

University of Potsdam  
Faculty of Science  
M.Sc. Computational Science

## Interdisciplinary Project

Hanna Kretz  
Matriculation number: 824063

Solar System Detection with GeoAI  
April 2025 – July 2025 at virtualcitysystems GmbH

University	Prof. Dr. Sukanya Bhowmik Philipp Ungrund	sukanya.bhowmik@uni-potsdam.de philipp.ungrund@uni-potsdam.de
virtualcitysystems GmbH	Dr. Lutz Ross Dr. Stefan Trometer	lross@vc.systems strometer@vc.systems

Berlin, August 14th, 2025

## **Abstract**

In this interdisciplinary project, installed solar systems in Berlin were detected with the GeoAI [16] and a functioning analysis workflow that can be used to detect installed solar systems in other cities was built and tested. Currently, there is no publicly available dataset in Germany including all installed solar systems with their locations and sizes, so using GeoAI to detect them could help to close this gap. The resulting data can be integrated into urban digital models. To ensure valid results, the GeoAI model for solar system detection was retrained using self-created training data. Furthermore, the accuracy of the detection of solar systems by GeoAI was investigated during this project. The lack of reference data made it difficult to determine accuracy. However, all signs indicate that the individual results of the GeoAI's solar system detection are inaccurate and insufficient, despite its own model training. Suggestions for improvement to increase the accuracy are provided. Nevertheless, a time series analysis of the change in installed solar systems with bias correction could provide valid results over the years.

# Contents

<b>Abstract</b>	<b>I</b>
<b>Contents</b>	<b>II</b>
<b>List of Abbreviations</b>	<b>IV</b>
<b>1 Introduction</b>	<b>1</b>
1.1 virtualcitysystems GmbH . . . . .	1
1.2 Motivation and Objective . . . . .	1
1.3 Used Tools and Workflows . . . . .	2
<b>2 Theoretical Background</b>	<b>3</b>
2.1 Terminology . . . . .	3
2.1.1 Vector and Raster Data . . . . .	3
2.1.2 Artificial Intelligence . . . . .	3
2.1.3 Geospatial Artificial Intelligence (GeoAI) . . . . .	3
2.1.4 Solar System . . . . .	4
2.1.5 Intersection over Union (IoU) . . . . .	4
2.2 Previous Study on Solar System Detection with GeoAI . . . . .	4
2.3 GeoAI by Qiusheng Wu . . . . .	5
2.3.1 Solar System Detection . . . . .	5
2.4 Data Basis . . . . .	5
<b>3 Model and Data Preparation</b>	<b>6</b>
3.1 GeoAI-Training . . . . .	6
3.1.1 Training Data . . . . .	6
3.1.2 Training Result . . . . .	6
3.1.3 Model Comparison . . . . .	7
3.2 Data Preprocessing . . . . .	9
<b>4 Solar System Detection with GeoAI</b>	<b>10</b>
4.1 Runtime and Storage Space . . . . .	10
<b>5 Results and Analysis</b>	<b>11</b>
5.1 Postprocessing . . . . .	11
5.1.1 Merge Overlapping Polygons . . . . .	12
5.1.2 Clipping by Building Footprints . . . . .	13
5.1.3 Filtering by Area . . . . .	13
5.2 Analysis . . . . .	14
5.2.1 Common Objects Misidentified as Solar Systems . . . . .	14
5.2.2 Statistics . . . . .	15
5.2.3 IoU from Part of the Area . . . . .	15
5.2.4 Comparison with Market Master Data Register (MaStR) . . . . .	16

5.3 Result-Integration in VC Map . . . . .	19
<b>6 Discussion and Outlook</b>	<b>20</b>
6.1 Discussion . . . . .	20
6.1.1 Error Analysis . . . . .	20
6.2 Outlook . . . . .	21
6.3 Conclusion . . . . .	22
<b>Appendix</b>	<b>V</b>
<b>A Additional Figures</b>	<b>VI</b>
A.1 Model Comparison: Exemplary Buildings with & without Solar Systems . . . . .	X
A.2 Examples of Correct vs. False Identification . . . . .	XIII
<b>B Additional Tables</b>	<b>XIV</b>
<b>C Source Codes and FME workflows</b>	<b>XVI</b>
<b>D AI Prompts</b>	<b>XXIV</b>
<b>Directories</b>	<b>XXVI</b>
<b>Bibliography</b>	<b>XXVII</b>
<b>List of Figures</b>	<b>XXVIII</b>
<b>List of Tables</b>	<b>XXIX</b>
<b>List of Source Codes and Workflows</b>	<b>XXX</b>
<b>Acknowledgement</b>	<b>XXXI</b>
<b>Statement of Originality</b>	<b>XXXII</b>

# List of Abbreviations

<b>AI</b>	Artificial Intelligence
<b>CNN</b>	Convolutional Neural Network
<b>COG</b>	Cloud-Optimized GeoTIFF
<b>CRS</b>	Coordinate Reference System
<b>2D</b>	2-dimensional
<b>3D</b>	3-dimensional
<b>DL</b>	Deep Learning
<b>FME</b>	Feature Manipulation Engine
<b>GeoAI</b>	Geospatial Artificial Intelligence
<b>GIS</b>	Geographic Information Systems
<b>IoU</b>	Intersection over Union
<b>LiDAR</b>	Light Detection and Ranging
<b>LLM</b>	Large Language Models
<b>LoD</b>	Level of Detail
<b>ML</b>	Machine Learning
<b>MaStR</b>	Market Master Data Register
<b>PV</b>	Photovoltaics
<b>QGIS</b>	Quantum Geographic Information System
<b>RAM</b>	Random-Access Memory
<b>SAM</b>	Segment Anything Model
<b>TD</b>	Training Dataset
<b>TO</b>	True Orthophoto
<b>VCS</b>	virtualcitysystems GmbH

# Chapter 1

## Introduction

This interdisciplinary project is part of the Master's programme in Computational Science at the University of Potsdam and aims to expand knowledge of a natural science in relation to computer science. In this case, the interdisciplinary discipline is geospatial science. The project intends to enable analyzing and working on interdisciplinary and complex problems, as well as developing new processes and techniques.

### 1.1 virtualcitysystems GmbH

The company virtualcitysystems GmbH (VCS), which is based in Berlin, develops and sells software for geo data systems of urban digital models and shadows [14]. There are currently 29 people working at VCS in Berlin and Munich. This interdisciplinary project was part of the employment at VCS as a working student.

In the following, a selection of the products from VCS that are important for this project is listed:

- **VC Publisher** - for data preparation and publishing map
- **App Configurator** - integrated in VC Publisher; for data integration into interactive map
- **VC Map** - interactive map and urban digital model that can be published
- **VC Solar** - integrated in VC Map; calculates and visualizes solar potential on surfaces

### 1.2 Motivation and Objective

The initial idea was to investigate one interesting research question in the context of urban digital twins and urban sustainability. For example, individual tree detection, green roof analysis, and heat island analysis were found to be interesting. As a data source, the Copernicus Browser or Google Earth Engine could be used, since there are datasets with high temporal resolution but low spatial resolution.

Then the GeoAI, which will be introduced in more detail later, was discovered. There are multiple example scripts given, including a solar panel detection with a pretrained model, which is claimed to be able to identify installed solar systems in aerial images. Since there are no datasets including all installed solar systems with their locations and sizes in a German city publicly available, using GeoAI to detect them could help to close this gap. On the basis of the resulting dataset, a time series analysis could be conducted and the development of the installed solar systems over the years could be observed. As data sources, the previously mentioned ones were no longer an option, as high spatial resolution was required.

The project includes detecting installed solar systems in Berlin using GeoAI, since high-resolution

datasets were available for this city. The aim is to build an analysis workflow that can be used to detect installed solar systems in other cities too. Furthermore, it was investigated during this project how accurate the identification of solar systems by GeoAI is. The final goal was to be able to include a layer of detected solar systems in the VC Map of any choosable city.

## 1.3 Used Tools and Workflows

Tables B.1 and B.2 show the specifications of the used hardware and software components.

The workflow shown in Table 1.1 was developed for solar system detection in a choosable city and for preparing it as a layer for the VC Map.

Table 1.1: Workflow for Solar System Detection of Any City

Step	Tool
1. Data search and data download of spatial data	individual data source
2. Data preparation	FME
3. Data processing (solar panel identification)	GeoAI, Python
4. Result postprocessing	FME, QGIS, Python
5. Data integration of the result	VC Publisher
6. Visualization of results	App Configurator, VC Map

The project workflow and the structure of this report are shown in Table 1.2; it is based on the previous workflow.

Table 1.2: Workflow for Interdisciplinary Project

Step	Tool
→ Theoretical Research	
→ Training Data creation & Training of AI-model	QGIS, GeoAI
2. Data preparation	FME
3. Data processing (solar panel identification)	GeoAI
4. Result postprocessing	FME, QGIS, Python
→ Result analysis	FME, QGIS, Python
5. Data integration of the result	VC Publisher
6. Visualization of results	App Configurator, VC Map
→ Report and presentation of project	

There were three additional ideas which could be investigated during the project if the time allows:

- Comparison of detected solar systems with entries from Market Master Data Register (MaStR)
- Time series analysis of solar systems installed in recent years
- Green roof analysis

# Chapter 2

## Theoretical Background

### 2.1 Terminology

#### 2.1.1 Vector and Raster Data

Both vector and raster data are types of spatial data, which are analyzed in the scientific field of geospatial science. Vector data is used to represent features such as roads, buildings, and other objects as points, lines, and polygons. In contrast, raster data represents continuous surfaces, such as elevation and temperature, in grid-based formats. These data types enable spatial analysis and decision-making across disciplines such as urban planning and hydrology, typically within Geographic Information Systems (GIS) [5].

A prime example of raster data is orthophotos, which are geometrically corrected aerial images that represent the Earth's surface with uniform scale, thus allowing accurate measurements. True Orthophoto (TO) are corrected for building lean and occlusions, which is especially advantageous in urban environments [5].

#### 2.1.2 Artificial Intelligence

Artificial Intelligence (AI) is the science of making machines perform tasks that typically require human intelligence, such as perception, reasoning, learning, and decision-making [11].

Subfields or components of AI are:

- **Machine Learning (ML)** - a subfield of AI focused on data-driven learning
- **Deep Learning (DL)** - a subset of ML using deep neural networks
- **Large Language Models (LLM)** - specialized DL model trained on text
- **Convolutional Neural Network (CNN)** - another type of DL model, particularly effective for image and spatial data analysis

An AI model can be overtrained. This is the case when the AI memorises training data and hardly generalises. An overtrained model is characterized by high training accuracy but stagnating or even worsened accuracy on validation when using the AI. Overtraining can be recognised by instability in early epochs [7].

#### 2.1.3 Geospatial Artificial Intelligence (GeoAI)

A Geospatial Artificial Intelligence (GeoAI) integrates AI techniques with methodologies from the field of geospatial science, with the objective of extracting meaningful information [4]. A GeoAI includes a CNN in most cases.



### 2.1.4 Solar System

Solar System is the general term for a technical system that uses solar energy. It can be both a photovoltaic system or a solar thermal system. Photovoltaics (PV) is the technology for converting sunlight directly into electricity, and solar thermal is the technology for converting solar radiation into heat. An individual module that absorbs solar energy is called a solar panel [15].

### 2.1.5 Intersection over Union (IoU)

The Intersection over Union (IoU) is a metric used to quantify the overlap between two polygons, expressed as a percentage of the total area. This metric is commonly used in DL object identification to calculate the ratio between the overlap of the detected object's bounding box and the real object's bounding box, ranging from 0 to 1 [13]. In Figure 2.1 is shown how the IoU is calculated, and some examples of overlapping polygons are provided. The higher the overlap, the higher the IoU.

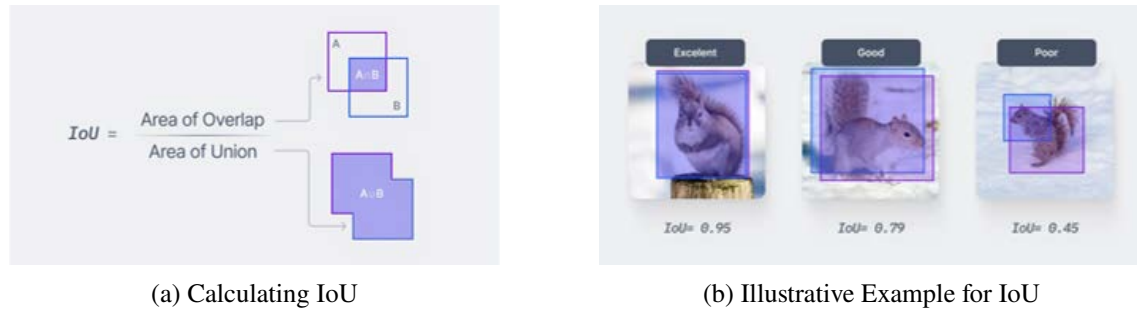


Figure 2.1: Intersection over Union [13]

## 2.2 Previous Study on Solar System Detection with GeoAI

The only paper found that has a comparable research topic as this interdisciplinary project is called "GeoAI for detection of solar photovoltaic installations in the Netherlands" from Kausika et al. [4]. In 2021, Kausika et al. identified all solar systems in the Netherlands by using a GeoAI based on the CNN model "TernausNet". The data they used (both Orthophoto and TO) had a spatial resolution of 10 cm and three bands (RGB). The workflow of the study included Data Preprocessing, Model Training, Model Implementation, and Postprocessing.

The training datasets that were used had a width of 102.4 m. During the training, they found that the model was better at detecting solar systems on smaller buildings rather than larger buildings. That's why they retrained with more diverse training datasets and achieved higher accuracy.

By using FME [12] for postprocessing steps like removing polygons outside of building footprints and filtering polygons with an area less than 0.6 m<sup>2</sup>, they improved the result quality significantly. Since the used dataset had a storage need of 4 TB, the computing process and storage had to be optimized to be as efficient as possible.

As they used both Orthophoto and TO they found out that "Orthos have better accuracy in detecting the shape of the panel correctly while TOs have better location accuracy". Combining both results yielded a final result with high accuracy. Objects that were misidentified often in their analysis are greenhouses (filtered out), tanks (filtered out), rooftop dormers, and wet patches on rooftops. Since the automated postprocessing steps removed some true positives, they decided to try a customized postprocessing with visual inspection and human intuition. This "refined the results", but simultaneously was the biggest source of error in the study: human intuition and human control, which lacks objectivity.

A comparative study with the PV register database of the Netherlands showed that due to "the difference in methodology for identifying solar panels" the datasets had "completely different

results".

They suggest using 3-dimensional (3D) models to improve the results during either training or postprocessing, e.g. to remove false positives of dormers due to height jumps. Other suggestions are to include the random forest classifier in postprocessing to improve the results and to use K-means clustering techniques to find similarities in training datasets to be able to train more specifically.

## 2.3 GeoAI by Qiusheng Wu

The GeoAI that was used in this project is developed by Wu [16] and is a powerful open-source Python platform that combines AI with geospatial data analysis. It enables processing, analyzing, and visualizing vector, raster, and LiDAR data using DL methods [16].

As the GeoAI has a modular structure, it is flexible and can be expanded. The provided examples can be customized, and custom plugins can be developed and integrated. Furthermore, the range of potential applications is greatly expanded by existing Python libraries.

There are AI models for automated segmentation and object identification integrated into GeoAI, e.g. Segment Anything Model (SAM) and YOLOv5. The GeoAI is still in active development, but documentation, tutorials, and examples are already provided online [16].

The example scripts include visualization of different geographic data, object detection (e.g. building footprints, solar systems, cars, ships, and parking slots) and training scripts to train own models. They can be executed exemplarily in Jupyter Lab or Google Colab.

### 2.3.1 Solar System Detection

One of these mentioned example scripts is for Solar System Detection. It is provided with sample data; the used model is trained on American data.

The adjusted script to identify solar systems in Berlin had a high error rate: There were both false positives and false negatives in the result. After this test, it was decided that for more valid results, the solar system detection model should be retrained. For this training, another adjusted example script can be used. The training is explained in more detail in Section 3.1.

## 2.4 Data Basis

The aerial imagery used as the data basis for the solar system detection was provided by VCS [14] and was captured by Aerowest GmbH [1]. The characteristics of the orthophoto of Berlin are listed in Table 2.1.

Table 2.1: Characteristics of Data Basis

Characteristic	Description
Spatial data	True Orthophoto (TO)
Observation time	2023
Observer	Aerowest GmbH [1]
Bands	4: RGBI (red, green, blue, infrared)
Data format	*.tif + *.tfw
Tile size	900 m x 900 m
Tile number	1667
Spatial resolution	9 cm

## Chapter 3

# Model and Data Preparation

### 3.1 GeoAI-Training

The provided example script to train GeoAI models, specifically for solar panel detection, was adjusted and used for this training. It was run in Anaconda’s JupyterLab on PC226; the hardware and software specifications are listed in Tables B.1 and B.2. The source code for the training is shown in Appendix C.

During the training, the status of each epoch is printed. The value of the IoU is a benchmark value that indicates how accurate the model is at that point in time. In Section 2.1.5 this metric is introduced. In the case of this training, the polygons of the training data and the polygons of the GeoAI are compared. The goal of  $\text{IoU} \geq 0.95$  was set at the beginning of the training to ensure high validity.

#### 3.1.1 Training Data

As training data, the solar systems on six tiles were manually identified in QGIS [10] and stored in a vector file. The characteristics of each Training Dataset (TD) are listed in Table 3.1. The tiles that were used as training datasets were several times larger than those in the previously described study by Kausika et al. [4].

