# Smart Home Solar Energy Management: A Machine Learning-based Recommendation and Automation Tool

Adrian Georg Herrmann
Freie Universität Berlin

6 June 2021

Master's Thesis
Supervisor: Dr. Emmanuel Baccelli

## Acknowledgements

I am greatly thankful to my employers Benoit Kaivers and Andreas Kollmann from AVM for letting me work on this thesis as part of my occupation there. Indeed, my thanks to Benoit is threefold, as he also provided data from his private PV system, and most importantly suggested this topic to me in the first place. I would also like to thank Marcel Wältermann from AVM for valuable advice.

I am also grateful to Eric van Uden from AVM greatly for providing data from his own domestic PV system, a contribution that has been absolutely essential to this thesis.

A big thanks goes out to my magnificent and brilliant brother Luis whose expertise and guidance for practical and theoretical matters of data analysis and machine learning has been invaluable as has been his proofreading and feedback; and to my very dear friends: To Windy for her proofreading and extensive feedback, to Dhruti for her proofreading and valuable linguistic advice, and to Konsti and Felix for multiple good words of advice on various fronts.

# Contents

Solar energy production is meeting increased interest, including in people's homes. Smart collection and evaluation of energy production and consumption data and weather forecasts can enable systems that maximize self-consumption of energy produced with a PV installation. This thesis proposes design and implementation of such a Home Energy Management System based on a novel battery-less approach, and presents an evaluation thereof. The system uses a machine learning energy production model to provide knowledge that enables the user to maximize the use of their installation, including recommendations for home appliance load distributions and automation.

# 1. Introduction and Motivation

Solar photovoltaics (PV) have been seeing close-to-exponential growth in the past two decades [1][2][3][4] - such a growth, in fact, that it has been consistently underestimated by agencies such as the International Energy Agency (IEA) [5][6][7]. This growth has been fueled by steadily decreasing costs, reaching a decline of roughly 75% in less than ten years as of 2014 [2], with this development similarly continuing in the years since. PV installations on residential rooftops constitute a significant portion of this market growth [8], reflecting a growing interest of homeowners and tenants in tapping solar power for environmental and economic reasons: While the average cost of residential electricity in Germany increased to 29.47 cents per Wh in 2018 and 31.37 cents per Wh in 2020 [9], the levelized cost of residential solar energy as of 2018 was estimated at around 10 to 13 cents per Wh and is projected to decrease further [10]. At the same time, the feed-in tariff as per the *Erneuerbare-Energien-Gesetz* (Renewable Energy Sources Act) has decreased to 8.16 cents per Wh as of 2021, down from 28.74 cents per Wh in 2011 [11][12]. Therefore, residential owners of PV systems have an interest in *self-consuming* as much of the energy from their PV installation as possible, instead of *exporting it into the grid*. Similarly, they may also have an interest in *minimizing energy consumption from the grid*.

Naturally, a PV installation's potential energy production at any given moment is highly variable and dependent on many factors, ranging from the sunlight's intensity (subject to e.g. the sunlight angle or possible cloud cover) to a higher air temperature impairing electrical performance. Due to this variability, the amount of produced energy can be hard to predict. Consequently, operating a PV installation without taking this into account can result in 'wasting' much of the energy that is produced by the installation if less energy is self-consumed at a given moment than is produced.

Without any further measures, this surplus energy would be exported to the grid. One possible remedy to this is the use of energy storage in the form of batteries. Batteries exist with a wide range of capacities and different levels of efficiency and longevity. They are estimated to cost around EUR 800 and EUR 1400 per kWh of capacity in Germany as of 2019 [13]. While the cost is expected to decline further, it does represent a considerable addition to the cost of acquisition and operation of a PV system. In fact,

the better a PV system's performance, the greater the potential surplus energy and the required battery size and thus its cost.

Therefore, being able to predict the energy production of a PV installation for a given moment can not only increase self-consumption, but also decrease the necessary battery size - the better this prediction, the smaller a battery is required. In an ideal case, this would allow efficient operation of a PV system under a novel approach that would enable reducing the battery size and function to only that of a load buffer, or enable even operation without a battery altogether.

While knowing the energy production of a PV installation is valuable, this knowledge on its own can still cause friction for the user when it comes to deciding how specifically to make use of that energy to maximum effect, as this requires knowledge of all appliances' energy profiles, average operation times and more. Smart home ecosystems with automation capabilities can be leveraged to make use of this knowledge and ease the burden of home appliance load scheduling off the user.

This thesis proposes a software system - a *Home Energy Management System* - that aims to achieve the aforementioned goals of *maximizing energy self-consumption*, *minimizing grid consumption* and *minimizing grid exports* for a battery-less scenario. It is designed to be modular, extensible for new functionality and use cases, and adaptable to different hardware, software and service ecosystems. Energy production is predicted with a machine learning-based model using weather data, and schedules for home appliances are suggested to the user or automated entirely.

# 2. The Problem Domain

## 2.1. Related work

The concept of an application that manages residential energy loads and appliances is not new and different forms have already been discussed in previous literature. The approach presented in this thesis borrows and adopts some terminology and constraints and concepts from related work, but differs in others.

A possible definition for an *Energy Management System* (EMS) is e.g. as "a collection of computer-aided tools used by operators of electric facilities to monitor, control, and optimize the performance of the generation and/or transmission system" [14]. Elaborating on this, Freschi and Repetto further say that "different computer aided tools are implemented from short time control modules to scheduling or commitment of power production units on a day/week basis. EMS has the objective of maximizing system performances by monitoring and control functions which require a centralized system of data collection and a decision making procedure". Originally, such a system would have been used mostly in industrial settings but the rise of residential energy production and home automation has introduced use cases to home settings as well, in which case one refers to a **Home Energy Management System** (HEM or HEMS). Shafie-Khah and Siano consider some practical difficulties of a HEMS to be e.g. uncertainty in energy consumption due to consumers' behavior and greater comfort requirements [15]. These points set a HEMS apart from an industrial-focused EMS.

### 2.1.1. Infrastructure and architecture

Zhou et al. situate a HEMS [16] in an environment that offers a *smart grid*, a term first officially coined in the *Energy Independence and Security Act of 2007* of the United States, wherein a smart grid is characterized as achieving, among others, "deployment of 'smart' technologies [...] for metering, communications concerning grid operations and status, and distribution automation", and also "deployment and integration of advanced electricity storage and peak-shaving technologies, including plug-in electric and hybrid electric vehicles" [21]. The smart grid is leveraged e.g. to gather real-time energy prices, which are employed towards the main goal of *minimizing the cost of energy*, e.g. through pricing strategy-based energy scheduling strategies.

