions absent from the training data. In our own generated datasets, omissions such as missing environmental features (*e.g.,* bodies of water) can lead to biased predictions or faulty assumptions, leading to AI hallucinations when ML outputs diverge from real-world conditions.

- **Risk of automation:** Standards proposed by bodies such as ETSI envision 'zero-touch' architectures where ML-driven decisions could act on NDT observations. However, faulty models risk propagating erroneous automated actions, potentially triggering degraded network performance or even service outages.

Based on the lessons learned in this project, we propose three criteria to guide ML-NDT integration:

1. **ML-NDT Feedback Loop:** NDT-generated status polls should continuously update ML models, creating a closed-loop system where real-time data supplements historical training sets, reducing the drift between predictions and operational reality.

2. **Bounded Autonomy:** ML models should augment, not replace, NDT observations. For instance, when NDT polling fails for some network elements, ML models could interpolate missing data. However, decision-making authority should remain constrained until robust safeguards against erroneous predictions are established.

3. **Error Detection and Trustworthiness Assessment:** Determining when the combined ML-NDT outputs can be trusted remains an open research problem. While our work has incorporated basic quality checks within the ML pipeline, defining similar mechanisms for the combined framework (*e.g.,* anomaly detection, confidence scoring, rollback mechanisms) requires further study and standardisation efforts.

These criteria provide a starting point for safe hybridisation of ML models with NDT platforms. Future research must focus on formalising trustworthiness metrics and automated fail-safes before fully autonomous, zero-touch network management can be realised.

While RQ2 examined the criteria for a combined ML and NDT framework to be trusted, we shall investigate why such trust should remain conditional in our attempt to answer **RQ3**. Accurate modelling of wireless environments hinges on how well real-world factors are captured in datasets feeding both simulation platforms and ML models. However, our own analyses of our prototype as outlined in Chapters 5 and 6 reveal several critical gaps:

- **Incomplete representation of environmental geometry:** Publicly available datasets, such as 3D building models with low LoD, often lack finer-scale environmental features (*e.g.,* foliage, bodies of water, street furniture) that strongly affect signal propagation. Our own pipeline exposed this limitation: missing environmental elements led to systematic prediction biases when training ML models on simulated coverage maps, even before downstream analysis began.

- **Temporal drift between datasets and deployment reality:** The constant evolution of wireless environments should be acknowledged: new buildings, vegetation growth, or even temporary events (*e.g.,* construction sites) can rapidly invalidate previously accurate datasets. ML models trained on stale data may thus fail to predict real-world behaviours in real-time, producing outputs that look plausible but diverge from current network conditions.

- **Sparse or biased simulation scenarios:** ML models for zero-touch NDT rely heavily on the diversity and representativeness of their training datasets. However, simulation pipelines often generate scenarios with uneven coverage: urban environments tend to dominate due to their structural complexity and higher transmitter density, while rural or topographically diverse regions remain underrepresented. This imbalance risks creating models that perform well in dense metropolitan settings but fail to generalize to areas with sparse infrastructure or unique terrain features. Our prototype, for instance, highlighted this issue by showing how simulations of the flat Dutch landscape yielded propagation characteristics unlikely to reflect those of mountainous or heterogeneous regions. Future dataset generation efforts must therefore account for geographic and structural diversity to avoid biased model behavior in real-world deployments.

Together, these insights not only address our research questions but also chart a roadmap for evolving today's experimental prototypes into tomorrow's trustworthy, autonomous, and adaptive digital twin ecosystems. Building on our findings will help ensuring the future networks are built on both technical rigour and operational resilience.

9

Related Work

This chapter reviews prior work relevant to our research on 5G propagation and simulation. Signal propagation modelling has become a key research area, attracting significant interest from both academia and industry. We highlight selected projects, focusing on their methodologies, design choices, and outcomes, and discuss how these compare to our own approach.

9.1 Simulation Approaches in Ray Tracing Prediction

A substantial body of work has examined the use of ray-tracing for predicting 5G propagation. One notable example is the BostonTwin project, which served as a methodological inspiration for our prototype. BostonTwin was developed to characterise the behaviour of 6G signals and relied on Nvidia Sionna as its core simulation engine (11). Because of the overlap in goals and tools, BostonTwin was a major inspiration source to our methodologies, where we encoutered several challenges which mirrored those we faced during development. In addition, the findings of the BostonTwim team provide a useful benchmark for comparison. For instance, the BostonTwin team collaborated with the Boston Planning and Development Agency (BPDA) to obtain geospatial input datasets and a database of licensed antenna sites. BPDA maps employed EPSG:4269 (North American Datum 1983) and EPSG:5703 (North American Vertical Datum 1988), which were subsequently normalised to EPSG:4326 in the released datasets. While appropriate for the areas within the contiguous United States and Canada, the varying availability of coordinate systems highlight the complexity and potential for error when integrating data from multiple government sources.