Table 3.1: Training Datasets

Training Dataset	Location	Area	Bands	Tile Size	Spatial Resolution	Number of Solar Systems
TD1	Berlin	Lichtenrade	RGBI	900 m x 900 m	9 cm	114
TD2	Berlin	Niederschönhausen	RGBI	900 m x 900 m	9 cm	69
TD3	Berlin	Adlershof	RGBI	900 m x 900 m	9 cm	58
TD4	Berlin	Wuhlheide	RGBI	900 m x 900 m	9 cm	35
TD5	Templin	residential area	RGBI	1000 m x 1000 m	10 cm	17
TD6	Templin	industrial area	RGBI	1000 m x 1000 m	10 cm	16

#### 3.1.2 Training Result

There were 15 training runs in total. Table 3.2 shows how many epochs the model was trained with in each run, and which epoch had the highest IoU. The highest IoU of each training run is also

### 3.1. GEOAI-TRAINING

visible in Figure 3.1.

The first training had to be repeated because a backup didn't work and the data was lost. Surprisingly, the second training had a slightly better result of IoU even though the training data and settings were identical. The third training, with the same training dataset as the first two, was much longer with 300 epochs. In this training run, the 43rd epoch had the highest IoU with  $\text{IoU} = 0.9233$ . Since only the model with the highest IoU is saved, the following 257 epochs weren't used. Due to that unnecessary, it was decided that all following trainings would only be trained with 50 epochs. In addition, all following trainings were trained with other training datasets and on top of the previously trained model (see column "pretrained model"). In the next training, the IoU already got slightly worse. Trainings 5 and 6 were done with the datasets of Templin, which are both much smaller than the Berlin datasets. These two trainings got significantly higher IoUs, which could have a causal connection between the number of solar systems in the training dataset and the highest IoU. It is probably much easier for GeoAI to learn the solar systems in a small dataset.

Training 8 was accidentally canceled. After its repetition, it was decided that out of curiosity, all training datasets would be used again to test if the model would get better. Since every one of these trainings 10 - 15 had a lower highest IoU than the first runs of each dataset, it is clear that a repetition of trainings with already used training datasets does not increase the quality of the model.

Table 3.2: Training Runs

Training	Training Dataset	Pretrained model	Epochs	Runtime	Highest Epoch	Best IoU	Notes
1	TD1	-	20	~4 h	16	0.8839	Data lost
2	TD1	-	20	~4 h	19	0.8901	
3	TD1	-	300	~66 h	43	0.9233	
4	TD2	3	50	~11 h	24	0.9126	
5	TD5	4	50	~11 h	22	0.9500	
6	TD6	5	50	~11 h	30	0.9547	
7	TD3	6	50	~11 h	46	0.8708	
8	TD4	7	50	~11 h	-	-	
9	TD4	8	50	~11 h	6	0.9104	Cancelled
10	TD1	9	50	~11 h	40	0.8695	
11	TD2	10	50	~11 h	43	0.9006	
12	TD5	11	50	~11 h	46	0.9368	
13	TD6	12	50	~11 h	20	0.9132	
14	TD3	13	50	~11 h	28	0.7703	
15	TD4	14	50	~11 h	45	0.8612	

#### 3.1.3 Model Comparison

In Figure 3.1 it is visible that training run 3 has the highest IoU, when excluding the trainings with the small training datasets from Templin (Training 5, 6, 12 & 13).

The trained models can be compared in Appendix Section A.1 based on examples of buildings. It is visible that no model is error-free. The best results were achieved with the datasets used for training. There are some solar systems that were not detected (false negative) and objects that were misidentified as solar systems (false positive). When comparing the pretrained Mask R-CNN model which was trained on American data, the new trained models seem better. Nevertheless, based on this comparison, it is not possible to select a model that performs the best.

At the time of this training, only one raster and one vector dataset (with polygons of solar systems) can be used as training data. It is not possible to explicitly show the model which objects should not be identified as solar systems. Also, it is not possible to include further datasets such as

### 3.1. GEOAI-TRAINING

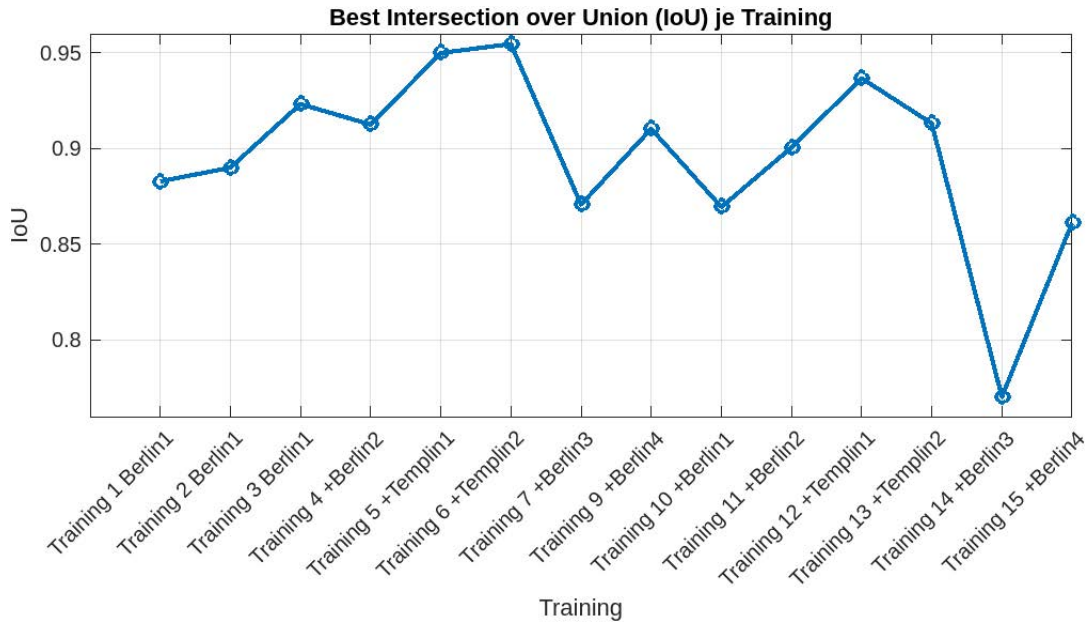


Figure 3.1: Highest IoUs per Training

building footprints. To include more datasets in the training could have the benefit of minimizing misidentifications of objects like pools, greenhouses, and cars.

In Figure 3.2 all IoUs by epochs per training run are shown (cut after 50 epochs; the complete diagram is shown in Figure A.1).

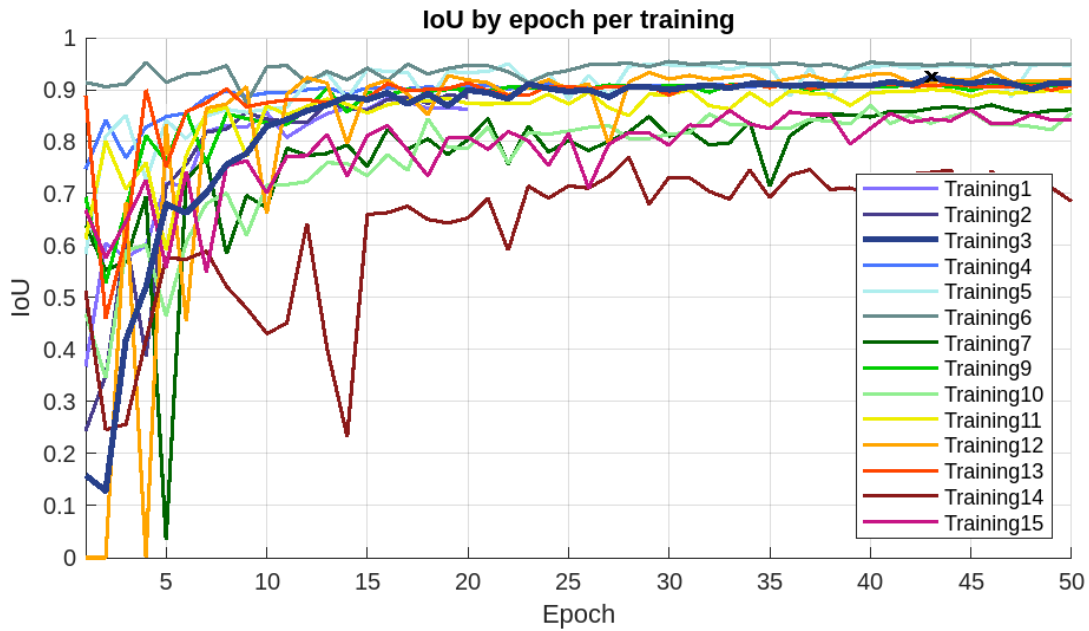


Figure 3.2: IoUs by Epochs per Training

Almost every graph of IoU has many ups and downs; especially in the early epochs, there are large fluctuations. This instability is interpreted as a sign of overtraining. The smoothest curve is the one from training three, which was the one with 300 epochs and the first training run, on which all further training runs were based. The training data for this training was located in Berlin Lichtenrade and had 114 manually identified solar systems. The highest IoU of this training is marked with an "x" (43rd epoch with IoU = 0.9233) in Figure 3.2. Based on the fact that this is the



only model which is not overtrained, it was decided that the model saved in this training would be used for the actual solar system detection in Berlin.

This training and model comparison show that additional training, training data, or training epochs do not necessarily improve the model or increase its accuracy. Once the overtraining threshold is reached, the model either stops learning or becomes worse.

## 3.2 Data Preprocessing

The described data basis mentioned in Section 2.4 had to be prepared before using it for the solar system detection of the GeoAI. The raw data was available in tiles with a size of 900x900 m each; those had to be merged. The FME workflow used is shown in Listing C.7. The RasterMosaicker in FME [12] was used to create one file with all merged tiles. Since Potsdam was included in the raw data, the data was clipped by using the contour of Berlin (shapefile - \*.shp). The result file was written as a Cloud-Optimized GeoTIFF (COG) (\*.tif), so in this preparing process, the data was also converted. A COG is useful for large datasets and is a type of the classic GeoTIFF format that is internally tiled so that it can be read partially and efficiently – without having to download the entire file.

In Figure 3.3 the result of this preprocess is visualized. The FME process had a runtime of multiple hours. Due to the size of the dataset (~390 GB) there were some difficulties when processing. An exemplary defective preprocessing result is visible in Figure A.2

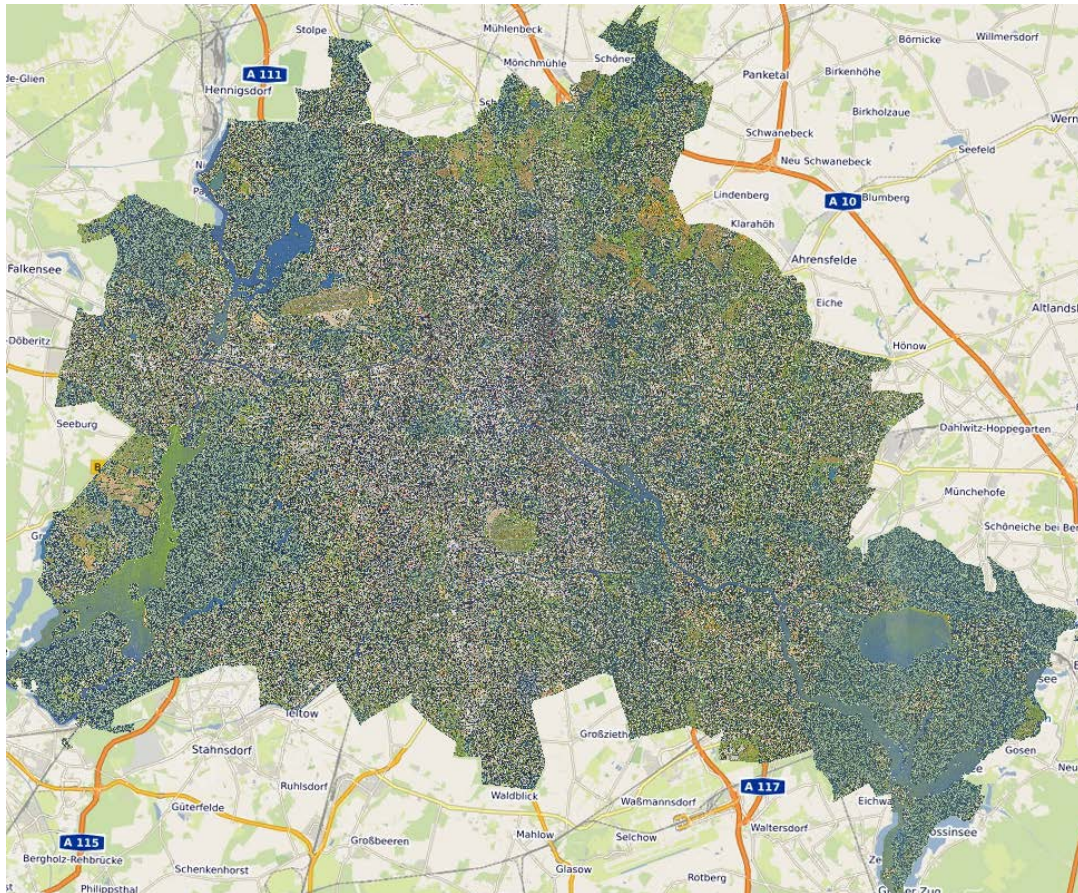


Figure 3.3: Orthophoto of Berlin 2023 (North is up; viewed in QGIS [10]; Layers from [1][9])

## Chapter 4

# Solar System Detection with GeoAI

The Python script for the solar system detection was provided as an example script of the GeoAI. The script was adapted for the described data basis of Berlin and was run in Anaconda's JupyterLab on PC226; the hardware and software specifications are listed in Tables B.1 and B.2.

Despite using the COG format, the Berlin dataset was too large to be handed over to the GeoAI function in its entirety. An error message appeared several times stating that the data could not be loaded into Random-Access Memory (RAM). Even the individual Berlin districts, which could have been processed one after the other, were too large.

Due to time pressure, a Python script was written with the help of Copilot [7] that cuts the orthophoto into tiles and creates a result file for each tile. The tiles will need to be joined back together later. Each tile has a size of 2048 x 2048 pixels, with 10 % overlap, which was suggested by Copilot. The source code for the solar system detection is shown in the Appendix C.

### 4.1 Runtime and Storage Space

The process of identifying solar systems in Berlin took a lot longer than expected in the beginning. Based on the runtime of the process on individual tiles, it was calculated that this solar system identification would take approximately 58 h.

Since the data was cut into relatively small tiles, each process of identifying solar systems took 7 s and exporting the result another 7 s. Due to this, the process was estimated to take about 116 h.

When, after a period of five days, the process was still not complete, the data was inspected: A significant number of empty tiles were located in areas outside of Berlin. FME [12] generated a bounding box around the city of Berlin when preparing the data, with the space between the city corner and the bounding box being filled with empty tiles. These were transparent when inspected in QGIS [10], which is why the empty data was not recognized. Consequently, the quantity of tiles to be processed increased from 32,000 to 62,000. The expected runtime then was calculated to 225 h.

Due to a temporarily missing network connection, the process was interrupted for 6 h. After a total runtime of 244.63 h  $\approx$  10 d the solar system detection was completed. The result files took 258 GB of storage space.



## Chapter 5

# Results and Analysis

An overview of the raw result is visible in Figure 5.1.

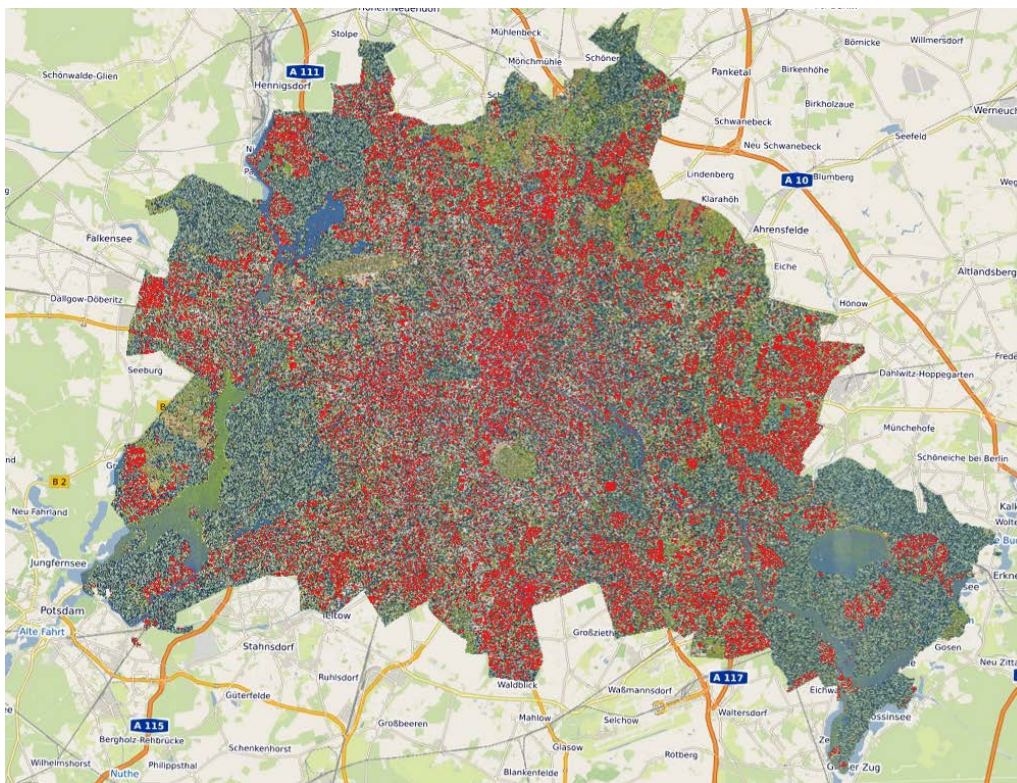


Figure 5.1: Solar Systems (red polygons) in Berlin (North is up; edited in QGIS [10]; Layers from [1][9])

### 5.1 Postprocessing

To be able to analyze the results, there were some steps of postprocessing necessary. The chosen steps are visible in Table 5.1. Since FME [12], which was mostly used for these postprocessing steps, was not able to merge all result tiles at once, it was necessary to split the tiles into seven batches, with ~10,000 tiles each. Those tiles per batch were merged by using the function `RasterMosaicker` of FME again (Listing C.8).