Zhou et al. identify the components of a HEMS' in-home infrastructure as follows:

1. **Communication and networking**, referring to the variety of such technologies employed throughout the environment, including power line communication, ZigBee, BACnet, Bluetooth and human-machine interface systems.
2. **Smart meters**, regarded mainly as the interface to the smart grid, e.g. gathering real-time energy prices.
3. **Smart HEMS center**, the hub whose main functions include [17]:
   a) A *control panel* with a real-time display,
   b) A *human-machine interface*, supporting browsing, monitoring and *task setting*,
   c) Integration of energy storage devices, *analyzing* local energy production,
   d) Optionally setting water, gas and other controls.
4. **Home appliances**, schedulable and non-schedulable, where schedulable can be interruptible and non-interruptible.

Based on this infrastructure, Zhou et al. present an architecture for a HEMS by identifying a number of modules:

1. **Monitoring** of the current state,
2. **Logging** of past states,
3. Direct and remote **control**,
4. **Management** of the renewable energy system, energy storage, home appliances (through control schemes to minimize their energy consumption), plug-in electric vehicle (proposed as energy buffer) and battery.
5. **Alarm** to handle faults and abnormalities.

Systems that attempt to *reduce grid exports and grid consumption* by shifting and managing loads are referred to by O'Shaughnessy et al. [19] as "Solar Plus", in opposition to "standalone solar" approaches that may produce excess energy. Luthander et al. review research into such Solar Plus systems and identify two dimensions over which solutions are sought, namely *energy storage* and *load management*. They find that a battery size of 0.5–1 kWh per installed kW of PV power can increase energy self-consumption by 13-24% whereas load management can achieve increases of 2-15%.

### 2.1.2. Load management

Ogunjuyigbe et al. propose [18] an online ad hoc load management scheme for home appliances in a residential PV installation scenario. Appliances are classified based on "user comfort" and "quality of life" into load levels, beginning with "uncontrollable" loads (demanding instantaneous power, e.g. a computer or TV or a toaster) and continuing with "controllable" (deferrable) loads (e.g. a washing machine or water heater). The load management scheme is then formulated as a mixed-integer linear program (MILP) based on these load levels, the battery's rating and size, the available power output of the PV installation and other constraints. The available power output of the PV system for a given period is dependent on previous load demand and the amount of power derivable from solar irradiation at that moment. Battery and PV size are not assumed as given but are instead determined based on the user's load demand such that they are able to meet it using the load management scheme. The MILP is solved hourly using the incoming data from that hourly period.

## 2.2. Contributions of this thesis

While previous work centers on finding solutions over the dimensions of energy storage and load management, this thesis proposes a different way. We consider scenarios where energy storage is assumed to be non-existent or of negligible size, so the dimension of energy storage becomes irrelevant in the exploration of solutions. Instead, it is proposed that in addition to the load management dimension, another should be examined: *predicting energy production*. It is further proposed as a consequence of this to examine *offline, a priori load management*, as an estimate of future energy production allows managing loads based on estimated future energy budgets, instead of the current one.

Based on this new approach, this thesis further contributes the following:

1. A new modular system architecture for a HEMS application, situated in Zhou et al.'s infrastructure as an interpretation of the *smart HEMS center*, that employs this approach. This is discussed in section 3, which presents the design of the application and provides a specification, including data types, database scheme, modules, interfaces and program sequences.

2. Two datasets of energy production from example locations in Berlin, Germany and Wijchen, Netherlands. Section 4.1 characterizes and analyzes this data and

addresses the prototype hardware deployment for its collection. Section 4.2 characterizes and analyzes weather data collected at stations near the two example locations with respect to how it correlates with the energy production data.

3. An exploration of machine learning-based models to predict energy production, proposed and evaluated in section 5.

4. An exploration of heuristic offline, a priori load management schemes to automate or suggest tasks to the user such that the stated goals of *maximizing self-consumption*, *minimizing grid consumption* and *minimizing grid exports* are achieved, discussed in section 6.

5. A partial implementation with a basic web user interface, allowing the user to request predictions of energy production and suggestions for appliance load scheduling. This is discussed in section 7.

Considering these contributions, the thesis will attempt to answer the following question: Can it be said **if a battery-less HEMS appears feasible or promising**, particularly in the geographical regions for which data was collected?

All code assets produced in the context of this thesis and the energy production datasets are available under `https://github.com/adrianghc/HEMS`.

# 3. Design of the HEMS

## 3.1. Scenario definition

The HEMS application (henceforth referred to simply as HEMS) operates in an infrastructure that consists of a home environment and one or several external data sources for weather. The home environment holds an actor (one or several **users**), any number of **appliances** (i.e. **energy consumers**) and one PV installation (i.e. the **energy producer**). External data sources provide information to the HEMS that is required for its operation but cannot be obtained from any of the home environment's members. The sum of the HEMS itself, the user, the appliances, the PV installation and the external data sources constitutes the infrastructure.

This infrastructure includes a PV system, but it does **not** presuppose a **battery**. A small battery can be present and act as a load buffer, in which case it is situated in the infrastructure as an extension of the PV system. The HEMS is conceptualized such that its operation is not reliant on a battery.

There are no limitations to the possible PV hardware, neither in size nor in performance nor the specific PV technology employed. However, the installation must offer a software interface (e.g. network-based) to enable gathering of measurement values. This interface can be part of the PV system's hardware itself or provided by additional hardware like a smart outlet. Measurements should be available in a fixed resolution, for which possible values are a 15-minute, 30-minute or hourly cadence.

Appliances can be **schedulable** or **non-schedulable**, and the former can offer some form of binary **automation** (i.e. a device is either on or off) or lack it. Automation

can be provided directly by the appliance or through separate hardware like a smart outlet, but in each case the automation must be available to the HEMS through some software interface. All appliances must be configured in the HEMS with their respective *power rating*, and schedulable devices must also be configured with their *average duty cycle* and how often they can be scheduled within a week. The latter value can change if an appliance requires more or less frequent operation during some time period, e.g. because of seasonal variability. All types of appliances can offer a software interface to enable gathering of energy consumption values, which can then be used to continually update the preset power ratings, but this is not mandatory. Therefore, the collection of energy consumption measurements is optional for the HEMS' operation.