There are also methodological differences between BostonTwin and our approach. First, BostonTwin re-centres all scenes to the origin point (0,0,0) in its simulations and published datasets. We explicitly avoided this design choice. While re-centering simplifies single-scene visualisation, it shifts the burden of coordinate management and multi-tile merging to the user. In contrast, our pipeline retains original coordinates, enabling seamless merging of adjacent tiles. We believe this to be particularly important since signal transmission is inherently continuous and not bounded by arbitrary scene limits (see chapter 4). Second, BostonTwin outputs its datasets in `.geojson` format, whereas we adopt `.npy`. The likely motivation behind `.geojson` is to preserve geospatial metadata and 3D structure information. Our focus, on the other hand, is on compact numerical representations of propagation outcomes, leaving the inference of environmental context to downstream ML models. The trade-off here is clear: BostonTwin datasets are richer but heavier, while ours are lightweight and optimised for large-scale training at the cost of requiring additional inference effort.

Finally, BostonTwin provides publicly available datasets for the Boston metropolitan area, whereas our contribution lies in releasing a configurable pipeline prototype. This enables future researchers to customise simulations to their own use cases, including choice of geographical input, antenna database, and transmission parameters.

Another representative project is PMNet, which proposes a trained neural network model for predicting 5G and 6G path loss (12). PMNet was trained on datasets generated from scenes at the University of Southern California (USC), University of California, Los Angeles (UCLA), and selected areas in Boston. Alongside the trained model, the authors released their datasets and ML training pipeline. Their evaluation compared PMNet against more established signal propagation models, namely 3GPP and RadioUNet, using Wireless InSite simulations as the ground truth of the comparisons. PMNet influenced our design choice to include automated satellite image generation for each scene, and to extend our own work by validating our generated datasets against 3GPP baselines and alternative models, which would be a central objective of the next phase of our research.

The PMNet pipeline differs from ours in several respects. Most notably, it employs the proprietary Wireless InSite simulator, whereas our pipeline is built on the open-source Nvidia Sionna framework. Furthermore, PMNet focuses exclusively on predicting path loss between transmitters and receivers. In contrast, our pipeline additionally computes Received Signal Strength (RSS) and Signal-to-Interference-plus-Noise Ratio (SINR), and provides direct visualisation tools for quick analysis of user-defined geographical inputs.

Finally, although PMNet publishes its ML training pipeline, it is oriented towards extending the authors' pre-trained model. By comparison, our emphasis is on generating flexible propagation datasets and visualisation maps, facilitating use cases ranging from ML model training to exploratory scenario testing.

This comparison illustrates the diversity of approaches in recent literature: from dataset publication (BostonTwin), to trained-model release (PMNet), to pipeline-oriented contributions such as our own. Our work aims to complement this ecosystem by providing a lightweight, extensible pipeline that bridges between raw simulation, dataset generation, and ML-ready outputs.

9.2 Simulation Approaches in System-Level Telco Simulations

Characterising radio propagation through RSS and SINR is only one part of simulation for a gNodeB site-planning and building. Other critical aspects belong to the Physical (PHY) and System (SYS) layers, such as OFDM modulation and demodulation, MIMO precoding and decoding, and the evaluation of Bit Error Rate (BER). These parameters ultimately shape the Quality of Service (QoS) perceived by end users and are sensitive to both network conditions (*e.g.,* user density) and engineering choices in infrastructure planning. Our ray tracing pipeline does not cover these PHY/SYS-level effects, as its focus is on the spatial and material-dependent behaviour of radio signals.

The 5G-LENA project began as an experimental module for the ns-3 software, an open-sourced telecommunications simulation tool, to extend the software's use for simulating 5G RAN (13). Similar to our prototype's architecture, 5G-LENA is a pipeline-driven software modeling packet behaviour at the PHY and SYS layers. It enables configurable scenarios at the PHY and SYS layers, such as variable numerologies, bandwidth allocations, and scheduling policies, thereby capturing packet-level interactions within the RAN rather than physical-environment propagation. In this sense, 5G-LENA complements our prototype: whereas our work centres on propagation modelling, 5G-LENA abstracts the radio channel and concentrates on the pipeline-like flow of packets through the radio stack. Both types of simulation are essential, but they target different aspects of telco research.