The data was also converted into shapefiles (\*.shp), thus transitioning from a raster to a vector data format. In order to integrate it into the VC Map, it is necessary to use the \*.geojson format.



Upon the direct saving of the data as a \*.geojson file, some issues were encountered: First, the Coordinate Reference System (CRS) was not the one that was supposed to be used (EPSG 25833). Consequently, the result could not be visualised with the orthophoto, which had the previously mentioned CRS. The reason was that the \*.geojson format does not support this projection. Another problem was that many polygons were exported incorrectly. Polygons with an area of less than 1 m<sup>2</sup> were no longer output as polygons, but as points. Consequently, these areas were therefore unusable. Furthermore, all polygons' attributes (e.g. area value) were rounded to integers, as floating-point numbers were not supported. The resolution to these issues was achieved through the utilisation of the \*.shp data format, followed by the conversion of the data into \*.geojson. It was ensured that both the data and the CRS were saved correctly (Listing C.8).

Then the seven vector files of each batch could be merged into one. The total number of polygons in this file was 100,761 polygons.

Table 5.1: Postprocessing Steps

Postprocessing Step	Result	Number of Polygons
	Raw data	
Split into 7 batches with ~10000 tiles each	↓	
	Batch 1 2 3 4 5 6 7	
Remove empty tiles, mosaicker, conversion *.geotiff → *.shp	↓	
	*.shp 1 2 3 4 5 6 7	
Merge seven files	↓	
	solar.shp	100761
Merge overlapping polygons	↓	
	solar_clean.shp	92478
Clip by building footprints	↓	
	solar_clean_clipped.shp	83368
Filtering Polygons with area < 1 m <sup>2</sup>	↓	
	solar_clean_clipped_o1.shp	72655

In the following, three different filtering steps and their results are described.

### 5.1.1 Merge Overlapping Polygons

Due to the overlap of 10 % of each tile, there were also overlapping polygons. An example is visible in Figure 5.2. These were merged, so that every identified solar system had just one polygon (FME workflow visible in Listing C.9). After this postprocessing step, there were 92,478 polygons left.



Figure 5.2: Example: Overlapping red Polygons (North is up; viewed in QGIS [10]; Layer from [1])

### 5.1.2 Clipping by Building Footprints

Since there are some misidentified objects, the idea was to clip the results by building footprints so that just the identified solar systems that lay on a building were kept (FME workflow visible in Listing C.10). This filtering resulted in 83,368 polygons. Examples of clipped polygons are visible in Figure 5.3.

As shown in Figure 5.3a some misidentified objects, such as pools, like in this case, were filtered out when clipping by footprints. In some cases, when the detected solar system was bigger than the building, the edge of this solar system was also cut out.

Figure 5.3b shows one of the few cases where a correct solar system was filtered out because the building shown in the orthophoto was not included in the building footprint dataset. The building footprints are from 2025, so the orthophoto was older and therefore both datasets were incompatible.

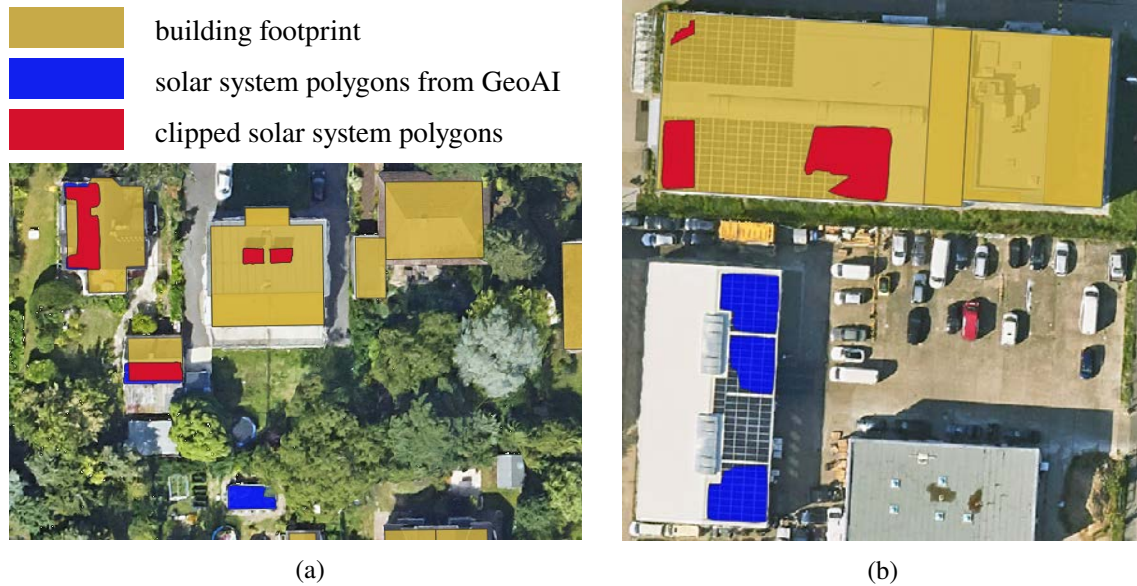


Figure 5.3: Examples: Clipping by Building Footprints (North is up; edited in QGIS [10]; Layer from [1])

At the same time as clipping with building footprints, it was checked for each polygon where the solar system is located. The attributes 'address' and 'usage type' were then transmitted to the solar system polygons. So after this process, each polygon had been assigned a corresponding address. An exemplary attribute table is visible in Figure A.3.

### 5.1.3 Filtering by Area

As a last postprocessing step, the polygons with an area of less than  $1 \text{ m}^2$  were filtered, because normally there is no solar system this small. The area of the polygons had to be calculated first (FME workflow visible in Listing C.11), then the polygons could be filtered by size (FME workflow shown in Listing C.12). After this step, there were 72,655 polygons left. For comparing all polygons with an area less or more than  $1 \text{ m}^2$  the results are shown in Figure A.4.

An example of polygons with an area of less than  $1 \text{ m}^2$  is visible in Figure A.5. In this example, it is visible that some polygons are this small (red) even though the actual solar system is bigger (green) and just wasn't detected in total. In Section 5.2.3 it will be analyzed if this filtering step is improving the result or not.

## 5.2 Analysis

Some examples of correct vs. false identification can be viewed in Appendix Section A.2. It is visible that some solar system detections were identified with high accuracy. Some others have missing parts in the middle, without a visible reason. And other solar systems weren't detected at all. There are also some cases of misidentifications like roof windows, which were identified as solar systems.

Generally, the solar systems are often installed on a roof with a slope angle, but the orthophoto is in contrast perfectly vertical. This is why many solar systems are trapezoidal rather than rectangular when viewed from this perspective. Furthermore, the area calculated from this perspective is not entirely accurate.

### 5.2.1 Common Objects Misidentified as Solar Systems

Objects that were misidentified multiple times are listed down below.

- Roof windows
- Glass canopies
- Greenhouses (partly removed when clipped by building footprints)
- Shade (partly removed when clipped by building footprints)
- Conservatories (partly removed when clipped by building footprints)
- Pools (removed when clipped by building footprints)
- Cars (removed when clipped by building footprints)

To understand these misidentifications better, a library of spectral signatures of different objects was collected; it is shown in Figure 5.4. A spectral signature is the value of the bands (in this case RGBI) for one pixel. One object has pixels with similar spectral signatures. If different objects also share similar spectral signatures, they are difficult to distinguish without context.

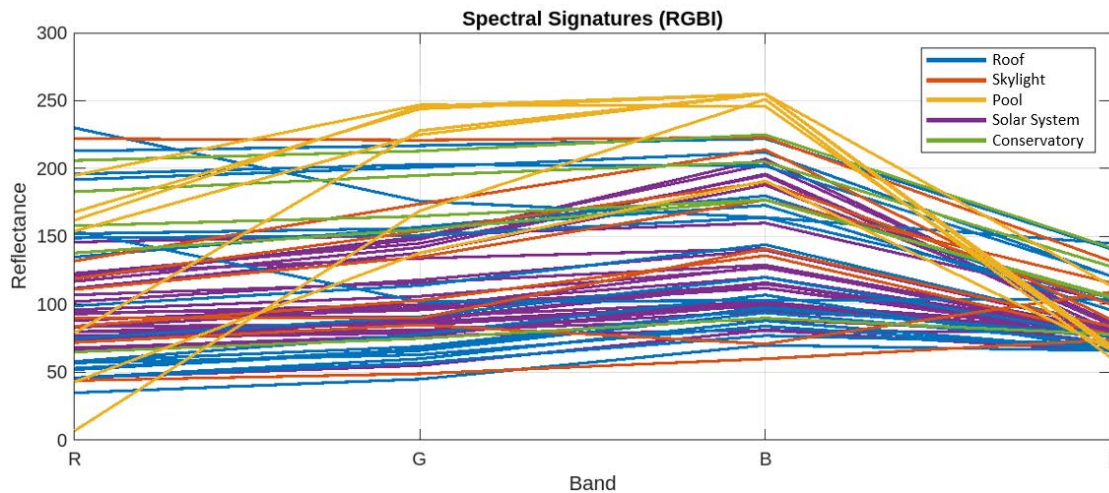


Figure 5.4: Spectral Library of Common Objects Misidentified as Solar Systems

In this case, it is visible that solar systems (purple signatures) share similarities with all other collected objects: roofs (dark or in shadow), skylights, pools, and conservatories. So this shows why it is difficult for the GeoAI to distinguish between some objects and solar systems and, therefore, misidentifies some of them.

A solution to this problem could be Hyperspectral or Multispectral data: Both contain more than

4 bands for each pixel. This way, the spectral signatures would get more complex and easier to distinguish. In the case of Berlin, this would multiply the size of the dataset and possibly lead to other difficulties.

### 5.2.2 Statistics

Table 5.2 shows some of the statistical parameters for the area per polygon in  $\text{m}^2$  for each postprocessing step, calculated by QGIS [10].

The number of polygons gets smaller with each step. The mean and the median of the area of all polygons get bigger with each postprocessing step, except when clipping with the building footprints: the mean is smaller there because many polygons get smaller when clipping. The standard deviation (stddev) is relatively high compared to the average polygon size. This shows that the size of the polygons is not normally distributed. You can also see this in the histogram in Figure A.6.

The total area gets smaller with each post-processing step. Most of the area is cut out when clipping by building footprint, and the least when filtering polygons with an area of less than  $1 \text{ m}^2$ . The smallest polygon's value didn't change in the first step of filtering. In the second step, some polygons were cut into very small polygons. The smallest polygon after filtering out the ones with an area of less than  $1 \text{ m}^2$  shows that the filtering worked, because its area is then  $1.0001 \text{ m}^2$ . The biggest polygon of all didn't change in the postprocessing; it has an area of  $1038.49 \text{ m}^2$ .

Table 5.2: Statistical Parameters of Area per Polygon  $\text{m}^2$

Dataset	Count	Mean	Median	Stddev	Sum	Min	Max
solar.shp	100,761	16.45	9.14	28.21	1,657,775.71	0.008	1038.49
solar_clean.shp	92,478	17.75	10.10	29.06	1,641,493.57	0.008	1038.49
solar_clean_ clipped.shp	83,368	16.39	8.80	28.35	1,366,769.24	$2 \times 10^{-13}$	1038.49
solar_clean_ clipped_o1.shp	72,655	18.78	10.67	29.63	1,364,177.50	1.0001	1038.49

### 5.2.3 IoU from Part of the Area

To get a value of accuracy of the result of the identified solar systems, the IoUs of the four Berlin tiles the model was trained with were calculated. There was no other correct dataset as reference available, so the manually created datasets were used. For this, the result was cut into the shape of these four tiles with the FME workflow shown in Listing C.15. A short Python script, which is shown in Appendix C, was written with the help of Copilot [7] for calculating the IoUs.

In Table 5.3 the number of solar systems per tile can be compared. The GeoAI drew more polygons than there were in the training datasets in every tile. This doesn't necessarily mean that the solar system detection is of lower quality, because there could be multiple solar systems on one roof, which in the training data might just be identified as one. However, this could also mean that there are misidentifications, such as roof windows, which were detected as solar systems.

Table 5.3: Comparison of Polygon Counts in Training Data and GeoAI Result

Training Dataset	Location	Number of Polygons in Training Data	Number of Polygons of GeoAI Identifications
TD1	Berlin Lichtenrade	114	144
TD2	Berlin Niederschönhausen	69	121
TD3	Berlin Adlershof	58	75
TD4	Berlin Wuhlheide	35	38

Figure 5.5 shows the result of the calculated IoUs per tile and postprocessing step.

Three of the tiles had an average calculated IoU > 0.69. The highest IoU was calculated on the tile that the used GeoAI model was trained with. The IoU of the tile of training dataset 3 was calculated as ~0.37. The reason for this poor result might be that the solar systems were not easy to identify, both for the GeoAI and for the human eye, which manually identified the solar systems for the training dataset. The training dataset might not have been good enough. Even if this result is ignored, the calculated IoU of the other three tiles is also not satisfactory. However, it reflects the impression one gets when looking at the examples of correct and false identifications.

Comparing the IoUs of the postprocessing steps, it shows clearly that clipping the data by building footprints has the most impact: Three of the four tiles have a higher IoU. The calculated IoU doesn't change much in the last postprocessing step but three decrease a little, so the small polygons with an area of less than 1 m<sup>2</sup> improve the quality of the result slightly.

The calculated mean of the IoUs per postprocessing step is shown in Table B.3. The average IoU is the highest after clipping the data by building footprints. Based on this, it was decided that this version of the result (solar\_clean\_clipped.shp) would be used as the final result.

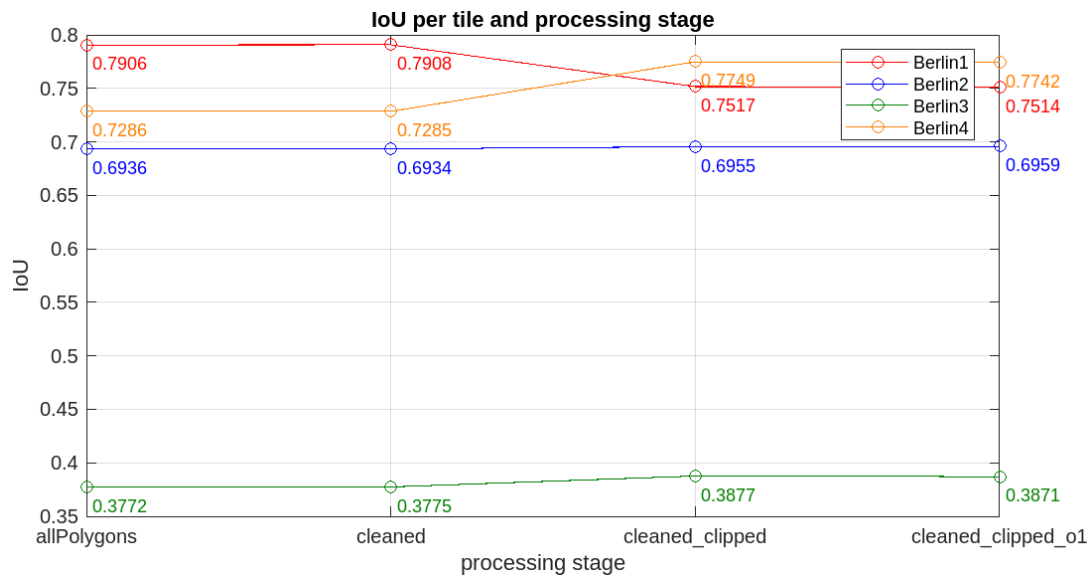


Figure 5.5: IoU per Tile and Postprocessing Step

#### 5.2.4 Comparison with Market Master Data Register (MaStR)

The MaStR is an official German register in which master data relating to the electricity and gas market is recorded. The MaStR is maintained by the Federal Network Agency [2]. There is also data available about installed solar systems because they have to be registered.

The data can be exported as a \*.csv file. There were 49,127 entries of solar systems in Berlin in total, but just 1,335 of them had public information on addresses and coordinates. The post code was public for every entry, as well as values such as “Installed output of the EEG system”. Since no comparison with all data entries was possible due to the lack of location information, two different aspects are analyzed: First, entries are compared based on location and approximate size where an address is available. Second, the number of registered solar systems per postcode area is compared.

##### Comparison of Area by Location

For the comparison of location and approximate size, first the data from the MaStR had to be prepared. The location data given in the downloaded \*.csv file was ready to use. There were no size specifications given in the data. Therefore, an estimated size was calculated from the value “Installed output of the EEG system” since it is roughly proportional to the area. Solar systems vary



## 5.2. ANALYSIS

in efficiency, meaning proportionality is not fully guaranteed.