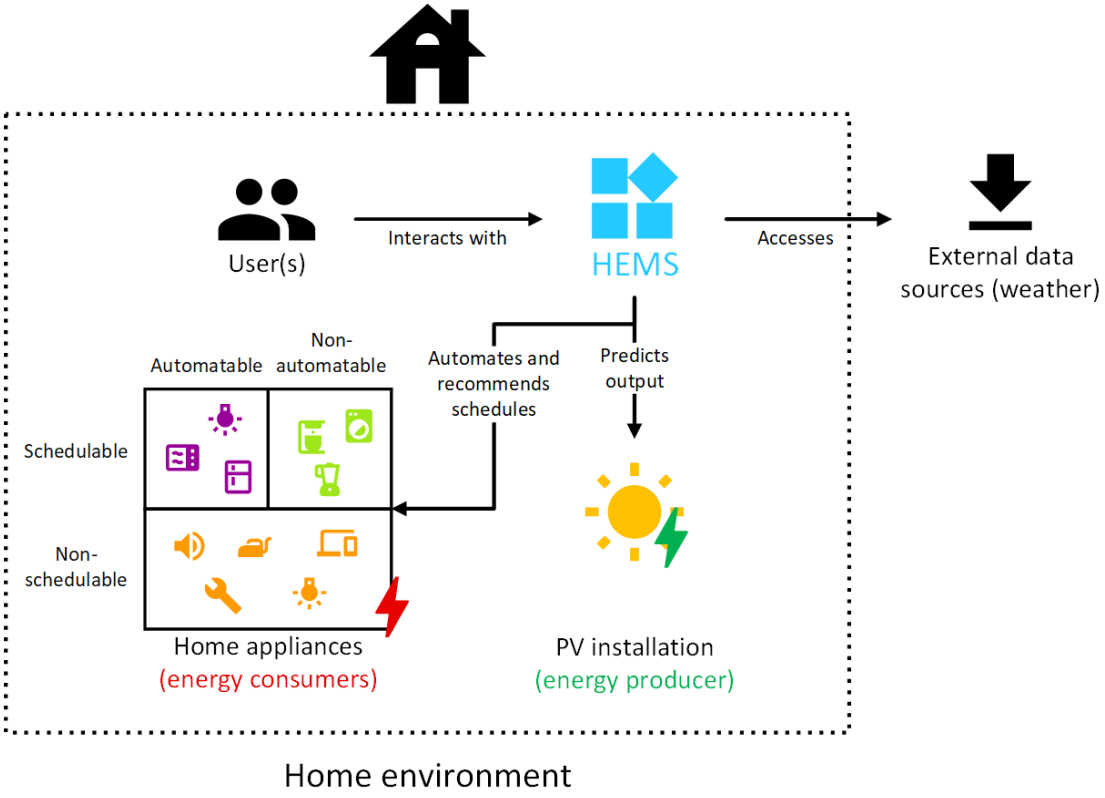



Figure 1: Infrastructure and environment of the HEMS

Some examples for schedulable appliances are:

- Ventilation systems and ceiling fans,
- Indoor, passageway or security lights,
- Radiators and cartridge heaters,
- Ovens (schedulable for e.g. pre-heating),

- Refrigerators,
- Washing machines, drying machines and dishwashers,
- Water pumps, water heaters and blenders,
- Manual or automatic lawnmowers (or the charging thereof if they have a battery),
- Manual or automatic vacuum cleaners (or the charging thereof if they have a battery),
- Electric cars (i.e. the charging thereof),
- Other portable devices that can be charged.

The HEMS can either **recommend** or **automate** schedules for appliances based on the energy budget such that energy self-consumption is maximized and grid exports are minimized, or grid consumption is minimized. In principle, the HEMS could be operated in an environment where no single appliance can be automated, in which case the HEMS would only be able to offer recommendations and no automations. However, while non-schedulable appliances are permitted, there must also be schedulable ones, as otherwise there is no schedule to suggest. Non-schedulable appliances are still useful knowledge for the HEMS as they add to baseline energy consumption and thus subtract from the predicted energy budget. They can be appliances that run constantly or on a fixed, preset schedule, which is manually added by the user and can be modified at any time.

The user can request recommendations anytime. Recommendations can be discarded or committed by the user, in which case they become scheduled **tasks**. A task represents a single timespan during which one or several appliances are in operation. More complex scenarios involving repetition and recurrence can be realized through **automation profiles**. Automation profiles must be added manually by the user, while tasks can be added manually, through user commit upon recommendation, or automatically, in which case the HEMS skips the recommendation and decides on its own which tasks to schedule. The schedule on which the HEMS automatically creates tasks can be configured by the user. Manually added tasks and their lengths can be used by the HEMS to continually update appliances' average duty cycles.

When creating recommendations by request or when automating appliances, the HEMS first **predicts the energy production** for some period of time. Energy production is predicted on the basis of **weather forecasts** from the user's location, which is provided by the user. The source of this weather data is an external data source that is the only part of the infrastructure outside the home environment, and therefore also the only part of the infrastructure that requires an internet connection for communication.

Since weather forecasts lose precision the further out they are, so does the prediction for the energy production, which means that the period of time for the energy predictions must strike a balance between maintaining sufficient accuracy and enabling the user to plan well ahead. A period of one week achieves that: According to the US National Oceanic and Atmospheric Administration (NOAA), while the accuracy of ten-day

forecasts plummets to about 50%, seven-day forecasts have on average an accuracy of about 80% [23]. In addition, a week is a very natural time period for planning purposes.

The user sets their **geographical location** upon first run of the HEMS. It can be changed at any time, in which case all knowledge tied to that geographical location, i.e. weather and energy production, is reset.

The energy production model is dynamic and depends on the environment, including in particular the user's geographical location and PV system performance. Therefore, it is specific to each user and must be trained for each user individually. To maintain accuracy of the energy production model, it must be continuously re-trained. This is done automatically on a schedule without user intervention.

There is no limitation as to what type of hardware the HEMS can run on as long as it has access (network-based or otherwise) to all relevant parts of the infrastructure, i.e. appliances, PV system and weather sources, and as long as the user can interact with it. To this end, it must offer a **user interface**, e.g. a web interface or an application with a frontend. Additionally, the hardware must satisfy the HEMS' system requirements such as sufficient memory and storage. Some functionality of the HEMS such as training the energy production model may have greater hardware requirements that may be at odds with other constraints (see 3.5.5). To account for this, parts of the HEMS' functionality can be offloaded to more powerful devices instead of its entirety being hosted on one single device.

## 3.2. Architecture

The aforementioned functionality offered by the HEMS can be split into several cohesive modules with low coupling, making it easier to extend, replace and add functionality and to maintain existing one. Some of the modules are concerned with 'outward' functionality such as interaction between the HEMS and the infrastructure, i.e. interaction with the user, the home environment or external data sources. Others concern themselves with 'inward' functionality, which includes getting and setting internal states and tracking information. The modules are:

- A module that offers a way for the user to interact with the HEMS. This module is called **User Interface Module** and is discussed in 3.5.2. Interactions with the HEMS can range from commands for action (e.g. a command to make new recommendations), commands for data manipulation (e.g. a command to add a new appliance or change an existing one) or viewing available data (e.g. list all scheduled tasks).

- A module that collects new data measurements from the infrastructure. This is the **Measurement Collection Module** and is discussed in 3.2. Data measurements include energy production data from the PV system, energy consumption data from the appliances, and weather data from external sources.

- A module that generates recommendations on user request and/or automates appliances, if possible and configured. This module is called **Automation and Recommendation Module** and is discussed in 3.5.4.