By contrast, Nvidia Sionna framework can bridge both domains: it supports PHY and SYS layer simulations while also integrating with ray tracing. These features are currently outside the scope of our work, where we use it primarily for signal propagation modeling. A potential future direction would be to extend our pipeline to incorporate PHY/SYS

aspects, thereby offering a more comprehensive simulator that spans both physical channel characteristics and system-level performance.

9.3 Simulation Approaches in Urban Scenarios Simulation

Our proposed multi-stage pipeline architecture for dataset generation resonates with approaches developed in other domains to model and simulate the dynamic behaviour of complex urban environments. A notable example is the Multi-Agent Transport Simulation (MATSim), a Java-based platform originally designed to simulate and predict car traffic flows in an urban setting (14). MATSim is structured around five core components:

- **Initial Demand:** Defines the input to the simulation, such as the population of a city or the number of vehicles in circulation. Each entity in the dataset is represented as an *agent* (*e.g.,* a person, vehicle, or passenger)
- **mobsim:** The mobility simulation engine, responsible for executing the activities of each agent. This includes fine-grained behaviours such as vehicle responses to traffic conditions or time-dependent travel patterns. Each agent follows a *plan*, *i.e.,* a synthesised sequence of activities within a defined set of parameters.
- **scoring:** Evaluates batches of agent plans using a game-theoretic framework. Plans are ranked, typically following a normal distribution, with underperforming plans flagged for removal.
- **replanning:** Functions as the feedback mechanism of the pipeline. Plans marked for removal are modified by incorporating alternative activities referred to as "best-response" adjustments. If these modified plans outperform the originals, they original plans are retained for subsequent iterations. Otherwise, they are discarded.
- **analyses**: Aggregates the accepted plans after the final iteration and provides performance metrics according to user-defined objectives.

The modular design of MATSim enables research groups to extend the framework for diverse use cases, including the optimisation of road toll and tax pricing (15) and calculating commuting cyclists' safety risks (16). MATSim's adaptability was the key reason that it was incorporated as the core of the SIMBA MOBi framework, developed by Swiss Federal Railways (SBB) in 2017 to forecast passenger demand by simulating door-to-door travel patterns (17). MATSim illustrates how a modular, feedback-driven simulation system can

be progressively extended from a tool for academic research into a robust platform for nationwide infrastructure planning.

This trajectory mirrors our aspirations for the proposed prototype: although developed within a different application domain, it shares the principles of staged processing, feedback-based refinement, and reliance on realistic datasets to evaluate large-scale, complex systems. Drawing inspiration from this architecture, our prototype could be extended to support future-oriented simulations. This would enable telecom radio engineers to incorporate potential gNodeB deployment sites and assess how such additions may influence existing signal propagation patterns.

9.4 Simulation Approaches in Distributed Physics

Distributed computing has long been employed to enable large-scale simulations of physics-constrained systems, where spatial and temporal dependencies demand high levels of parallelism. An illustrative example is the eSalsa-POP (Parallel Ocean Program) project, a Fortran-based framework designed to predict oceanic Eddy currents on multi-year timeframes (18). As a simulation workload, eSalsa-POP exhibits two dimensions of parallelisation. First, the global map is partitioned into grid cells, excluding cells whose more than 50% of the area is covering landmass. Each ocean cell's state is iteratively updated based on neighbouring 'halo cells', reflecting the continuous nature of physical processes such as temperature, salinity, and wind-driven circulation. Convergence is reached once successive iterations plateau, indicating that local interactions have been sufficiently stabilised.

Secondly, a single execution generates multiple sub-simulations with slight variances to initially-assigned parameters. This design reflects the sensitivity of chaotic systems, where small parameter variations may produce divergent outcomes. In HPC terms, this corresponds to ensemble simulations, enabling exploration of a parameter space and the identification of possible worst-case scenarios.