Again, with the help of Copilot [7] a Python script (Appendix C) was written to (1.) extract coordinates and power values from \*.csv file; (2.) group entries that have the same location and sum up “Installed output of the EEG system”; (3.) calculate the area of solar systems at each location and create circles at each location of solar systems; and (4.) export \*.geojson file.

Figure 5.6 shows how the result looked like. The circles in brown show how the circles looked before grouping the entries that have the same location; in red, the result after grouping is shown.

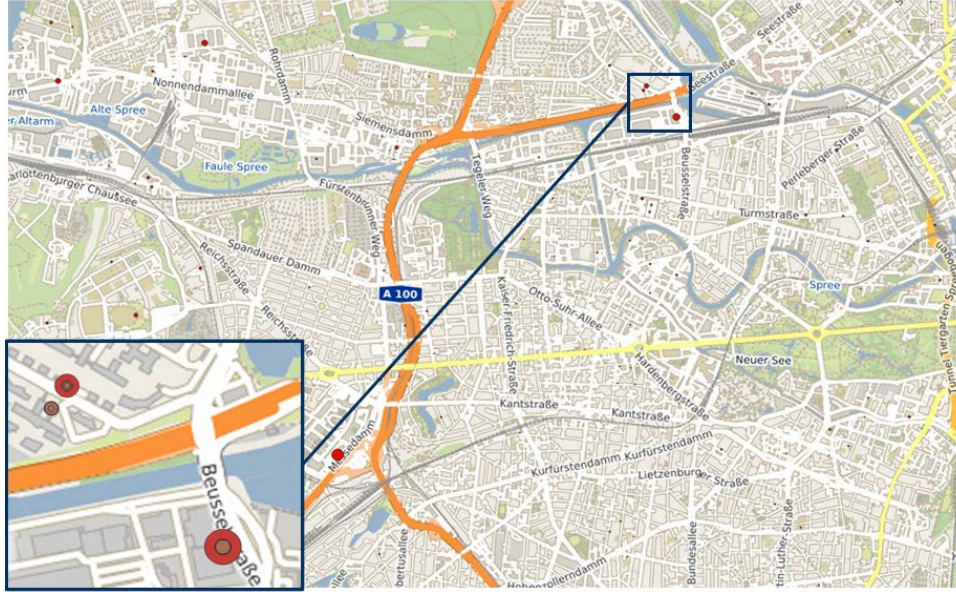


Figure 5.6: Visualized Data from MaStR (North is up; edited in QGIS [10]; Layer from [9])

To prepare the results from the GeoAI with FME [12] for this comparison, (1.) the sum of the area of detected solar systems per building was calculated; (2.) the center of the building was calculated; and (3.) the circles were created at the center of each building (FME workflow shown in Listing C.13). The result of this preparation is shown in Figure 5.7a.

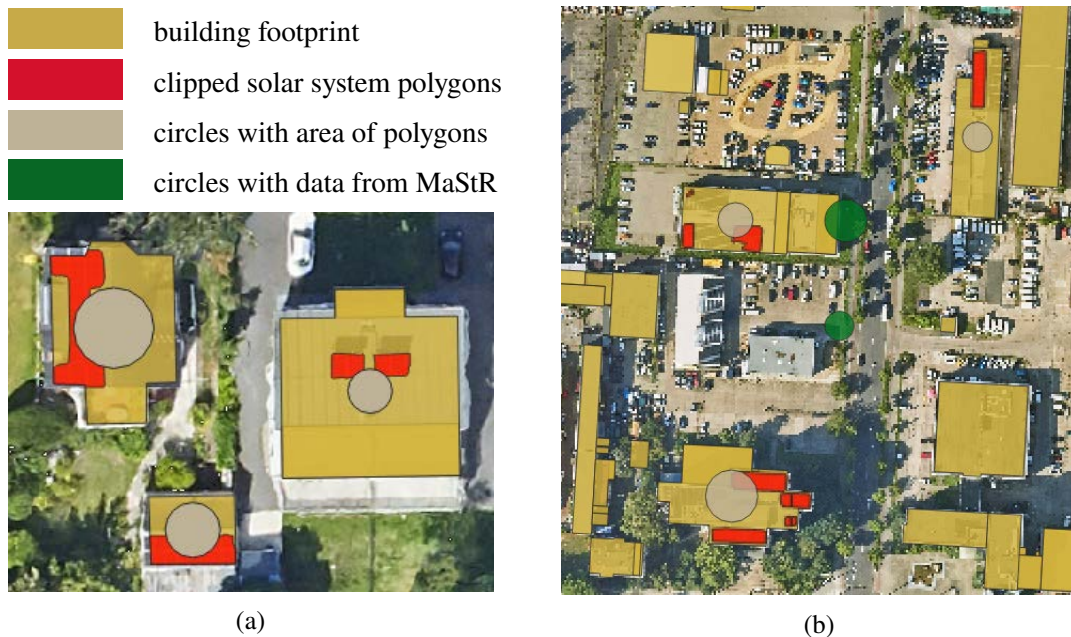


Figure 5.7: Area Comparison: Solar Systems from GeoAI vs. MaStR (North is up; edited in QGIS [10]; Layer from [1])

Some circles had to be filtered because there was a problem with calculating the size of the circle. Instead of 44,544, there were only 36,295 circles left to compare. In Figure A.7 it is visualized how the circles looked before filtering.

When the two datasets were visualised together, it was found that the circles representing the data from MaStR were not located on the buildings, but next to them. This made the comparison much more difficult, as the overlap could not be calculated. An example in Figure 5.7b shows that in this case, the circles are of similar size. However, this is not the case in many other examples. This comparison is insufficient and does not provide a basis for valid conclusions.

### Comparison of Number of Solar Systems per Postcode

For comparing the number of registered solar systems per postcode area, it was also necessary to group the solar systems that are located at the same address. To get the number of solar systems in a postcode area both from GeoAI and the MaStR, FME [12] was used again (workflow shown in Listing C.14).

The visualized results are shown in Figure 5.8. It is visible that in most areas there are more entries of solar systems in the MaStR than the GeoAI identified. In some cases, where the number of solar systems is relatively high, the absolute difference between the numbers is also high; this can also be seen in the graph of absolute differences in Figure A.9. The relative difference in these areas is relatively low. The graph of relative differences, which was calculated by dividing the absolute difference by the number of data entries from the MaStR, is shown in Figure 5.9.

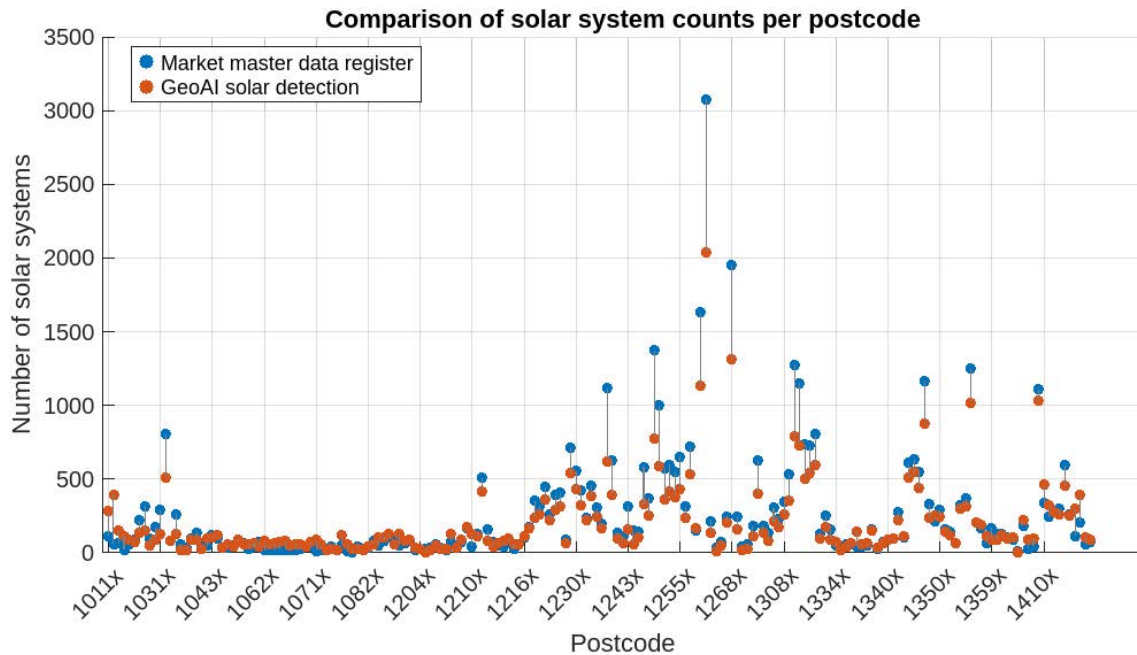


Figure 5.8: Comparison of Number of Solar Systems per Postcode

Since there are some postcode areas where the number of solar systems is relatively small (until postcode 1250x) there is a graph given with just this part of the comparison in Figure A.8. In most of these cases, there were more solar systems identified from the GeoAI than entries in the MaStR. Due to the small total number of solar systems, the relative difference is comparatively high in these postcode areas.

This means that the ratio of solar systems identified by GeoAI to registered solar systems changes as the number of solar systems increases. The higher the number of registered solar systems, the smaller the ratio. Conversely, the smaller the number of registered solar systems, the higher this ratio increases.

To sum up, this comparison showed that there are high differences in the number of solar systems



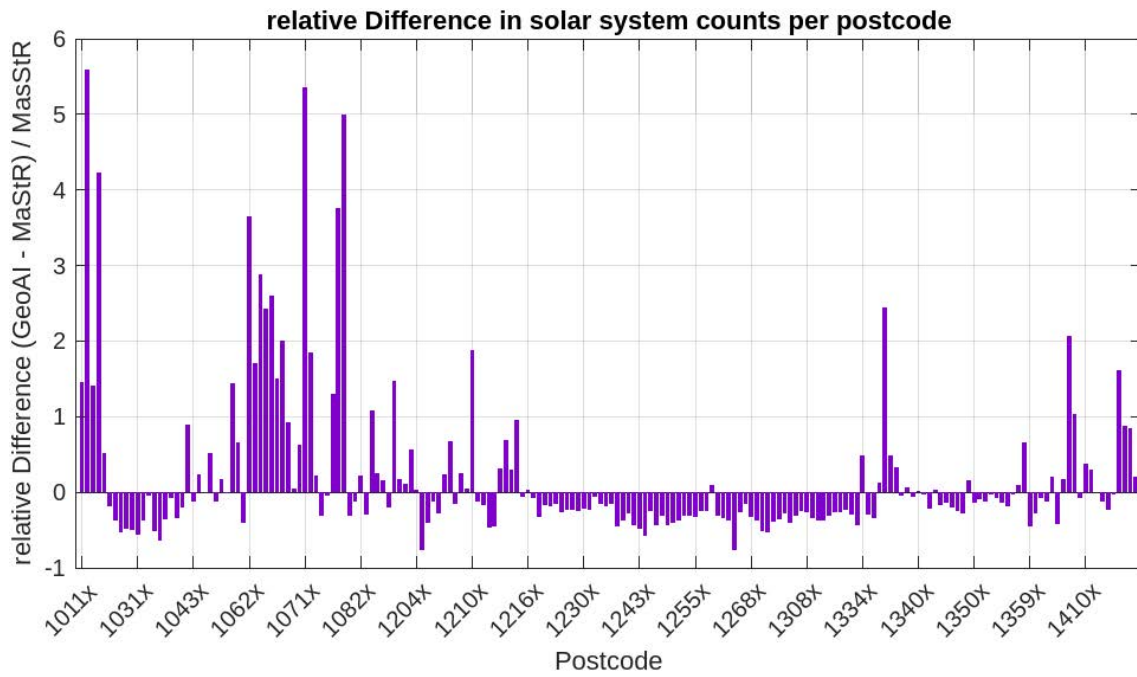


Figure 5.9: Relative Differnce of Number of Solar Systems per Postcode

per postcode. The solar system detection, therefore, did not provide a comparable and useful result. This high difference between the solar systems identified by the GeoAI and the official register was also the case in the study mentioned in Section 2.2 [4]. The justification for this discrepancy was attributed to the differing detection methodologies.

### 5.3 Result-Integration in VC Map

To integrate the final result (solar\_clean\_clipped.shp) into the VC Map, the file had to be converted into a \*.geojson file; QGIS [10] was used for this step. Then it was integrated into the map through the VC Publisher and the App Configurator. As background, the orthophoto of Berlin 2023 was processed in the Publisher to convert it into a streamable format. After building the Map in the App Configurator and bringing all together, the result looked as shown in Figures 5.10, A.10 and A.11.

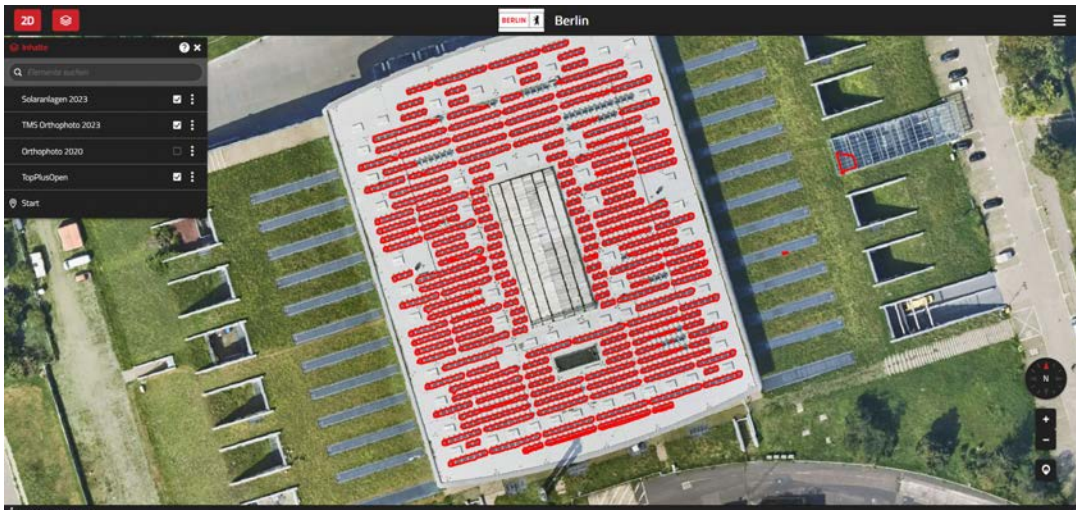


Figure 5.10: VC Map



# Chapter 6

## Discussion and Outlook

### 6.1 Discussion

It was difficult to determine the accuracy of the solar system detection. No analysis that was done could provide an accuracy value that applies to the entire dataset. Therefore, none of these analyses could be used as a basis for a generally valid conclusion. However, all of the examples, the calculated IoU of the individual tiles, and the comparison with the entries from the MaStR showed that GeoAI's solar system detection is inaccurate and insufficient.

This error is a systematic error. Systematic errors occur in a constant or proportional shift of the measured values. They influence the accuracy and correctness. That means these results could nevertheless allow valid statements to be made about progress over time [6]. When doing a time series analysis with bias correction, the change of installed solar systems remains valid over the years, although the error rate is constant.

The following proposals have the potential to yield more accurate results of the solar system detection using GeoAI:

- Hyperspectral or multispectral data could be used. This would help distinguish between solar systems and other objects, as already mentioned in the analysis. Although the size of the dataset would be much bigger, the solar system detection could be much more precise. Right now, the GeoAI is not able to handle this kind of data, so that would need to be adjusted and the model would need to be retrained.
- In the study by Kausika et al. [4] the training datasets were smaller (102.4 m) than the ones used in this project (900 m - 1000 m). Maybe the model would have been better if the training datasets were smaller; it would be interesting to test this.
- Another change that could increase the result quality significantly is to include more information in the training data, e.g. building footprints and objects that are specifically not solar systems. It would also be interesting if the distinction between different building categories (e.g. residential building, public building, factory, or monument) would increase the model quality. The GeoAI would also have to be adapted for this.
- As in the study by Kausika et al. [4], both orthophotos and TOs could be processed and then combined, as the combined result is said to have a higher accuracy than the individual results.

#### 6.1.1 Error Analysis

In the process of working on the project, a number of problems and errors occurred, which are discussed below; solutions are also proposed.

## 6.2. OUTLOOK

- The hand-drawn polygons used as training data were not entirely accurate and with higher precision, a more accurate model could have been trained. As said in paper [4] "the quality of the model is highly dependent on the quality of the training data". For example, it would have been more accurate to mark individual solar panels instead of the total area of all panels per roof. This would have meant substantial additional effort, but would probably also have made a significant difference in quality.
- Twelve of the 15 training runs weren't used. Insights were still gained from this, but some of these runs would not have been necessary and could have been skipped.
- Perhaps a different model would have delivered better results, because in retrospect, the examples in the model comparison (Appendix A.1) of Model 3 weren't the best. In order to make a decision for one model based on numbers, the IoUs of all models could have been calculated for all tiles of the training datasets. The highest mean value would have been that of the most accurate model.
- The first tile merging of the data preprocessing was also not necessary, since the data was cut into tiles during the solar system detection. Here, too, knowledge was gained: The raw data tiles could have been used instead of cutting new ones.
- When writing the Python skript for the data processing, which included this tile cutting, it was a mistake to trust Copilot [7] with the tile size. The tiles could have been larger, which would have shorten the runtime. Next time more research should be put in questions like this to minimize similar mistakes.
- In general the time schedule was planned tight, which made it difficult to comply. The data preprocessing and data processing took way longer than expected, which is the reason why no time series analysis could be performed.