- A module that trains the environment-specific energy production model, the **Model Training Module**, discussed in 3.5.5.
- A module that infers knowledge from the energy production model, i.e. predictions of energy production for some time period. This is called the **Knowledge Inference Module** and is discussed in 3.5.6.
- A module that stores all persistent data such as measurements to add, read, modify or delete it on other modules' request. This is the **Data Storage Module**, discussed in 3.5.7.

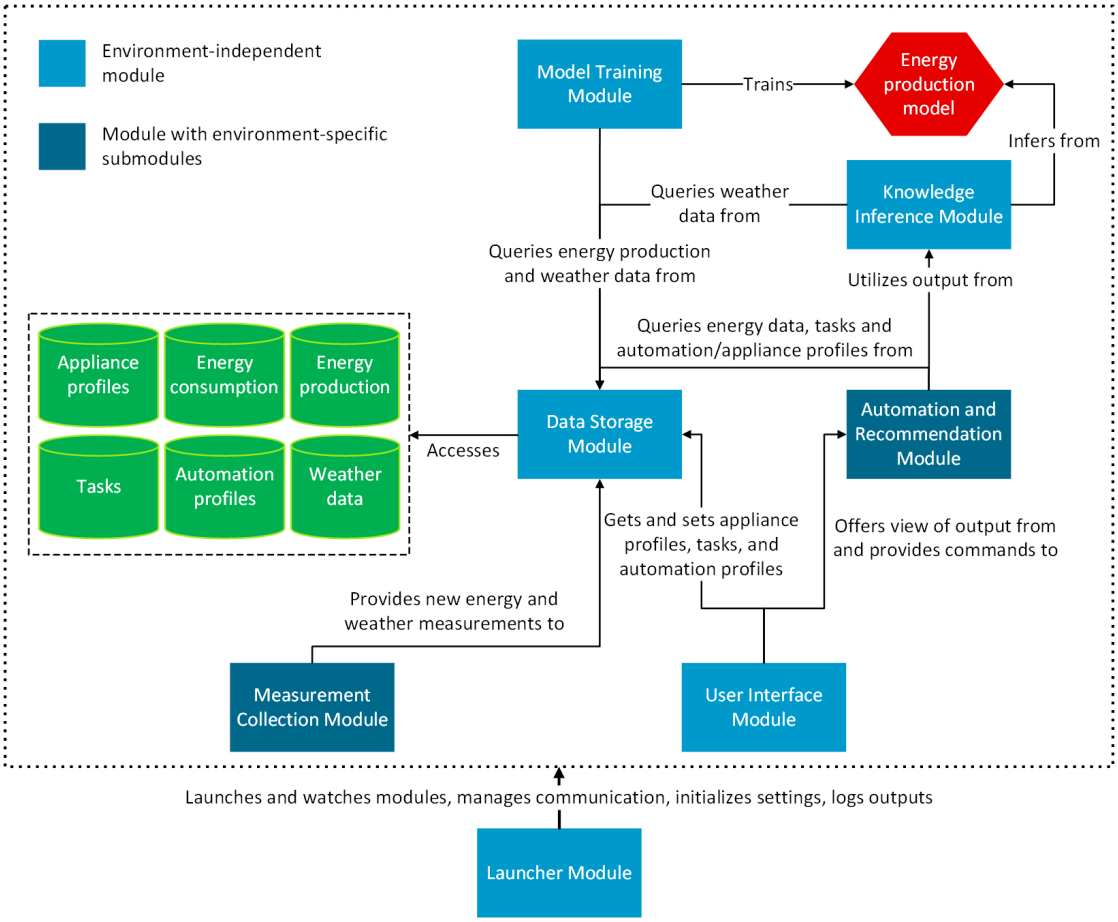



Figure 2: Architecture of the HEMS

In addition to all these, there is one special module:

- The **Launcher Module**, which is not concerned with outward or inward functionality of the HEMS but with managing and starting modules, managing their communication, initializing settings and offering a programmatic entry point to the HEMS application. 3.5.1 discusses this module.

The modules must communicate and interact with each other to achieve all the HEMS' functionality as a collective. Modules communicate with each other by exchanging *messages* for the purpose of requesting another module to do something (*command messages*) or to return something (*request messages*), in which case a *response message* is sent back. The messages that each module can accept and the responses that it returns constitute the module's interface to the others.

## 3.3. Data types

The HEMS is a data-centric application: Data of various types is collected, stored and read in order to generate knowledge and results. Understanding this data and defining appropriate data types is crucial to ensure that all modules can handle and interpret data consistently and according to their respective requirements.

### 3.3.1. Appliances

| Field name | Type (default) | Description |
|---|---|---|
| **id** | unsigned int | The appliance's id. Must be greater than 0. If it is 0, it indicates a new entry to be stored. |
| name | string | A string containing the appliance's name. |
| uri | string | A string containing an URI to automate an appliance and/or read energy consumption data off it. |
| power | float (0) | The power rating of the appliance. |
| duty_cycle | unsigned int (0) | The average duty cycle of the appliance in hours. |
| schedules_per_week | unsigned int (0) | How often an appliance should be scheduled in the time frame of a week. A value of 0 means that the appliance is not schedulable at all. This describes appliances that run without input from the HEMS but add to the total energy consumption. |
| tasks | [unsigned int] | A set of ids for tasks attached to this appliance. |
| auto_profiles | [unsigned int] | A set of ids for automation profiles attached to this appliance. |

Table 1: The *appliance* type (**id** is the unique identifier)

The HEMS needs to work with **appliances** (see table 1). For each appliance there exists an **appliance profile** that compiles name, power rating and average duty cycle

(in minutes) of an appliance and has a variable number of automation profiles and tasks attached to it. Additionally, a value states how often an appliance should be scheduled in the time frame of a week. This can also be used to say whether the appliance is schedulable or not, as the latter would be the case if the appliance should be scheduled 'zero times' in the time frame of a week.

Each appliance can have a unique resource identifier (URI) that the HEMS uses to read energy consumption data off it, and to automate it if applicable. The interpretation and format of this URI is entirely up to the implementation and the infrastructure.

The power rating is set initially when creating a new appliance, but it may be updated as new energy consumption measurements come in. Likewise, the average duty cycle is set initially but may be updated based on the length of user-declared tasks (see *tasks* in 3.3.2). Non-schedulable devices have no average duty cycle and any value for the latter would be ignored.