Although our pipeline prototype originates from a different application domain, there are parallels that can be drawn. Although the map-tiling strategy was primarily necessitated by the division of 3DBAG input datasets, this nonetheless mirrors the spatial partitioning of eSalsa-POP. A potential avenue for future improvement lies in adopting a 'halo-cell' strategy for radio signal propagation, allowing neighbouring tiles to influence simulation outcomes at their borders. In the present iteration, a halo-like approximation has been implemented by including gNodeB sites located within 200 m beyond a tile's boundaries. However, this approach may still be insufficient, as it neglects the effects of signals being

reflected or absorbed by surfaces beyond the simulated region. Similarly, the ensemble approach of sub-simulations could be adapted to systematically test parameter variations in transmitter placement or power levels, thereby offering insights into system robustness and Quality of Service (QoS) degradation risks.

10

Conclusion

This work presented a prototype pipeline for simulating and characterising 5G signal propagation behaviour, intended to support both researchers and telecommunications professionals. Within the scope of our study, we generated datasets by simulating signals transmitted by all Base Transceiver Stations (BTS) located in the Netherlands. These datasets included key parameters such as Received Signal Strength (RSS), Signal-to-Interference-plus-Noise Ratio (SINR), and Path Loss.

The pipeline was conceived as an input subsystem, producing training datasets for downstream Machine Learning models. This contributes to the broader objective of augmenting Network Digital Twin (NDT) platforms with Artificial Intelligence. To this end, we examined the feasibility and trustworthiness of ML-trained data as a component of hypothetical automated decision-making within NDT environments, and identified the future improvements necessary to move towards a practical integration of AI and NDT.

At its current iteration, the prototype is not without limitations, especially in terms of performance benchmarking, the representativeness of generated data, and the absence of indoor scenarios. Nonetheless, it demonstrates the viability of large-scale simulation pipelines as a foundation for AI-driven experimentation. The work thus establishes a methodological bridge between radio-propagation simulation, dataset generation, and the emerging paradigm of intelligent NDTs.

Looking ahead, scaling the approach beyond the Netherlands, incorporating richer physical environments, and aligning outputs with industry standards will be critical steps. Ultimately, the central lesson of this thesis is that NDT platforms enriched by AI are not a distant aspiration, but a feasible trajectory, provided that the research community continues to invest in reproducible pipelines, trustworthy datasets, and transparent evaluation.

References

[1] 3GPP. **NR; User Equipment (UE) radio transmission and reception; Part 1: Range 1 Standalone**. Technical report, 3GPP, 4 2025. 5

[2] Christopher Cox. *An Introduction to 5G: The New Radio, 5G Network and Beyond.* John Wiley & Sons, Inc., 2021. 6

[3] Federal Communications Commission. **Final Catalog of Eligible Expenses and Estimated Costs**. Technical Report 3060-1270, Federal Communications Commission, September 2021. Accessed online. 9

[4] Jakob Hoydis, Sebastian Cammerer, Fayçal Ait Aoudia, Avinash Vem, Nikolaus Binder, Guillermo Marcus, and Alexander Keller. **Sionna: An Open-Source Library for Next-Generation Physical Layer Research**. 3 2022. 11

[5] Henri Bal, Dick Epema, Cees de Laat, Rob van Nieuwpoort, John Romein, Frank Seinstra, Cees Snoek, and Harry Wijshoff. **A Medium-Scale Distributed System for Computer Science Research: Infrastructure for the Long Term** . *Computer*, **49**(05):54–63, May 2016. 12

[6] Ravi Peters, Balázs Dukai, Stelios Vitalis, Jordi van Liempt, and Jantien Stoter. **Automated 3D Reconstruction of LoD2 and LoD1 Models for All 10 Million Buildings of the Netherlands**. *Photogrammetric Engineering & Remote Sensing*, **88**:165–170, 3 2022. 13

[7] Wenzel Jakob, Sébastien Speierer, Nicolas Roussel, Merlin Nimier-David, Delio Vicini, Tizian Zeltner, Baptiste Nicolet, Miguel Crespo, Vincent Leroy, and Ziyi Zhang. **Mitsuba 3 renderer**, 2022. https://mitsuba-renderer.org. 13

[8] Wenzel Jakob, Sébastien Speierer, Nicolas Roussel, and Delio Vicini. **DR.JIT: a just-in-time compiler for differentiable rendering**. *ACM Transactions on Graphics*, **41**:1–19, 7 2022. 13

[9] Filip Biljecki and Ken Arroyo Ohori. **Automatic Semantic-preserving Conversion Between OBJ and CityGML.** In *UDMV*, pages 25–30, 2015. 43