## 6.2 Outlook

In addition to the suggestions for improvement mentioned in Section 6.1 for more accurate results, here are some ideas that could be investigated in the future:

- Possibly new polygons could be drawn from pieced polygons by using building footprints and LoD2 data. In cases like shown in the first Figure of Appendix Section A.2, the solar system could be marked entirely, if a script found the corners of this rectangular shape.
- It could be interesting to analyse the quality of the trained GeoAI model with another city.
- If there is a way to get all coordinates or addresses of MaStR entries, a further, more detailed comparison could be interesting.
- Instead of integrating 2-dimensional (2D) data in the VC Map, it is maybe possible to create 3D data by laying the solar systems on LoD2 buildings and this way get height data for each identified solar system. It would also be interesting to add an elevation of ~10 cm to the solar systems. This could be implemented using FME [12].
- A time series analysis of solar system installations in recent years would show if the systematic error could be balanced out with bias correction.
- A green roof analysis through a similar detection like the solar systems could be interesting. There probably will be difficulties distinguishing between green areas and green roofs but the building footprints could helpful here too.

## **6.3 Conclusion**

The objective of this interdisciplinary project with the topic "Solar System Detection with GeoAI" was achieved. The installed solar systems in Berlin were detected, a functional analysis workflow that can be used to detect installed solar systems in other cities was built and tested, and it was investigated during this project how accurate the identification of solar systems by GeoAI is. Accuracy was difficult to determine due to a lack of reference data. However, everything indicates that the individual results of the GeoAI's solar system detection are inaccurate and insufficient, despite its own model training. Nevertheless, a time series analysis with bias correction could show that the change in installed solar systems remains valid over the years.

# Appendix

## Appendix A

### Additional Figures

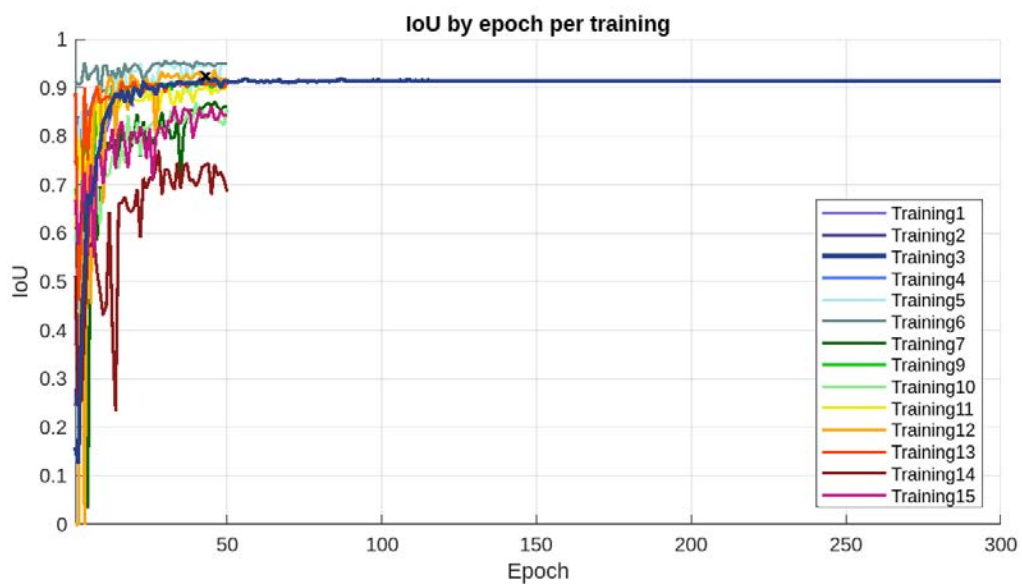


Figure A.1: IoUs by All Epochs per Training

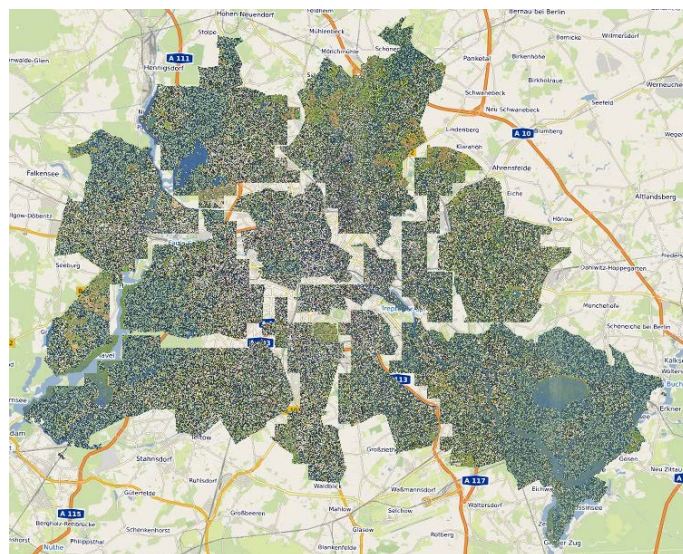


Figure A.2: Defective Preprocessing Result (North is up; edited in QGIS [10]; Layers from [1][9])

	Q_uuid	Q_clipped	Nutzung	Adresse
1	26c78b83-fdc7-...	yes	Wohnhaus	Odilostraße 21
2	6b2c3cb5-a134-...	no	Wohnhaus	Ansgarstraße 7
3	7e35bc87-268d-...	no	Wohnhaus	Falkentaler Steig 76A
4	8e494ba6-2731-...	no	Gebäude für Fo...	Robert-Rössle-Straße
5	8cb566eb-32c3-...	no	Wohnhaus	Klosterheider Weg 55
6	9e926a33-9433-...	yes	Wohnhaus	Gawanstraße 9
7	936c290a-44cc-...	yes	Wohnhaus	Viereckweg 93
8	e9fdabca-7954-...	no	Wohnhaus	Viereckweg 20
9	6cae6cdb-6baa-...	no	Krankenhaus	Hobrechtsfelder Chaussee
10	d661f20a-db7e-...	no	Wohnhaus	Hörstenweg 89

Figure A.3: Exemplary Attribute Table

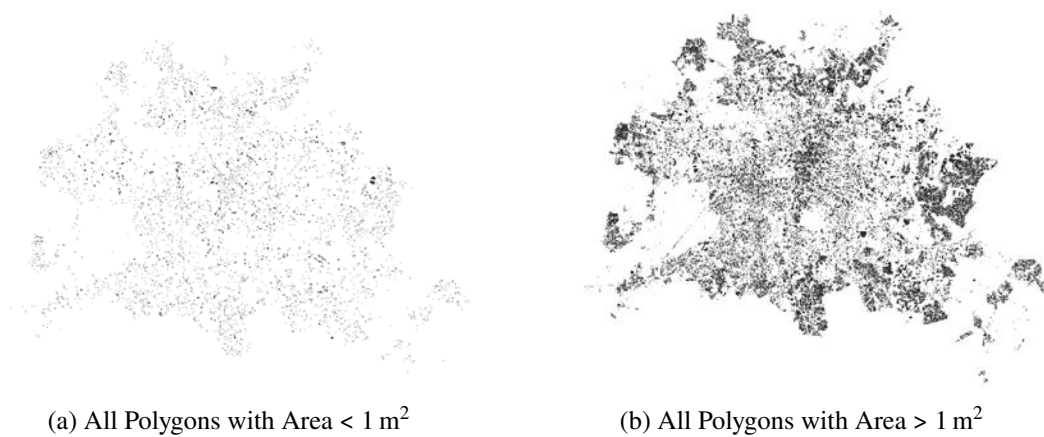


Figure A.4: Result of Filtering by Polygon Size (North is up; edited in QGIS [10])

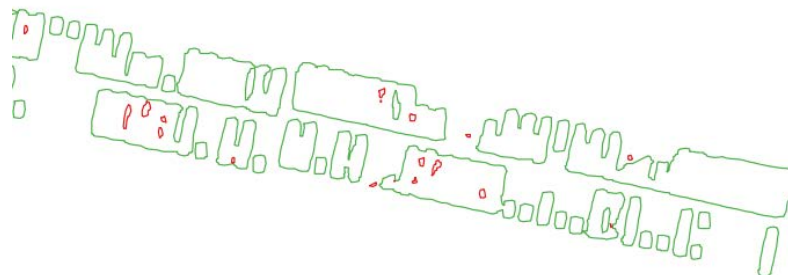


Figure A.5: Example: red Polygons with Area of < 1 m<sup>2</sup> (North is up; edited in QGIS [10])

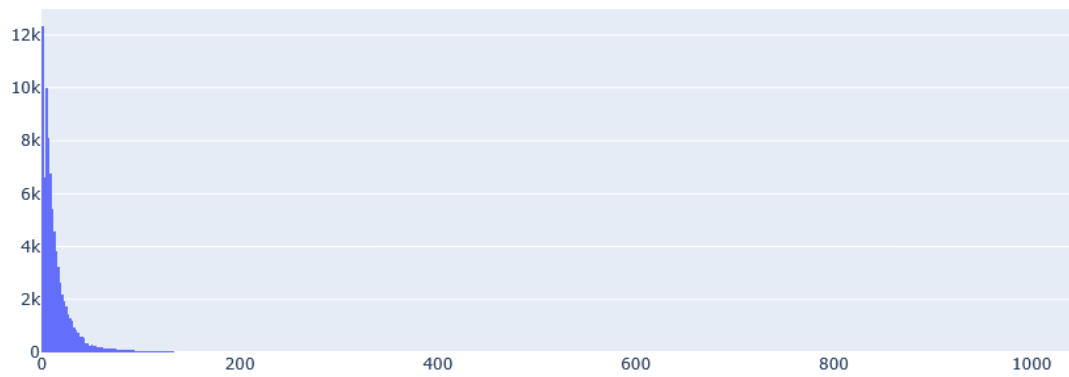


Figure A.6: Histogram of Size of Polygons in m<sup>2</sup> (solar\_clean\_clipped.shp)

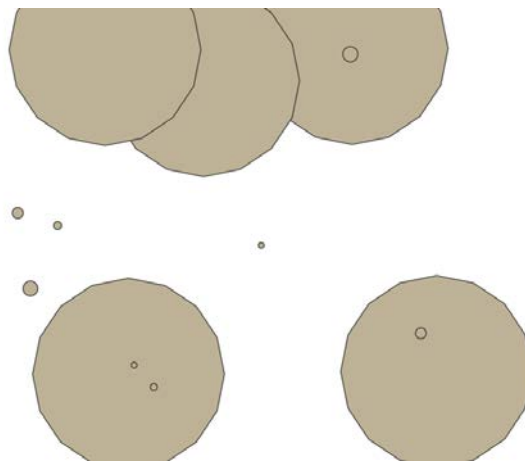


Figure A.7: Visualization of Faulty area\_sum Calculation (edited in QGIS [10])

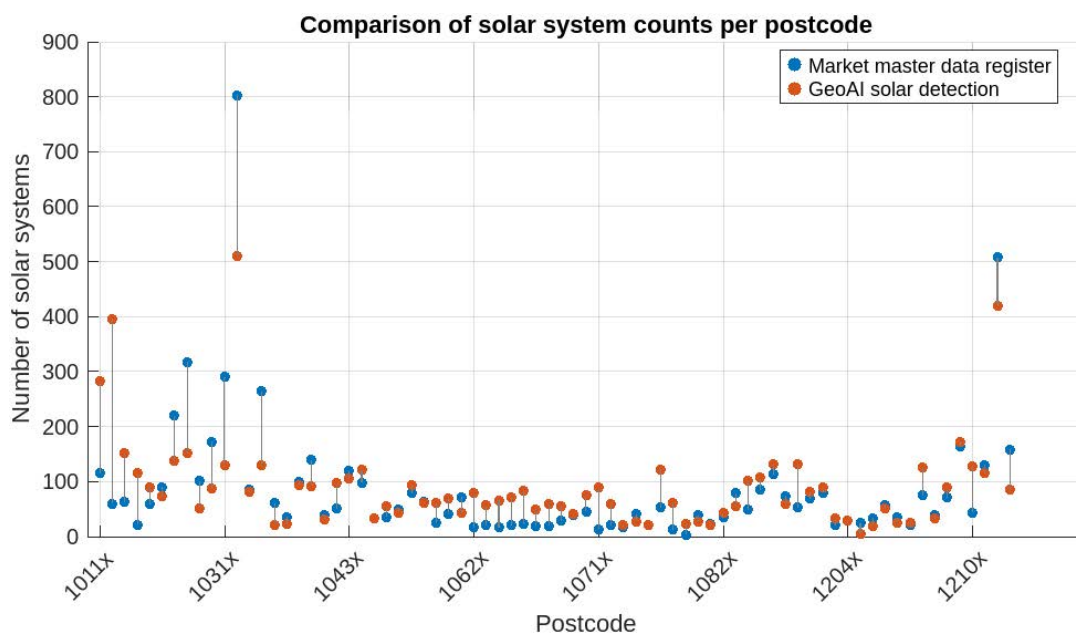


Figure A.8: Comparison of Number of Solar Systems per Postcode (until postcode 1250x)



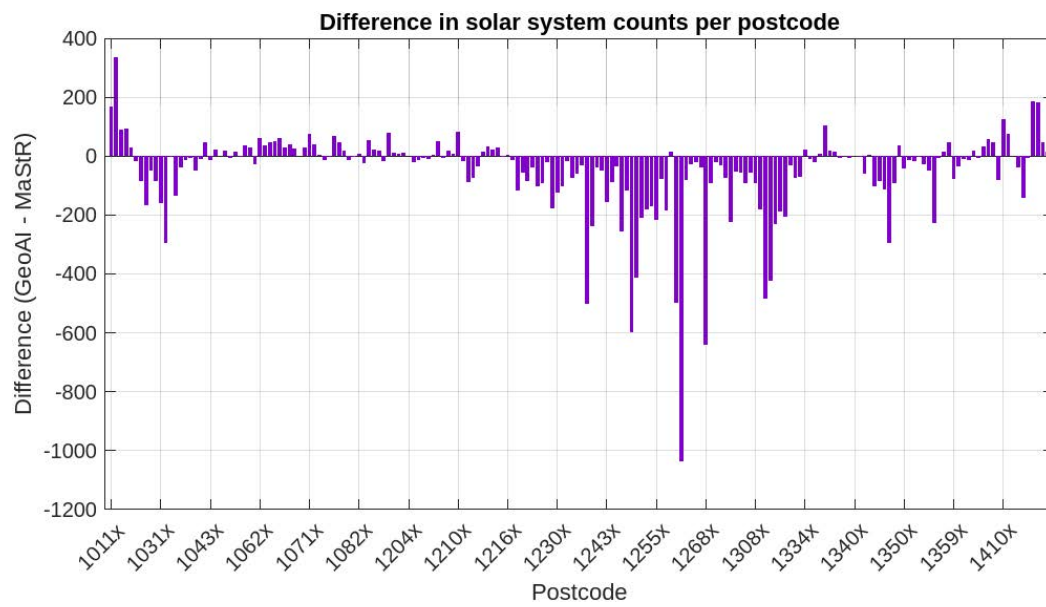


Figure A.9: Absolute Difference of Number of Solar Systems per Postcode

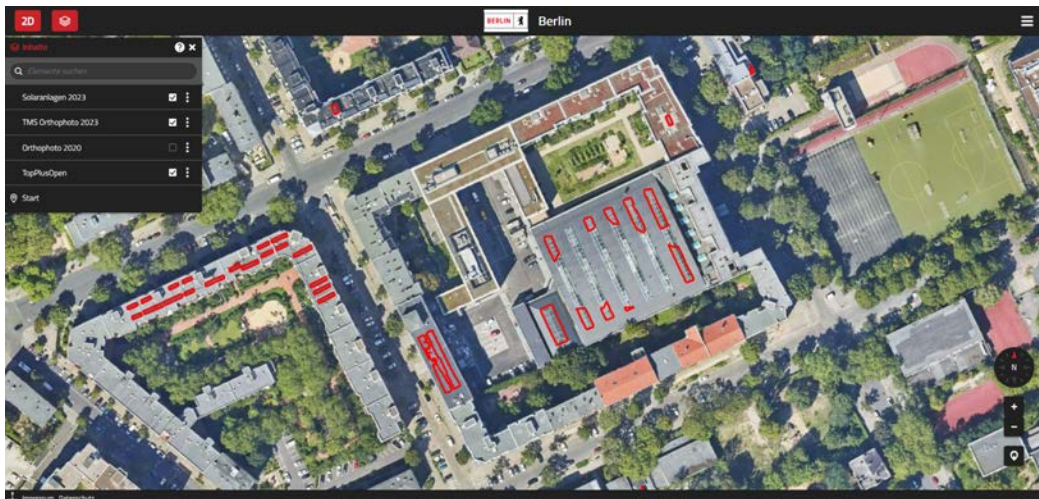


Figure A.10: VC Map 2

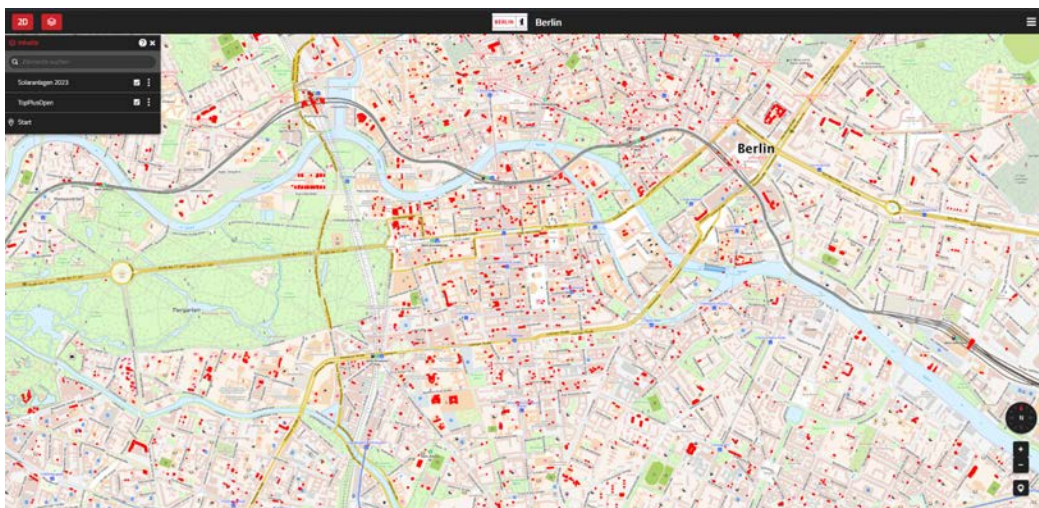


















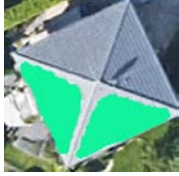















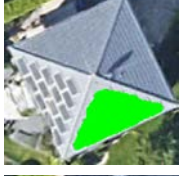







Figure A.11: VC Map 3





















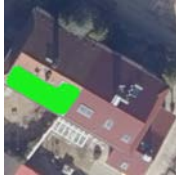
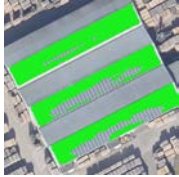