### 3.3.2. Tasks and automation profiles

| Field name | Type (default) | Description |
|---|---|---|
| **id** | unsigned int | The tasks's id. Must be greater than 0. If it is 0, it indicates a new entry to be stored. |
| name | string | A string containing the task's name. |
| start_time | timestamp | The time when the task's execution starts. |
| end_time | timestamp | The time when the task's execution ends. |
| auto_profile | unsigned int (0) | The id of the automation profile this task belongs to, or 0 if none. |
| is_user_declared | bool (false) | This states whether the task has been explicitly declared or committed by the user. These tasks are respected by the HEMS, whereas automatically generated tasks can be replaced by the Automation and Recommendation Module in the face of new predictions. Additionally, user-declared tasks can affect the average duty cycle of an appliance. |
| appliances | [unsigned int] | A set of ids for appliances attached to this task. |

Table 2: The *task* type (**id** is the unique identifier)

There are **tasks** and **automation profiles**. A task defines a specific, scheduled instance of one or several appliances being turned on for some time period. Automation profiles are similar in that they define the same as tasks, but in a recurring fashion. One approach for handling this might be defining some fields that can assign some recurrence to a task. However, there are several reasons why this is not the best approach:

- The more complex a recurrence becomes, the more fields would be needed. A recurrence like 'the same day every week' might be simple enough. Something like 'every Monday, Tuesday and Friday every week from September to November, except the first week of October' is much more complicated.
- When wanting an overview of all tasks that are scheduled at some point, the system would need to go through all task entries and procedurally generate all tasks that follow from the recurrence.

The solution is to handle tasks and automation profiles differently and separately. Tasks (table 2) have a name and specific start and end time and can optionally belong to an automation profile. A boolean value states whether the task has been explicitly declared or confirmed/committed by the user. These tasks are respected by the HEMS, whereas automatically generated tasks can be replaced if deemed sensible by the Automation and Recommendation Module in the face of updated weather forecasts, and thus new predictions, depending on the mode of operation. Additionally, user-declared tasks affect the average duty cycle of an appliance.

| Field name | Type | Description |
|---|---|---|
| **id** | unsigned int | The automation profile's id. Must be greater than 0. If it is 0, it indicates a new entry to be stored. |
| name | string | A string containing the automation profile's name. |
| profile | string | A string containing the automation profile in iCalendar syntax. |
| appliances | [unsigned int] | A set of ids for appliances attached to this automation profile. |
| tasks | [unsigned int] | A set of task ids spawned from this automation profile. |

Table 3: The *automation profile* type (**id** is the unique identifier)

Automation profiles and tasks are in a 0,1:n relation. That is, each automation profile 'spawns' any number of tasks, and each task can have at most one automation profile as its originator. Whenever an automation profile is added, a number of tasks that follow from this automation profile are added to the tasks table. Whenever an automation profile is deleted, all tasks that followed from that automation profile are deleted as well. When an existing automation profile is updated, all tasks that originally followed from that profile are deleted and new tasks are added, just as they would when adding a new automation profile. This mechanism also allows single tasks to diverge from their definition in the automation profile by modifying them while still keeping them linked to the original automation profile. Similarly, single tasks can be deleted from a series.

The question remains what may be the best way to describe an automation profile. One could try to define an increasing number of fields to accommodate ever more complex recurrence scenarios - or one could solve this the same way a pre-existing and well-established application does: Like calendar events, with iCalendar syntax. The

field with iCalendar syntax would be interpreted by the Automation and Recommendation Module when adding, modifying or deleting automation profiles, and by the User Interface Module when letting the user view, add, edit or delete them.

### 3.3.3. Energy consumption and energy production data

The HEMS needs to keep track of **energy** measurements, both **production** and optionally **consumption**. This needs to be done with a sensible and consistent resolution - e.g. every fifteen minutes. If the resolution is consistent, then only the end of the 15-minute period needs to be stored for each entry.

Energy consumption data (see table 4) should be stored per appliance. There may be energy consumption that comes from unspecified non-schedulable appliances for which no *appliance* entries exist. This baseline energy consumption can be described with the energy consumption data type by forgoing a value for the appliance id.

Energy production data is simple and straightforward, as there is one single source, see table 5.

| Field name | Type (default) | Description |
|---|---|---|
| **time** | timestamp | The end of the 15-minute interval. |
| **appliance_id** | unsigned int (0) | The id of the appliance that has consumed energy, or 0 if this is consumption data from appliances for which no individual measurements exist. |
| energy | float | The amount of energy consumed. |

Table 4: The *energy consumption* type (**time, appliance_id** are the compound unique identifier)

| Field name | Type | Description |
|---|---|---|
| **time** | timestamp | The end of the 15-minute interval. |
| energy | float | The amount of energy produced. |

Table 5: The *energy production* type (**time** is the unique identifier)

### 3.3.4. Weather and sunlight data

Weather data (table 6) is used to predict energy production of the PV system. It can be historical measurements or future forecasts. When training the model, historical measurements are used, while inference uses at least forecasts (see section 5). Weather data is downloaded continuously (see section 3.5.3) and regularly during the HEMS' operation. In this process, weather data for future time points is continuously replaced with newer, more accurate forecasts until it is eventually replaced with the measurements that were actually observed.

The HEMS can use data from more than one weather station near the user's location. Each weather station has a unique id that is stored in the special *settings* type (see 3.3.5). Weather data that is collected includes global radiation, cloud cover, temperature, humidity and pressure (see 4.2).

| Field name | Type (constraints) | Description |
|---|---|---|
| **time** | timestamp | The point in time. |
| **station** | unsigned int | The weather station's id. |
| radiation | float ($\geq 0$) | The global radiation in kJ per square meter.c |
| cloud_cover | unsigned int (0-100) | The cloud cover in percent. |
| temperature | float | The temperature in °C. |
| humidity | unsigned int (0-100) | The humidity in percent. |
| pressure | float | The air pressure in hPa. |

Table 6: The *weather* type (**time, station** are the compound unique identifier)

Sunlight angles (table 7) are used analogously to the weather data to train the model and to infer from it. However, sunlight data differs from weather data in that it does not need to be downloaded but can be calculated on demand for the user's geographical location.[1] For the same reason, it represents a special case in that it does not need to be stored. Negative sunlight angles (i.e. nightly angles) are flattened out to 0.

| Field name | Type (constraints) | Description |
|---|---|---|
| **time** | timestamp | The time. |
| angle | float ($\geq 0$) | The sunlight angle in degrees at that time. |

Table 7: The *sunlight* type (**time** is the unique identifier)

### 3.3.5. Settings

The settings are a special type that compile a common state that is shared and *synchronized* across modules through an implementation-specific synchronization mechanism that ensures that modules react to settings changes initiated by other modules as they see fit, and that they can accept or reject new settings. An example for this settings mechanism is described for the partial implementation discussed in section 7.1.

There is only one unique instance of the settings type, which is another difference to the other types that have been discussed. Settings include at least the user's geographical location, timezone and information concerning the weather stations, like their ids and the intervals at which they release new data. Intervals can also be included for retrieval

[1] One possible algorithm to calculate can be found e.g. on `https://de.wikipedia.org/wiki/Sonnenstand#Genauere_Ermittlung_des_Sonnenstandes_f%C3%BCr_einen_Zeitpunkt`

of new energy production or consumption data, or for the automation of tasks if it is enabled. It is also sensible to include URIs for data collection from weather stations and the PV system.