[10] Cisco. **Cisco Private 5G-Simple, Intuitive, and Trusted**. Technical report, 2024. 44

[11] Paolo Testolina, Michele Polese, Pedram Johari, and Tommaso Melodia. **Boston Twin: the Boston Digital Twin for Ray-Tracing in 6G Networks**. In *Proceedings of the 15th ACM Multimedia Systems Conference*, MMSys '24, page 441–447, New York, NY, USA, 2024. Association for Computing Machinery. 53

[12] Ju-Hyung Lee and Andreas F. Molisch. **A Scalable and Generalizable Pathloss Map Prediction**, 2023. 54

[13] Natale Patriciello, Sandra Lagen, Biljana Bojovic, and Lorenza Giupponi. **An E2E Simulator for 5G NR Networks**, 2019. 55

[14] Kay W. Axhausen. *The Multi-Agent Transport Simulation MATSim.* Ubiquity Press, 8 2016. 56

[15] Artem Chakirov and Alexander Erath. **Overcoming challenges in road pricing design with an agent-based transport simulation**. *Arbeitsberichte Verkehrs-und Raumplanung*, **766**, 2012. 56

[16] Mohsen Nazemi. **Cyclists' perceived safety and its impact on bicycle mode choice**. In *2nd Future Cities Laboratory Conference: Transactions. Exchanging knowledge between Switzerland and Asia.* Eidgenössische Technische Hochschule Zürich, IVT, Institute for Transport Planning and Systems, 2017. 56

[17] Joschka Bischoff. **MATSim at SBB: Using and contributing to the open-source transport simulation for advanced passenger demand modeling.** Presentation at FOSDEM'24, 2 2024. Accessed: 2025-08-17. 56

[18] B. van Werkhoven, J. Maassen, M. Kliphuis, H. A. Dijkstra, S. E. Brunnabend, M. van Meersbergen, F. J. Seinstra, and H. E. Bal. **A distributed computing approach to improve the performance of the Parallel Ocean Program (v2.1)**. *Geoscientific Model Development*, **7**(1):267–281, 2014. 57

Appendix

The pipeline prototype developed in this project is currently given a temporary name '*5G-RT-SIM*', being a simple abbreviation of '5G Ray Tracing Simulator'. We are still brainstorming for new names fitting of a pipeline that can be promoted to the larger scientific community and telecommunications industry.

As of the submission of this Thesis report, it is available as a private repository under the author's GitHub account `https://github.com/chrissembiring`. The author may be contacted via email for access to clone or pull the repository. We expect to fully release the pipeline prototype and the generated dataset to the general public at a later date (after this Thesis' defence date) for further quality checks and more rigorous testing to be performed by the staff of Institute of Informatics (IvI), University of Amsterdam.

Reflection

This Thesis is the culmination of the author's studies pursuing a Master's degree in Computer Science jointly with Vrije Universiteit Amsterdam and University of Amsterdam. The project described in this Thesis document was influenced by the author's choices of courses taken, applying the knowledge gained in pursuit of a new research path. Those courses and the lessons learned in shaping this Thesis were:

- **Distributed Systems** (*course taken in Period 2, AY2023-24, coordinated by Prof. dr. Alexandru Iosup*): The course which taught the author of the complexity in dealing with workloads across heterogeneous computing infrastructures.
- **Programming Large-Scale Parallel Systems** (*course taken in Period 1, AY2024-25, coordinated by dr. Francesc Verdugo Rojano*): This course gave the author the first exposure towards parallel computing using OpenMPI and Julia, which was a stack briefly considered as an alternative technique for performing job-offloading across different DAS-6 worker nodes.
- **Digital Architecture** (*course taken in Period 2, AY2024-25, coordinated by dr. Remco de Boer*): The course which taught the author of the value of analysing software qualities against known standards such as ISO 25010 and valuable software design pattern habits such as Architecture Tradeoff Analysis Method (ATAM), developed by Software Engineering Institute in Carnegie Mellon University (SEI CMU)

The author's journey in developing the research topic to the completion of prototype and the Thesis report could not be said as straightforward. The topic of 5G signal characterisation caught the interest of the author simply due to the author's previous academic and professional background in telecommunication engineering. However, as the research unfolded, the author learnt much more about geoinformation and pipeline development habits rather than gaining new insights around 5G technology. This showed how research

could indeed be volatile, showing one from an unknown path to the next; nonetheless, the journey was fruitful all the same.