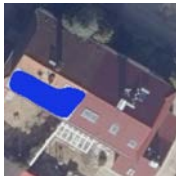


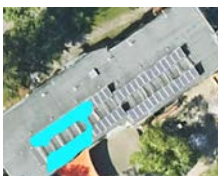
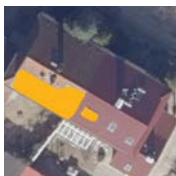



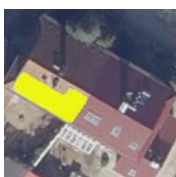

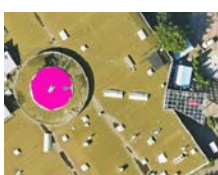

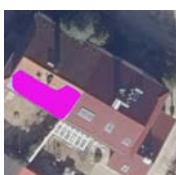

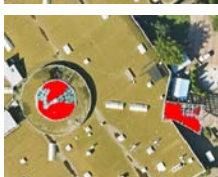
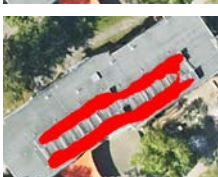
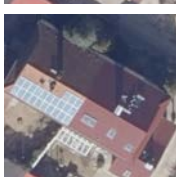

## A.1 Model Comparison: Exemplary Buildings with & without Solar Systems

Dataset	Berlin Tile without Training	Berlin Tile without Training	Berlin Tile without Training
Building			
Pre-trained Mask R-CNN model			
Modell 2 with TD1			
Modell 3 with TD1			
Modell 4 with TD2			
Modell 5 with TD5			
Modell 6 with TD6			
Modell 7 with TD3			
Modell 9 with TD4			



Dataset	TD1	TD1	TD1	TD3
Building				
Training Data				
Pre-trained Mask R-CNN model				
Modell 2 with TD1				
Modell 3 with TD1				
Modell 4 with TD2				
Modell 5 with TD5				
Modell 6 with TD6				
Modell 7 with TD3				
Modell 9 with TD4				

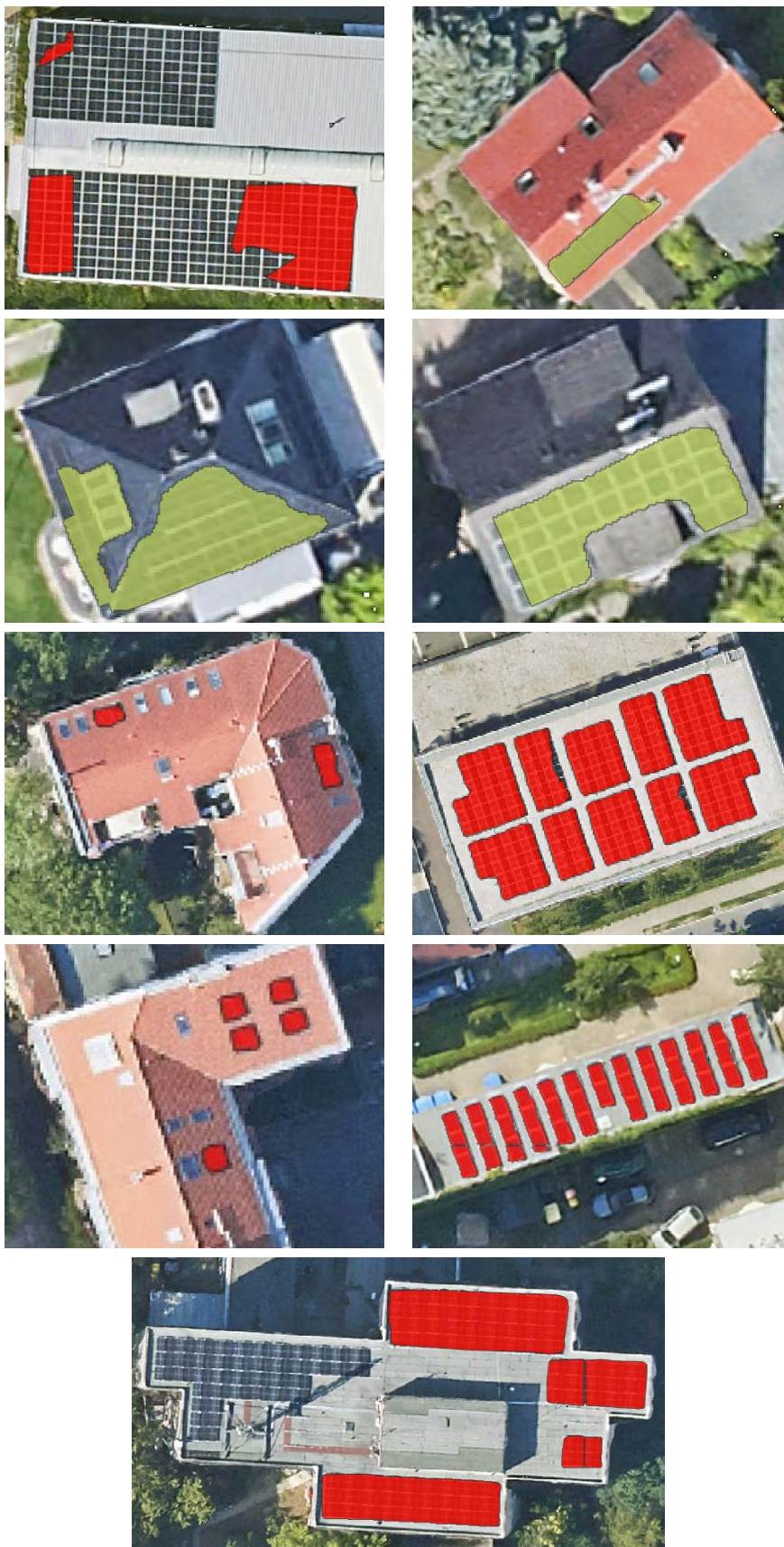


Dataset	TD3	TD3	TD5	TD5
Building				
Training Data				
Pre-trained Mask R-CNN model				
Modell 2 with TD1				
Modell 3 with TD1				
Modell 4 with TD2				
Modell 5 with TD5				
Modell 6 with TD6				
Modell 7 with TD3				
Modell 9 with TD4				



## A.2 Examples of Correct vs. False Identification

Polygons of identified solar systems are visible in red or green; North is up; edited in QGIS [10]; Layer from [1].



## Appendix B

### Additional Tables

Table B.1: Hardware Specifications

Specification	PC226	LAP183
Processor	Intel Core i7-9700 @ 3.00GHz (8 cores)	Intel Core i7-11800H @ 2.30GHz (8 cores)
Installed RAM	32.0 GB	32.0 GB
System Type	64-bit OS, x64-based processor	64-bit OS, x64-based processor
Operating System	Windows 11 Business	Windows 11 Business
Integrated GPU	Intel UHD Graphics 630	Intel UHD Graphics
Dedicated GPU	NVIDIA GeForce GTX 1070	NVIDIA T1200 Laptop GPU
Dedicated GPU Memory	8.0 GB	4.0 GB
Shared GPU Memory	15.9 GB	15.9 GB
Total GPU Memory	23.9 GB	19.9 GB
GPU Driver Version	32.0.15.7640	32.0.101.6647

Table B.2: Used Software Tools

Software	Description	Version
JupyterLab via Anaconda	Web-based interactive development environment for notebooks, code and data	4.42 in conda 24.11.3
Python	Programming language for automation and geospatial data analysis	3.12.9
GeoAI [16]	Python platform that combines AI with geospatial data analysis	0.6.0
Quantum Geographic Information System (QGIS) [10]	Open geoinformation system software for viewing, editing and analyzing spatial data	3.40.6-Bratislava
Feature Manipulation Engine (FME) [12]	Load and transformation software platform developed and maintained by Safe Software Inc.	FME(R) 2024.2.4.0
VC Publisher [14]	Product by VCS for data preparation and publishing map	5.3.23
App Configurator [14]	Product by VCS for data integration into interactive map	v6.1

Table B.3: IoUs per Postprocessing Step

Dataset	solar.shp	solar_clean.shp	solar_clean_clipped.shp	solar_clean_clipped_o1.shp
TD1	0.7906	0.7908	0.7517	0.7514
TD2	0.6936	0.6934	0.6955	0.6959
TD3	0.3772	0.3775	0.3877	0.3871
TD4	0.7286	0.7285	0.7749	0.7742
Mean of IoU	0,6475	0,64755	0,65245	0,65215

## Appendix C

# Source Codes and FME workflows

Listing C.1: Model Training for Solar System Detection (Based on Example Script [16])

```
1 # import libraries
2 import geoai
3 import rasterio
4
5 # data source
6 train_raster = "Berlin_georeferenced.tif"
7 train_vector = "Berlin_training_solarpanels.geojson"
8 pretrained = "output_berlin3/models/best_model.pth"
9
10 # output path
11 out_folder = "output_berlin4"
12 tiles = geoai.export_geotiff_tiles(
13     in_raster=train_raster,
14     out_folder=out_folder,
15     in_class_data=train_vector,
16     tile_size=512,
17     stride=256,
18     buffer_radius=0,
19 )
20
21 # training with pretrained model
22 geoai.train_MaskRCNN_model(
23     images_dir=f"{out_folder}/images",
24     labels_dir=f"{out_folder}/labels",
25     output_dir=f"{out_folder}/models",
26     num_channels=4, # number of spectral bands of imagery
27     pretrained=True, # false if training without pretrained model
28     pretrained_model_path=pretrained,
29     batch_size=4,
30     num_epochs=50, # the more the longer it takes
31     learning_rate=0.005,
32     val_split=0.2, # 80% for training, 20% for validation
33 )
```

Listing C.2: Solar Panel Detection with Pre-trained Mask R-CNN Model  
(Based on Example Script [16])

```
1 # import libraries
2 import geoai
3 import rasterio
4
5 # data source
6 raster = "../10_TrueOrthophoto_RGBI_TIF_9cm_wholeCity/ganzBerlin.tif"
7 geoai.print_raster_info(raster)
```

```

8 geoai.view_raster(raster)
9
10 # solar system detection
11 detector = geoai.SolarPanelDetector() #initialization
12 output = "../SolaranlagenBerlin/solar_panels_pretrained_ganzBerlin.tif"
13
14 masks = detector.generate_masks(
15     raster,
16     output_path=output,
17     confidence_threshold=0.4, # betw. 0-1 ; the higher, the more confident,
18     # the more selective, the more smaller number of polygons
19     mask_threshold=0.5, # if there are many panels missing (not detected),
20     # increase number
21     min_object_area=100, # smallest possible area of solar panels
22     overlap=0.25,
23     chip_size=(400, 400),
24     batch_size=4,
25     verbose=False,
26 )

```

Listing C.3: Solar Panel Detection (Based on Example Script [16]) with Model 3 including Tile Cutting (Coded with Copilot [7])

```

1 # import libraries
2 import geoai
3 import os
4 import rasterio
5 from rasterio.windows import Window
6 from math import ceil
7 from pathlib import Path
8
9 # configuration
10 input_cog_path = "../10_TrueOrthophoto_RGBI_TIF_9cm_wholeCity/ganzBerlin.tif"
11 # Path to raster
12 output_tile_dir = "../SolaranlagenBerlin/tiles" # folder for raster tiles
13 output_result_dir = "../SolaranlagenBerlin/results" # folder for results
14 model_path = "../output_berlin3/models/best_model.pth"
15 tile_size = 2048 # tilesize in pixels
16 overlap_ratio = 0.1 # 10% overlap
17 start_i = 0
18 start_j = 0
19
20 # prepare folders
21 os.makedirs(output_tile_dir, exist_ok=True)
22 os.makedirs(output_result_dir, exist_ok=True)
23
24 # open raster
25 with rasterio.open(input_cog_path) as src:
26     width = src.width
27     height = src.height
28     meta = src.meta.copy()
29
30 # Calculate step size (taking overlap into account)
31 step = int(tile_size * (1 - overlap_ratio))
32
33 # Number of tiles in the x and y directions
34 num_tiles_x = ceil((width - tile_size) / step) + 1
35 num_tiles_y = ceil((height - tile_size) / step) + 1
36 print(f"Create {num_tiles_x * num_tiles_y} tiles...")
37
38 for i in range(num_tiles_y):
39     for j in range(num_tiles_x):
40         if i < start_i or (i == start_i and j < start_j):

```



```

40         continue # skip
41
42     x_off = j * step
43     y_off = i * step
44
45     # Limitation to image size
46     win_width = min(tile_size, width - x_off)
47     win_height = min(tile_size, height - y_off)
48
49     window = Window(x_off, y_off, win_width, win_height)
50     transform = src.window_transform(window)
51
52     tile_data = src.read(window=window)
53
54     tile_filename = f"tile_{i}_{j}.tif"
55     tile_path = os.path.join(output_tile_dir, tile_filename)
56     output_path = os.path.join(output_result_dir,
57                                f"result_{i}_{j}.tif")
58
59     tile_meta = meta.copy()
60     tile_meta.update({
61         "height": win_height,
62         "width": win_width,
63         "transform": transform
64     })
65
66     # If result already exists, skip
67     if os.path.exists(output_path):
68         print(f"Skip tile {i}, {j} - result already exists.")
69         continue
70
71     with rasterio.open(tile_path, "w", **tile_meta) as dst:
72         dst.write(tile_data)
73
74     # GeoAI function: solar system detection
75     geoai.object_detection(
76         tile_path,
77         output_path,
78         model_path,
79         window_size=512,
80         overlap=256,
81         confidence_threshold=0,
82         batch_size=4,
83         num_channels=4,
84     )
85
86 print("Finished with the tiles.")