Different HEMS implementations can have more settings fields, and not all fields must be used by all modules. In general, the settings type is useful for parameters that must be synchronized between modules and must be up-to-date between them at all times.

| Field name | Type (default) | Description |
|---|---|---|
| longitude | float (0) | The user's longitude. |
| latitude | float (0) | The user's latitude. |
| timezone | float (0) | The user's timezone. |
| pv_uri | string | A string containing an URI to read energy production data from the PV system. |
| station_intervals | unsigned int → unsigned int (0-60) | Each weather station's interval in minutes at which they release new data. |
| interval_ energy_production | unsigned int (15) | The interval in minutes to collect new energy production data. Must be between 1 and 60 and a divisor of 60. |
| interval_ energy_consumption | unsigned int (15) | The interval in minutes to collect new energy consumption data. Must be between 1 and 60 and a divisor of 60. |
| interval_automation | unsigned int (0) | The interval in hours to automate tasks. A value of 0 means that no automation takes place and the HEMS relies on user commits entirely. |

Table 8: A possible *settings* type

## 3.4. Database scheme

The database scheme follows directly and for the most part straightforward from the data types except where there are m:n relations, as is the case for appliances, tasks and automation profiles: An appliance can be part of several automation profiles and tasks, and an automation profile or task can have more than one appliance assigned to it. This has to be modeled with additional tables. A database scheme excluding the settings is shown in figure 3 and a database scheme for the settings type suggested in 3.3.5 is shown in figure 4, both using SQLite data types.

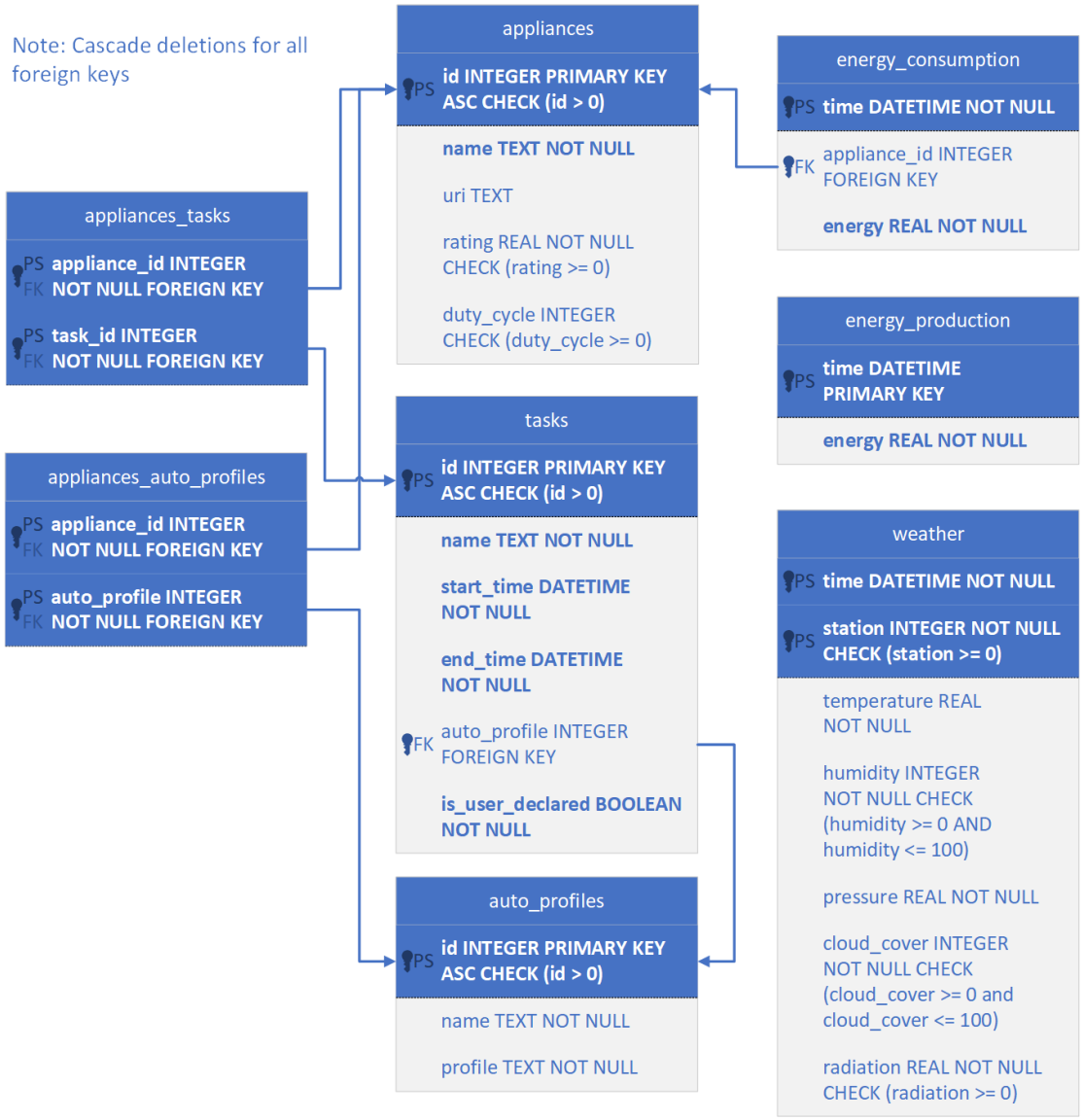



Figure 3: Database scheme of the HEMS (without settings)

## 3.5. Module functionalities and interfaces

One result of the modularization is that a large part of each module's functionality consists of handling incoming communication from other modules and responding to it. Therefore, each module's own functionality and requirements can in turn influence those of other modules. In the following, each module's functionality and message-based interface is discussed, plus some specific constraints and limitations.

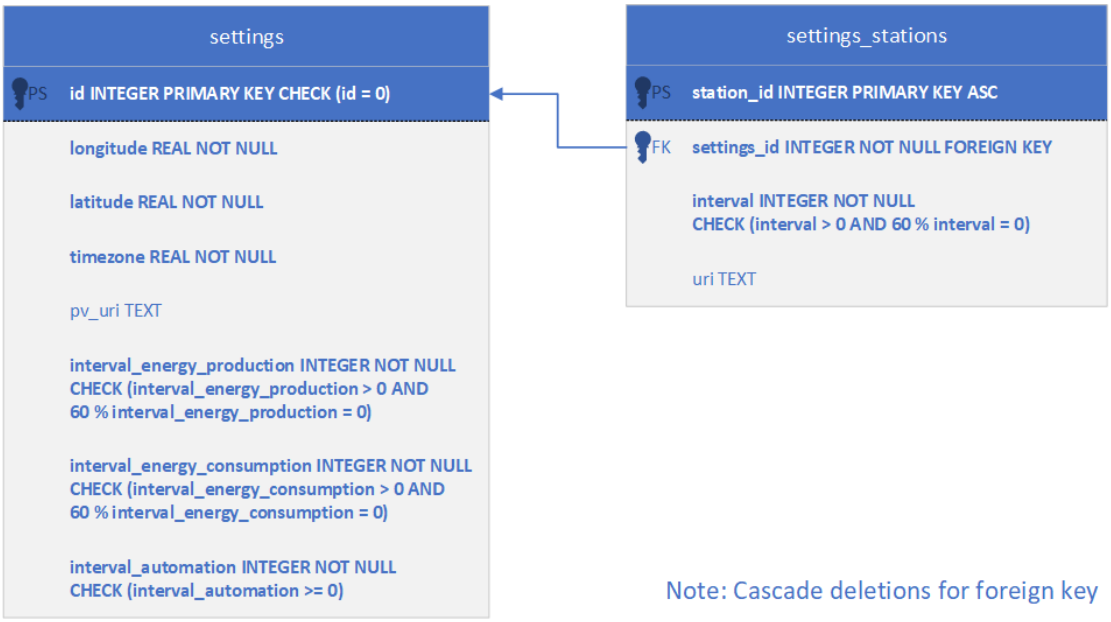



Figure 4: Database scheme of the HEMS for the suggested settings type

### 3.5.1. The Launcher Module

The **Launcher Module** is the entry point to the entire application. It handles arguments and forwards them to each module accordingly when starting them. Prior to starting the modules, it needs to establish their common communication mechanism (e.g. message queues) and initialize all HEMS settings for the current session (see 3.3).

The HEMS logs all outputs generated by other modules. It prints them out to the console and optionally to a text file for later review by the user.

During the HEMS' runtime, the Launcher Module watches the state of all other modules. It catches when a module terminates orderly or with an error, in which case the error is printed. The other modules and the communication mechanism are then shut down orderly (the meaning of which is up to each module) and the HEMS terminates. Appliances are left to continue running until the user turns them off manually.

The Launcher Module only needs to accept one type of message:

- **MSG_LOG**: This **command** message is used to send a log message (with a given log level) to the central logger in the HEMS Launcher.

### 3.5.2. The User Interface Module

The **User Interface Module** is the one that offers a way for the user to interact with the system. Therefore, all functionality that requires user input begins with the User Interface Module, where the user either asks the HEMS to do something or to display information.

The module offers the user a view of data that the HEMS has stored, including:

- **Appliance profiles**: The user needs to see which appliances are registered and their profiles.
- **Tasks and automation profiles**: The user needs to see which tasks are running and which tasks are scheduled (likewise automation profiles). If the HEMS is configured to automatically create tasks without user input, this may also allow the user to review and evaluate the HEMS' performance.
- **Energy production and consumption data**: Seeing an overview of how much energy their PV system produces and of total or appliance-specific energy consumption allows the user to diagnose problems and inform decisions and purchases.
- **Settings**: The user may want to review current settings, e.g. to identify a wrong geographical location.

Entering new data is another essential feature that the HEMS must enable through the user interface. This includes:

- **Appliance profiles**: The user needs to add new appliances or modify or delete existing ones to reflect changes to their home environment.
- **Tasks and automation profiles**: Tasks can (and automation profiles must) be added manually by the user. They must also be able to modify existing tasks and automation profiles or delete them, e.g. if they change their mind about operating an appliance at a given time. The *Automation and Recommendation Module* must be made aware of this.
- **Energy consumption data**: Some energy may be consumed by appliances that are not tracked by the HEMS but still add to baseline consumption (see 3.3.3). Since this energy consumption is not tracked by the HEMS, it may be added, modified or deleted manually by the user. Appliance-specific energy consumption data may also be entered by the user for non-schedulable appliances, but this should not be done for others.
- **Settings**: The settings of the HEMS must be modifiable through the interface.

Energy production and weather data should not be entered by the user to maintain and ensure their correctness.

The user must be able to request new recommendations. Unlike the previous functions, this does not ask for a view of existing data but instead it requests new action from the Automation and Recommendation Module. The User Interface Module requests new recommendations from the Automation and Recommendation Module (see 3.5.4), a view of which is then offered to the user via the user interface. The user can then confirm a recommendation or a subset thereof (i.e. a subset of tasks from the recommendation), or ignore or deny a recommendation entirely. This is notified back to the Automation and Recommendation Module.

Since the User Interface Module only generates new data at the user's request, there is nothing other modules can ask it to do. Therefore, there are no messages from other modules it needs to handle except for responses. However, a specific implementation may include messages that notify the User Interface Module of certain changes to data in order to enable a dynamic presentation thereof to the user. Other functionality may be added as seen fit by specific implementations depending on user requirements.

### 3.5.3. The Measurement Collection Module

The *Measurement Collection Module* is in charge of collecting new energy production and consumption measurements and weather data from their sources, and subsequently send this data to the Data Storage Module for persistence (see 3.5.7). All sources and their respective interfaces are specific to the infrastructure the HEMS operates in and therefore, this module cannot be generalized entirely and some parts need to be implemented specifically for each case.

Measurements are downloaded on a schedule specific to the type (see 3.3.5) and the respective source. Energy measurements are added anew, but weather data for future time points is continuously replaced with newer, more accurate forecasts until it is eventually replaced with measurements.

Data downloads may also be initiated manually by the user. To this end, the Measurement Collection Module should handle respective messages that are sent by the User Interface Module:

- **MSG_DOWNLOAD_ENERGY_PRODUCTION**: Collect energy production data for the fixed cadence ending at the given time.
- **MSG_DOWNLOAD_ENERGY_CONSUMPTION**: Collect energy consumption data for the fixed cadence ending at the given time, and for the given appliance.
- **MSG_DOWNLOAD_WEATHER_DATA**: Collect weather data for the given time and weather station.

The module also needs to handle new settings when they include geographical information, as that is needed for the weather collection, or when they include changes to the weather stations or any data collection intervals.

### 3.5.4. The Automation and Recommendation Module

The **Automation and Recommendation Module** uses predictions of energy production in order to schedule appliances such that energy self-consumption is maximized and grid exports and grid consumption are minimized. Algorithms that can be used to achieve this are explored in 6. The result is a series of tasks that are either added automatically or presented to the user to choose from. Tasks that are chosen through either means have to be run on schedule by the automation module. To this end, it must be aware when tasks and automation profiles are added or modified manually, i.e. by the

User Interface Module through user interaction. Two message types can be introduced for this:

- **MSG_SET_TASKS**: Notify the Automation and Recommendation Module of new or modified tasks.
- **MSG_SET_AUTOMATION**: Notify the Automation and Recommendation Module of new or modified automation profiles.

Likewise, two message types can be introduced to notify the module of manual deletion of tasks and automation profiles through the user interface:

- **MSG_DEL_TASKS**: Notify the Automation and Recommendation Module of task deletions.
- **MSG_DEL_AUTOMATION**: Notify the Automation and Recommendation Module of automation profile deletions.