```

Listing C.4: Postprocessing with GeoAI [16]

```

1  # import libraries
2  import geoai
3
4  # data source
5  raster = "../10_TrueOrthophoto_RGBI_TIF_9cm_wholeCity/ganzBerlin.tif"
6  output = "../SolaranlagenBerlin/solar_panels_pretrained_ganzBerlin.tif"
7
8  # visualization
9  geoai.view_raster(
10     output,
11     indexes=[2], # output mask contains 2 bands: 1. binary band (1/0 solar
12                 # panel or not) 2. confidence / propability of each detection -> color
13                 # bar: red (low confidence) to yellow (high confidence)
14     colormap="autumn",

```

```

13     layer_name="Solar Panels",
14     basemap=raster_path,
15 )
16
17 # vectorize masks
18 gdf = geoai.orthogonalize(
19     # convert raster to vector
20     output,
21     output_path="solar_panel_masks.geojson",
22     epsilon=0.2 # how smooth the polygons be? default 0.2; increase if its not
                  # simplified enough, decrease if its simplified too much
23 )
24
25 # calculate geommetric properties
26 gdf = geoai.add_geometric_properties(gdf)
27
28 # exemplary filter
29 gdf_filter = gdf[(gdf["elongation"] < 10) & (gdf["area_m2"] > 5)]
30 gdf_filter["area_m2"].hist()
31 gdf_filter["area_m2"].describe()
32 gdf_filter["area_m2"].sum()
33
34 # visulizing and saving result
35 geoai.view_vector_interactive(gdf, tiles=raster)
36 gdf_filter.to_file("solar_panels.geojson")

```

Listing C.5: Calculate IoU for Training Dataset Tiles (Coded with Copilot [7])

```

1 # import libraries
2 import geopandas as gpd
3 from shapely.ops import unary_union
4
5 # IoU calculation
6 def calculate_iou(manual_path, ai_path):
7     manual_gdf = gpd.read_file(manual_path)
8     ai_gdf = gpd.read_file(ai_path)
9
10    # CRS comparison
11    if manual_gdf.crs != ai_gdf.crs:
12        ai_gdf = ai_gdf.to_crs(manual_gdf.crs)
13
14    # Merge geometries
15    manual_union = unary_union(manual_gdf.geometry)
16    ai_union = unary_union(ai_gdf.geometry)
17
18    # Intersection and union area
19    intersection = manual_union.intersection(ai_union)
20    union = manual_union.union(ai_union)
21
22    # calculate IoU
23    iou = intersection.area / union.area if union.area != 0 else 0
24    return iou
25
26 # file paths
27 tiles = {
28     "005101": ("../Berlin3_training_solarpanels.geojson",
29              "../SolaranlagenBerlin/results/6
30              IoU_calculation/cleaned/005101.shp/005101.shp"),
31     "005119": ("../Berlin4_training_solarpanels.geojson",
32              "../SolaranlagenBerlin/results/6
33              IoU_calculation/cleaned/005119.shp/005119.shp"),
34     "915272": ("../Berlin2_training_solarpanels.geojson",
35              "../SolaranlagenBerlin/results/6
36              IoU_calculation/cleaned/915272.shp/915272.shp"),

```

```

31     "924047": ("../Berlin1_training_solarpanels.geojson",
32               "../SolaranlagenBerlin/results/6
33               IoU_calculation/cleaned/924047.shp/924047.shp")
34 }
35 # print results
36 for tile_id, (manual_file, ai_file) in tiles.items():
37     try:
38         iou_value = calculate_iou(manual_file, ai_file)
39         print(f"IoU of tile {tile_id}: {iou_value:.4f}")
40     except Exception as e:
41         print(f"error for tile {tile_id}: {e}")

```

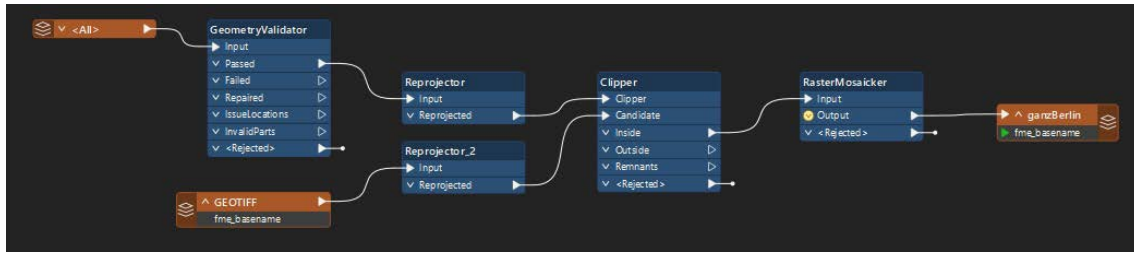
Listing C.6: Writing \*.geojson with MaStR Data (Coded with Copilot [7])

```

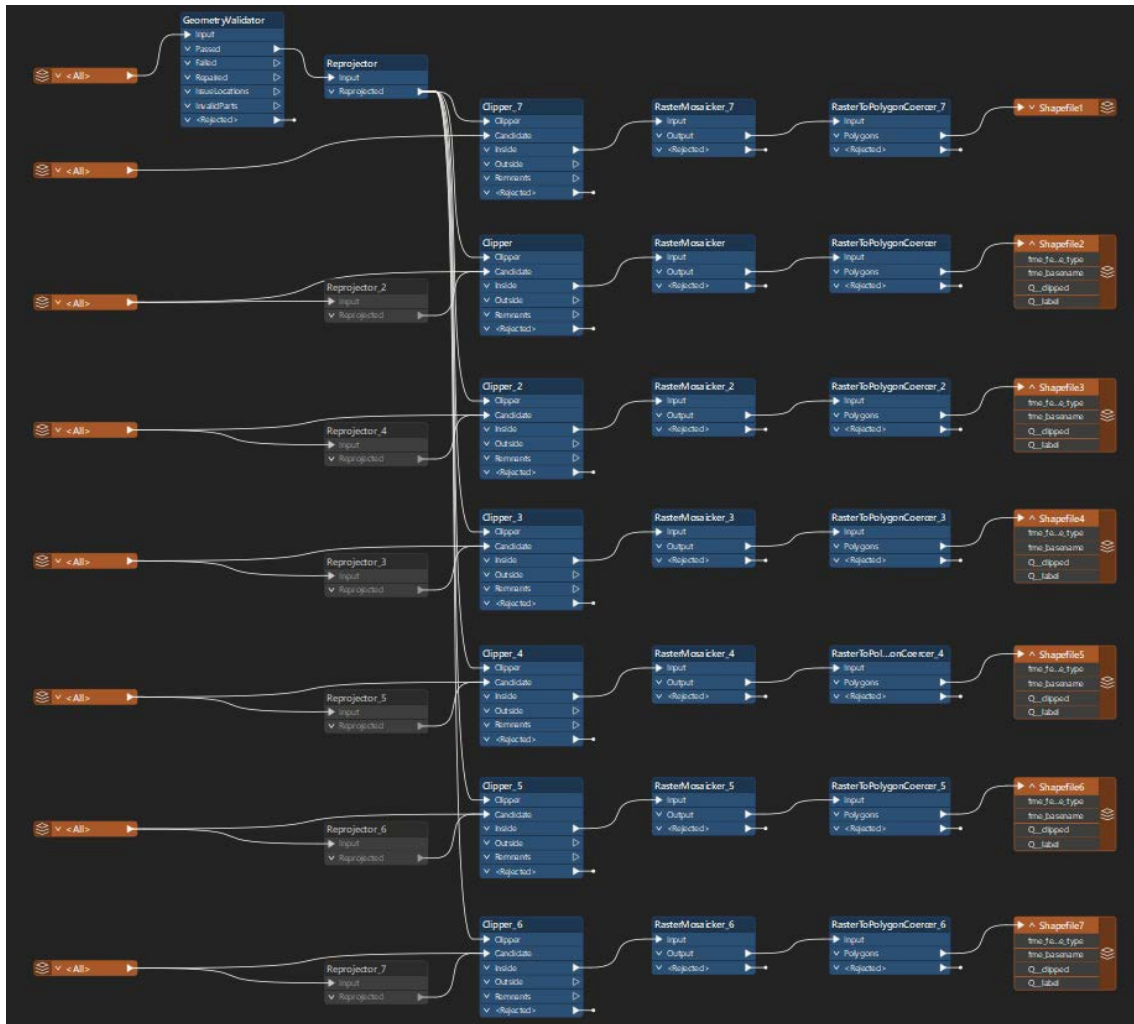
1  # Load CSV file
2  df = pd.read_csv("StromerzeugerSolarKoordinaten.csv", sep=';')
3
4  # Identify relevant columns
5  lat_col = [col for col in df.columns if 'breit' in col.lower()][0]
6  lon_col = [col for col in df.columns if 'lang' in col.lower() or 'lon' in
7             col.lower()][0]
8  kwp_col = [col for col in df.columns if 'installierte leistung' in
9             col.lower()][0]
10
11 # Convert decimal numbers to point notation and convert to float
12 df[lat_col] = df[lat_col].astype(str).str.replace(',', '.').astype(float)
13 df[lon_col] = df[lon_col].astype(str).str.replace(',', '.').astype(float)
14 df[kwp_col] = df[kwp_col].astype(str).str.replace(',', '.').astype(float)
15
16 # Group by coordinates and sum installed capacity
17 grouped = df.groupby([lat_col, lon_col], as_index=False)[kwp_col].sum()
18
19 # Calculate area (1 kWp = 6 m^2)
20 grouped["Flaeche_m2"] = grouped[kwp_col] * 6
21
22 # Create point geometries
23 geometry = [Point(xy) for xy in zip(grouped[lon_col], grouped[lat_col])]
24 gdf_points = gpd.GeoDataFrame(grouped, geometry=geometry, crs="EPSG:4326")
25
26 # Transform to metric coordinate system
27 gdf_projected = gdf_points.to_crs(epsg=3857)
28
29 # Generate circles with area (radius = sqrt(area / pi))
30 gdf_projected["geometry"] = [
31     Point(pt.x, pt.y).buffer(math.sqrt(area / math.pi))
32     for pt, area in zip(gdf_projected.geometry, gdf_projected["Flaeche_m2"])
33 ]
34
35 # Back to WGS84 for GeoJSON export
36 gdf_final = gdf_projected.to_crs(epsg=4326)
37
38 # Export as GeoJSON
39 gdf_final.to_file("solaranlagen_flaechen_gruppiert.geojson", driver="GeoJSON")

```

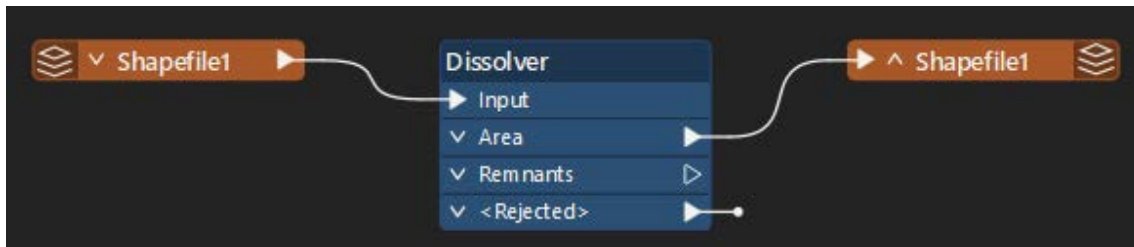
Listing C.7: FME Workflow: Data Preprocessing - Tile Merging and Clipping by Berlin-shape



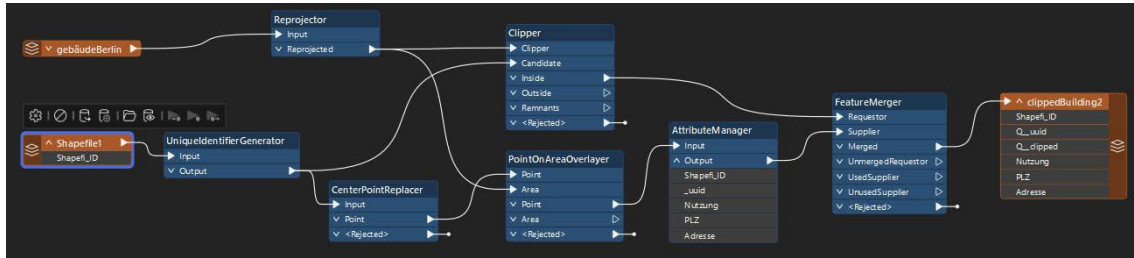
Listing C.8: FME Workflow: Result Postprocessing - Tile Merging and Conversion to \*.shp per Batch



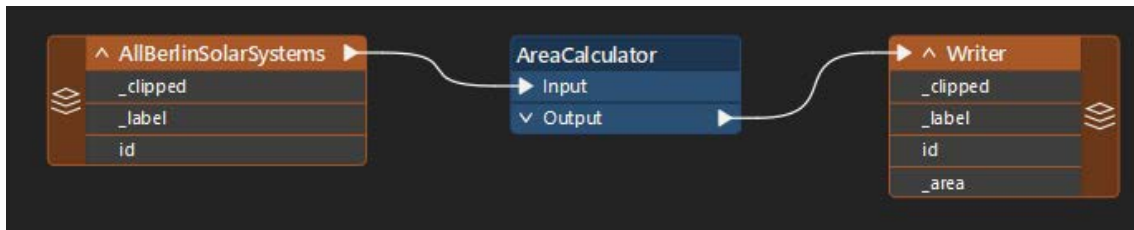
Listing C.9: FME Workflow: Result Postprocessing - Merging Overlapping Polygons



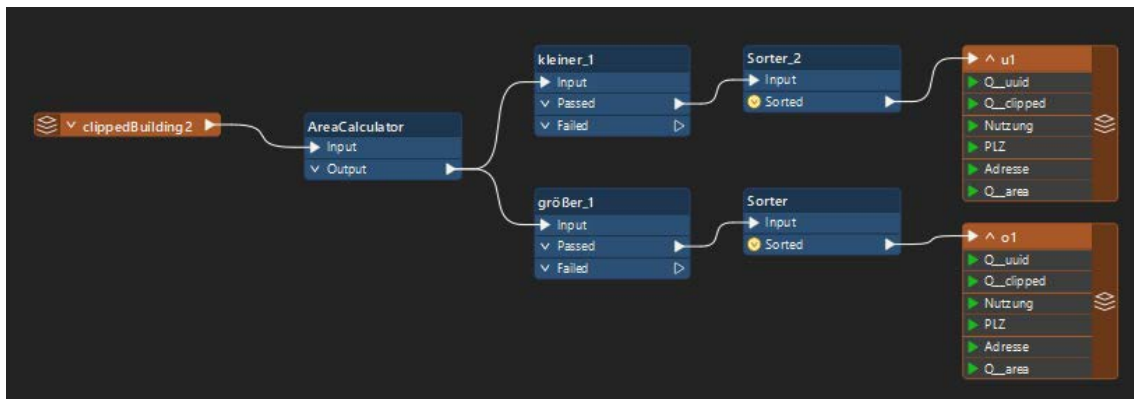
Listing C.10: FME Workflow: Result Postprocessing - Clipping Polygons by Building Footprint and Adding Attribute Address



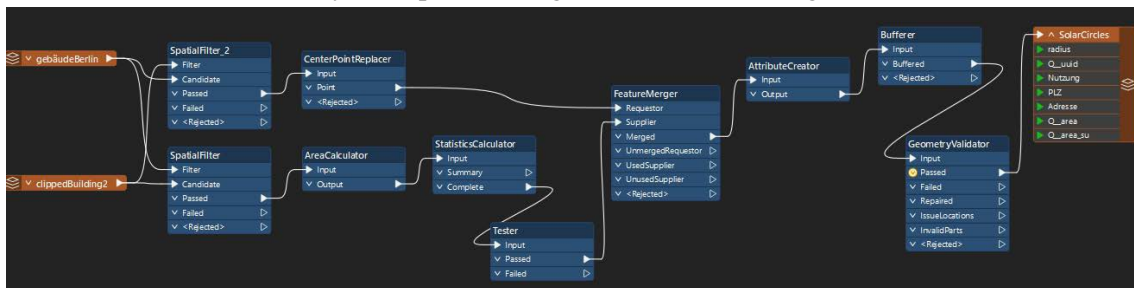
Listing C.11: FME Workflow: General Area Calculation



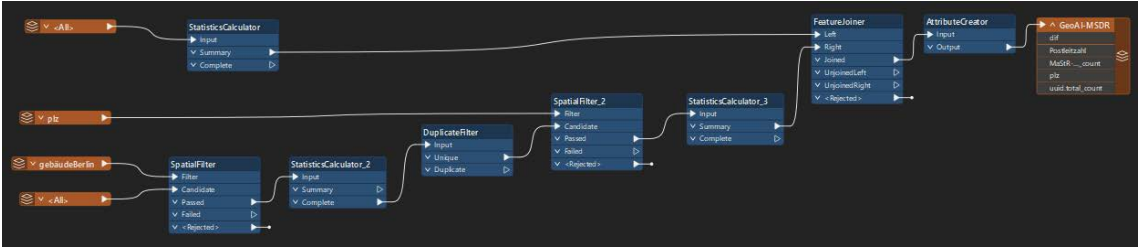
Listing C.12: FME Workflow: Result Postprocessing - Sorting Polygons area < 1 m<sup>2</sup> and area > 1 m<sup>2</sup>



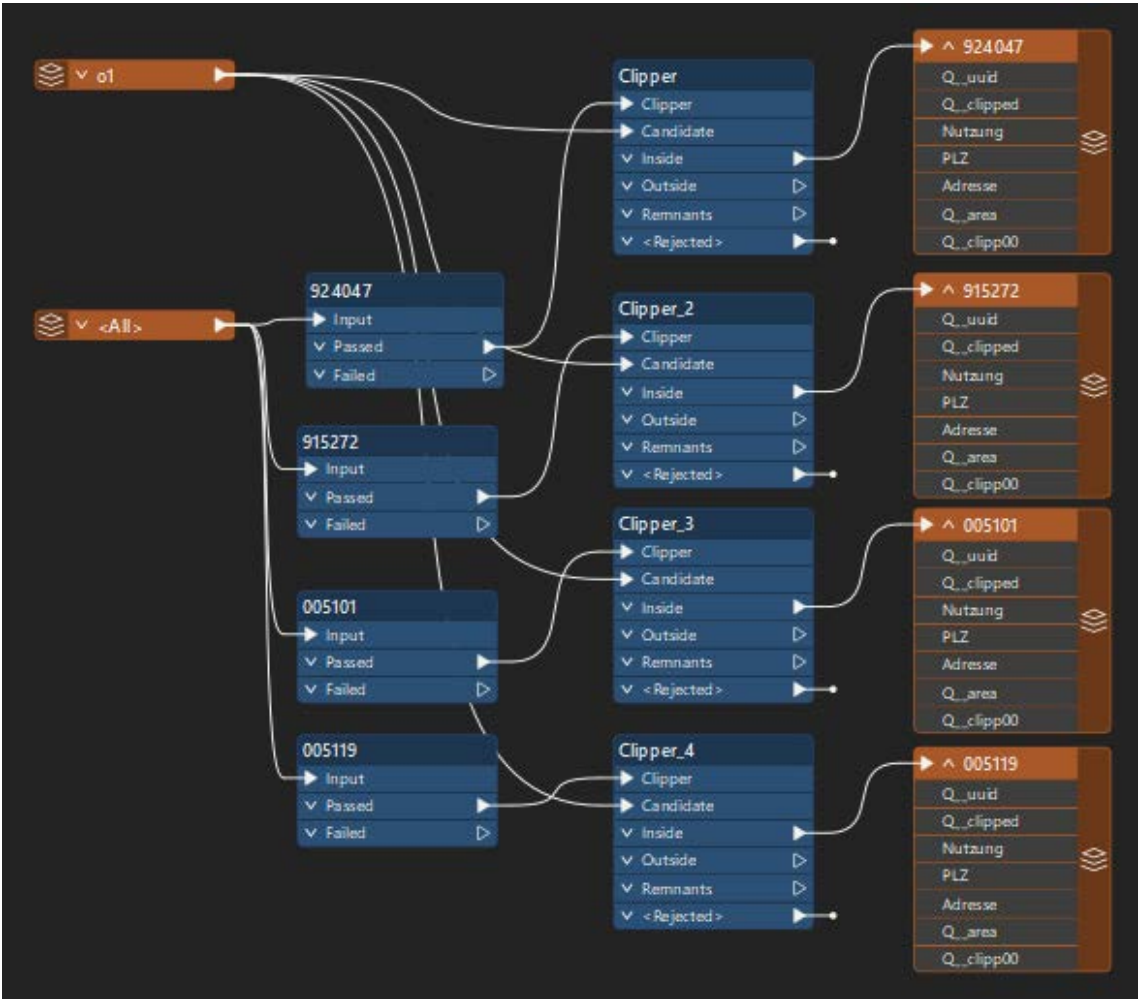
Listing C.13: FME Workflow: Result Analysis - Drawing Circles in Size of Total Area of Solar Systems per Building in Center of Buildings



Listing C.14: FME Workflow: Result Analysis - Counting Number of Solar Systems per Postcode Area for Comparison with MaStR



Listing C.15: FME Workflow: Result Analysis - Clipping for IoU Calculation in Tiles of Training Data



# Appendix D

## AI Prompts

The following prompts were used to generate Python scripts with Copilot [7].

Python script for tile cutting and Solar System Detection with GeoAI:

- Error reading raster: Unable to allocate 7.54 GiB for an array with shape (84882, 95422) and data type bool

Ich möchte die geoai <https://geoai.gishub.org/> dafür nutzen Solaranlagen in ganz Berlin zu identifizieren. Dafür habe ich Orthophotos mit 9cm Auflösung von Berlin und die KI auf Berlin trainiert. Jetzt bekomme ich diesen oben genannten Fehler. Wie könnte ich das lösen? Meinst du es wäre eine Möglichkeit den Datensatz in mehrere, größere Kacheln zu schneiden (gerade so klein genug, dass es funktioniert) und eine Schleife zu bauen, die diese Kacheln einzeln durchläuft. Im besten Fall würden die Polygone der Solaranlagen auch direkt in eine gemeinsame Datei eingetragen werden. Ist das möglich? Oder hast du eine bessere Idee?

**English translation:** Error reading raster: Unable to allocate 7.54 GiB for an array with shape (84882, 95422) and data type bool

I want to use the geoai <https://geoai.gishub.org/> to identify solar installations in Berlin. For this purpose, I have trained the AI on Berlin using orthophotos with a resolution of 9 cm. Now I am getting the above error. How could I solve this? Do you think it would be possible to cut the dataset into several larger tiles (just small enough that it works) and build a loop that runs through these tiles individually? In the best case, the polygons of the solar installations would also be entered directly into a common file. Is that possible? Or do you have a better idea?

- Das Orthophoto ist schon ein COG, aber es funktioniert trotzdem nicht, ich würde das mit dem Python-Skript gerne versuchen. Welche Kachelgröße würdest du vorschlagen? Ich weiß nicht was sinnvoll wäre... nicht zu groß und nicht zu klein. Geojson wäre am besten, aber die Funktion der GeoAI, die die Solaranlagen identifiziert, gibt ein tif aus, dabei könnte es auch bleiben, die Konvertierung kann ich danach auch machen

**English translation:** The orthophoto is already a COG, but it still doesn't work. I would like to try the Python script. What tile size would you suggest? I don't know what would be suitable... not too large and not too small. Geojson would be best, but the GeoAI function that identifies the solar panels outputs a tif, so it could stay that way. I can do the conversion afterwards.

- `geoai.object_detection( tile_path, output_path, model_path, window_size=512, overlap=256, confidence_threshold=0, batch_size=4, num_channels=4, )`

Das ist die Funktion. Man gibt den raster-path, den output-path, den model-path und ein paar parameter ein. Der Output wird also nicht von der Funktion zurückgegeben, sondern direkt gespeichert. Wie kann man das in dein Skript einfügen, dass die Ergebnisse korrekt



gespeichert werden?

**English translation:** `geoai.object_detection( tile_path, output_path, model_path, window_size=512, overlap=256, confidence_threshold=0, batch_size=4, num_channels=4, )`

That's the function. You enter the raster path, the output path, the model path, and a few parameters. The output is not returned by the function, but saved directly. How can you add this to your script so that the results are saved correctly?

Python script for calculating IoU of four tiles:

- Kannst du mir ein Python Skript schreiben, von dem die IoU (Intersection over Union) von 4 Kacheln berechnet wird? Ich habe die Kacheln 005101, 005119, 915272, 924047 je einmal als Vektordatensatz (geojson) mit Trainingsdaten (manuell identifizierte Solaranlagen) und einmal als Vektordatensatz (im Moment shp-file, aber könnte es zu geojson konvertieren) mit den Polygonen, die die GeoAI als Solaranlagen eingezeichnet hat. Der IoU-Wert (einzeln für die 4 Kacheln) soll mir einen Genauigkeitswert liefern, mit dem ich auf die Genauigkeit der Gesamtfläche schließen will.

**English translation:** Can you write me a Python script that calculates the IoU (intersection over union) of 4 tiles? I have the tiles 005101, 005119, 915272, 924047, once as a vector dataset (geojson) with training data (manually identified solar installations) and once as a vector dataset (currently shp file, but could be converted to geojson) with the polygons that GeoAI has marked as solar installations. The IoU value (individually for the 4 tiles) should provide me with an accuracy value that I can use to estimate the accuracy of the total area.

Python script for entering MaStR data in GeoJSON:

- Mit welchem Tool (FME, Python, QGIS) kann ich am besten einen Vektordatensatz erstellen aus ein csv-Datei, in der die Koordinaten bzw. Adressen stehen und mittels installierte Leistung die Fläche der Solaranlagen / Polygonen berechnet werden können? Also ich habe csv-Daten aus dem Marktsstammdatenregister zu Solaranlagen, da stehen auch noch sehr viel mehr Daten drin außer denen, die ich für den Vektordatensatz brauche. Ich möchte einen Vektordatensatz, in dem mindestens die Koordinaten markiert sind, zusätzlich könnten Kreise eingezeichnet werden, die der berechneten Fläche aus der installierten Leistung berechnet wurden

**English translation:** Which tool (FME, Python, QGIS) is best for creating a vector dataset from a CSV file containing coordinates or addresses, which can be used to calculate the area of solar installations/polygons based on installed capacity? I have CSV data from the market master data register on solar installations, which contains a lot more data than I need for the vector dataset. I would like a vector dataset in which at least the coordinates are marked, and circles could also be drawn that were calculated from the installed capacity.

- Die Datei heißt StromerzeugerSolarKoordinaten.csv

Gib mir bitte ein Python Skript dafür

**English translation:** The file is called StromerzeugerSolarKoordinaten.csv

Please provide me with a Python script for this.

- Wenn an einem Ort mehrere Solaranlagen sind, sollen die Flächen addiert werden, sodass nicht mehrere Kreise übereinander liegen

**English translation:** If there are several solar installations in one location, the areas should be added together so that multiple circles do not overlap

# Directories

# Bibliography

- [1] AeroWest GmbH. *AeroWest Geospatial Services*. <https://www.aerowest.de>. 2025.
- [2] Bundesnetzagentur. *Öffentliche Einheitenübersicht [Marktstammdatenregister]*. <https://www.marktstammdatenregister.de/MaStR>. Retrieved on July 2nd, 2025.
- [3] DeepL SE. *DeepL Translator*. <https://www.deepl.com>, <https://www.deepl.com/de/write>. Used in July 2025. 2025.
- [4] Bala Bhavya Kausika et al. “GeoAI for detection of solar photovoltaic installations in the Netherlands”. In: *Energy and AI* 6 (2021), p. 100111.
- [5] S. Kocaman, E. Gülch, and M. Yakar. “Quality aspects of true orthophoto in urban areas”. In: *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*. Vol. XLIII-B5-2020. 2020, pp. 191–196. DOI: 10.5194/isprs-archives-XLIII-B5-2020-191-2020. URL: <https://isprs-archives.copernicus.org/articles/XLIII-B5-2020/191/2020/isprs-archives-XLIII-B5-2020-191-2020.pdf>.
- [6] C. Kusumastuti et al. “A signal processing approach to correct systematic bias in trend and variability in climate model simulations”. In: *Geophysical Research Letters* 48.13 (2021), e2021GL092953.
- [7] Microsoft. *Copilot (GPT-4) [Large Language Model]*. <https://copilot.microsoft.com>. Used in July 2025. 2025.
- [8] OpenAI. *ChatGPT (GPT-4)*. <https://chat.openai.com>. Used in June 2025. 2025.
- [9] OpenStreetMap contributors. *OpenStreetMap WMS Layer*. <https://www.openstreetmap.org/#map=11/50.7599/5.9766>. Retrieved in July, 2025. 2025.
- [10] QGIS Development Team. *QGIS: A Free and Open Source Geographic Information System*. Retrieved on August 1st, 2025. 2025. URL: <https://qgis.org/>.
- [11] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 4th ed. Pearson, 2021.
- [12] Safe Software Inc. *FME: The All-Data, Any-AI Integration Platform*. Retrieved on August 1st, 2025. URL: <https://fme.safe.com/>.
- [13] V7 Labs. *Intersection over Union (IoU) – A Guide for Beginners*. Retrieved on June 2nd, 2025. 2023. URL: <https://www.v7labs.com/blog/intersection-over-union-guide>.
- [14] virtualcitysystems GmbH. *virtualcitysystems*. <https://vc.systems/>, <https://www.virtualcitymap.de>.
- [15] Wikipedia contributors. *Solaranlage*. <https://de.wikipedia.org/wiki/Solaranlage>. Retrieved on May 20th, 2025.
- [16] Qiusheng Wu. *GeoAI@GISHub*. <https://geoai.gishub.org/>. Retrieved on April 30th, 2025.

# List of Figures

2.1	Intersection over Union [13]	4
3.1	Highest IoUs per Training	8
3.2	IoUs by Epochs per Training	8
3.3	Orthophoto of Berlin 2023 (North is up; viewed in QGIS [10]; Layers from [1][9])	9
5.1	Solar Systems (red polygons) in Berlin (North is up; edited in QGIS [10]; Layers from [1][9])	11
5.2	Example: Overlapping red Polygons (North is up; viewed in QGIS [10]; Layer from [1])	12
5.3	Examples: Clipping by Building Footprints (North is up; edited in QGIS [10]; Layer from [1])	13
5.4	Spectral Library of Common Objects Misidentified as Solar Systems	14
5.5	IoU per Tile and Postprocessing Step	16
5.6	Visualized Data from MaStR (North is up; edited in QGIS [10]; Layer from [9])	17
5.7	Area Comparison: Solar Systems from GeoAI vs. MaStR (North is up; edited in QGIS [10]; Layer from [1])	17
5.8	Comparison of Number of Solar Systems per Postcode	18
5.9	Relative Differnce of Number of Solar Systems per Postcode	19
5.10	VC Map	19
A.1	IoUs by All Epochs per Training	VI
A.2	Defective Preprocessing Result (North is up; edited in QGIS [10]; Layers from [1][9])	VI
A.3	Exemplary Attribute Table	VII
A.4	Result of Filtering by Polygon Size (North is up; edited in QGIS [10])	VII
A.5	Example: red Polygons with Area of $< 1 \text{ m}^2$ (North is up; edited in QGIS [10])	VII
A.6	Histogram of Size of Polygons in $\text{m}^2$ (solar_clean_clipped.shp)	VIII
A.7	Visualization of Faulty area_sum Calculation (edited in QGIS [10])	VIII
A.8	Comparison of Number of Solar Systems per Postcode (until postcode 1250x)	VIII
A.9	Absolute Difference of Number of Solar Systems per Postcode	IX
A.10	VC Map 2	IX
A.11	VC Map 3	IX

# List of Tables

1.1	Workflow for Solar System Detection of Any City . . . . .	2
1.2	Workflow for Interdisciplinary Project . . . . .	2
2.1	Characteristics of Data Basis . . . . .	5
3.1	Training Datasets . . . . .	6
3.2	Training Runs . . . . .	7
5.1	Postprocessing Steps . . . . .	12
5.2	Statistical Parameters of Area per Polygon m <sup>2</sup> . . . . .	15
5.3	Comparison of Polygon Counts in Training Data and GeoAI Result . . . . .	15
B.1	Hardware Specifications . . . . .	XIV
B.2	Used Software Tools . . . . .	XV
B.3	IoUs per Postprocessing Step . . . . .	XV

# List of Source Codes and Workflows

C.1	Model Training for Solar System Detection (Based on Example Script [16]) . . .	XVI
C.2	Solar Panel Detection with Pre-trained Mask R-CNN Model (Based on Example Script [16]) . . . . .	XVI
C.3	Solar Panel Detection (Based on Example Script [16]) with Model 3 including Tile Cutting (Coded with Copilot [7]) . . . . .	XVII
C.4	Postprocessing with GeoAI [16] . . . . .	XVIII
C.5	Calculate IoU for Training Dataset Tiles (Coded with Copilot [7]) . . . . .	XIX
C.6	Writing *.geojson with MaStR Data (Coded with Copilot [7]) . . . . .	XX
C.7	FME Workflow: Data Preprocessing - Tile Merging and Clipping by Berlin-shape	XXI
C.8	FME Workflow: Result Postprocessing - Tile Merging and Conversion to *.shp per Batch . . . . .	XXI
C.9	FME Workflow: Result Postprocessing - Merging Overlapping Polygons . . . . .	XXI
C.10	FME Workflow: Result Postprocessing - Clipping Polygons by Building Footprint and Adding Attribute Address . . . . .	XXII
C.11	FME Workflow: General Area Calculation . . . . .	XXII
C.12	FME Workflow: Result Postprocessing - Sorting Polygons area < 1 m <sup>2</sup> and area > 1 m <sup>2</sup> . . . . .	XXII
C.13	FME Workflow: Result Analysis - Drawing Circles in Size of Total Area of Solar Systems per Building in Center of Buildings . . . . .	XXII
C.14	FME Workflow: Result Analysis - Counting Number of Solar Systems per Postcode Area for Comparison with MaStR . . . . .	XXIII
C.15	FME Workflow: Result Analysis - Clipping for IoU Calculation in Tiles of Training Data . . . . .	XXIII



## **Acknowledgement**

I would like to thank everyone who supported me in making this project happen. Special thanks go to Prof. Dr. Sukanya Bhowmik and Philipp Ungrund, who were my supervisors at the university and always gave me great advice and feedback. I would also like to thank my supervisors, Dr. Lutz Ross and Dr. Stefan Trometer, who gave me the opportunity to write my project at virtualcitysystems GmbH and always supported me with advice and assistance. I would also like to say thank you for the reliable and constructive working atmosphere. I'd also like to thank my colleagues Zehra Koç, Oliver Förster, Christian Dittmeyer, and Jannes Bolling. They helped a lot by solving problems on several occasions and were always available to answer my questions. I would like to thank my brother Philipp Kretz and my parents Barbara Litzenberger-Kretz and Mathias Kretz, as well as my friends, for their fantastic support throughout my studies. Thank you for always being there for me!



**Statement of Originality**  
(Eigenständigkeitserklärung)

**Name:** Hanna Kretz **Matriculation number:** 824063  
**Title of the Paper:** Solar System Detection with GeoAI  
**Examiner:** Prof. Dr. Sukanya Bhowmik

I hereby declare that the paper I submit follows the guidelines of good academic conduct of the University of Potsdam "Richtlinie zur Sicherung guter wissenschaftlicher Praxis für Studierende an der Universität Potsdam (Plagiatsrichtlinie)".

I confirm that the ideas I present in this paper are my own except indicated otherwise. I am aware that the following actions constitute cases of plagiarism:

- Submitting the works of others (including other students).
- Submitting work generated by artificial intelligence-supported technology.
- Submitting my own work or parts of my own work that have already been submitted for another examination.
- Copying ideas, sentences, paragraphs, or sections from other texts (including my own) without citation.
- Using images, tables, graphs, or other visuals without citation.
- Translating texts without citation.

I confirm that I have clarified any use of generative artificial intelligence (AI) with the examiner of this paper. I declare that (tick all that apply):

- ☒ I have used Copilot [7] and ChatGPT [8] for background research and independent study. I have not presented any content generated by AI as my own work.
- ☒ I have used Copilot [7] to generate Python scripts that I have adapted to include in my paper. I have documented the prompts and the respective chats in Attachment D of my paper.
- ☒ I have used DeepL [3] to significantly improve the language and style of my paper.
- ☐ I have not used any generative AI tools in conceptualizing, researching for, or writing this paper.

I am aware that if I fail to follow the guidelines and/or present misleading information, this will result in a failing grade and will be considered "Täuschungsversuch" (cheating attempt).

I am aware that my paper may be screened electronically for originality.

Date: Berlin, August 14th, 2025

Signature: H. Kretz