Automations and recommendations differ in that the former is done on a schedule while the latter is done when initiated by the user. This must be done with another message that the User Interface Module can send:

- **MSG_GET_RECOMMENDATIONS**: Request recommended tasks for the week beginning at a given time. The response is a list of tasks.

This schedule for automations, i.e. the schedule for when the HEMS automatically generates and commits tasks, must be no longer than one week, as that is the time frame for energy predictions and also the time frame for which tasks are recommended. On the one hand, the schedule for automation should not be too long: Consider a scenario where new tasks are automatically generated once a week on the basis of energy production predictions. Then the predictions for day seven would always be more inaccurate than the others (because of the inaccuracy of weather forecasts) and never become more accurate, as by the time new predictions are made, the tasks that were scheduled for day seven have already run. This would mean that tasks scheduled for every seventh day would always be based on worse predictions. Instead, the automation schedule should be shorter so that for days that are further out, the predictions would be updated and become more accurate before the tasks for that day run. On the other hand, if the automation schedule is too short, predictions may change too frequently and result in too much planning uncertainty for the user.

Automations and recommendations are based on each appliance's average duty cycle and power rating. While values for these are given initially (see 3.3.1) they can also be updated over time: Power ratings can be updated as a rolling average using new energy consumption measurements. Likewise, the average duty cycle can be a rolling average that is updated with the length of new user-defined tasks. Since average duty cycle and power ratings are used only by the automation module, it is sensible to let it update these values whenever automation runs on schedule or when recommendations are triggered.

### 3.5.5. The Model Training Module

The **Model Training Module** trains the model that is used to predict energy production, discussed in section 5. The model is trained with historical weather and energy production measurements, which the module retrieves from the Data Storage Module. In order to maintain and improve its accuracy over time and under changing conditions, it needs to be retrained regularly. While this should be done on a schedule, it may be sensible for the user to be able to manually request retraining through the user interface if they feel that accuracy has worsened or is insufficient. A simple message type can be introduced for this purpose:

- **MSG_TRAIN**: Request immediate retraining of the model.

The training of models can have high system requirements regarding CPU power and memory. Storage footprint may also be relevant, as the model is dynamic and can therefore not be pre-trained and compressed - a use case that is explicitly considered by machine learning frameworks such as Python-based TensorFlow[2] and PyTorch[3], which offer special solutions tailored to constrained mobile devices in the form of TensorFlow Lite[4] and PyTorch Mobile[5]. These light frameworks require pre-training, so they cannot be used for the HEMS. However, the full machine learning frameworks require a significant amount of storage: An experimental build of TensorFlow for an ARM platform that was made in the context of this thesis resulted in two dynamic libraries with sizes of around 149 MB and 25 MB.[6]

The HEMS application may be deployed by the user on hardware that does not meet the requirements of the machine learning framework and the model. Therefore, parts of the Model Training Module plus the model itself may be offloaded to another more powerful device. Depending on deployment or product constraints the module and the model may be hosted on cloud computing capacities (e.g. rented by the vendor of an HEMS-based product) or on an on-premises server with a GPU (e.g. for a single residential deployment).

### 3.5.6. The Knowledge Inference Module

The **Knowledge Inference Module** uses the model to infer predictions of energy production. Model inference requires at least weather data for the week-long time frame for which predictions are desired, plus potentially historical data depending on the specific model. Predictions are required for automation and for recommendations, so they must be triggered by a request from the Automation and Recommendation Module. Unlike for model training, no schedule is required for model inference.

Inference can be triggered by the following message type:

[2] `https://www.tensorflow.org/`
[3] `https://pytorch.org/`
[4] `https://www.tensorflow.org/lite`
[5] `https://pytorch.org/mobile/home/`
[6] The build target was a Qualcomm 'Dakota' SoC (717MHz Quad Core ARM Cortex A7 WiSoC + 2x 2x2 MAC/Baseband/Radio).

- **MSG_GET_PREDICTIONS**: Request a prediction for energy production for one week beginning at the given time. The response maps times to their respective predicted energy production. The times follow a fixed interval as discussed in 3.1, e.g. 15 minutes.

While model inference has lower system requirements than training, the system requirements of the HEMS are constrained by the latter. Therefore, if training and the model itself are offloaded to a different device than the HEMS application, this would apply to inference as well.

### 3.5.7. The Data Storage Module

The **Data Storage Module** is responsible for managing access to data storage for all other modules. Whenever other modules need to read or write measurements or other data, they can issue messages to the Data Storage Module. Centralized storage access makes it easier to manage access from multiple modules. It also allows abstracting away the database scheme for more complex data types so that other modules need not concern themselves with this.

The Data Storage Module stores all measurements, including weather data, energy production data and energy consumption data. It also stores the HEMS' settings across sessions and recovers them at the start of each session.

One message type for this module is straightforward: Other modules may generate new data and send it to the Data Storage Module for persistence. Since the data inside the database is reflected by the data types outlined in 3.3, one could implement a class of **MSG_SET** messages for each type. A response with the id of the newly inserted entry could be sent back for types with an id, i.e. appliances, tasks and automation profiles:

- **MSG_SET_APPLIANCE**: Modify an existing appliance (identified through a non-zero id) or add a new one, in which case the new appliance's id is returned.
- **MSG_SET_TASK**: Modify an existing task (identified through a non-zero id) or add a new one, in which case the new task's id is returned. Must only be used by the Automation and Recommendation Module (see 3.5.4)!
- **MSG_SET_AUTO_PROFILE**: Modify an existing automation profile (identified through a non-zero id) or add a new one, in which case the new id is returned. Must only be used by the Automation and Recommendation Module (see 3.5.4)!
- **MSG_SET_ENERGY_CONSUMPTION**: Modify existing energy consumption data (identified through the appliance id and the time) or add a new data point for a given appliance and time.
- **MSG_SET_ENERGY_PRODUCTION**: Modify existing energy production data (identified through the time) or add a new data point for a given time.
- **MSG_SET_WEATHER**: Modify existing weather data (identified through the station and the time) or add a new data point for a given station and time.

Analogous messages can be defined for deleting data:

- **MSG_DEL_APPLIANCE**: Delete an existing appliance (identified through a non-zero id).
- **MSG_DEL_TASK**: Delete an existing task (identified through a non-zero id). Must only be used by the Automation and Recommendation Module (see 3.5.4)!
- **MSG_DEL_AUTO_PROFILE**: Delete an existing automation profile (identified through a non-zero id). Must only be used by the Automation and Recommendation Module (see 3.5.4)!
- **MSG_DEL_ENERGY_CONSUMPTION**: Delete existing energy consumption data (identified through the appliance id and the time).
- **MSG_DEL_ENERGY_PRODUCTION**: Delete existing energy production data (identified through the time).
- **MSG_DEL_WEATHER**: Delete existing weather data (identified through the station and the time).

Querying data is more complex. Modules would want to get all data of a type or a subset that is delimited by some sensible condition. In principle, these conditions could become arbitrarily complex up to requiring a full query language - defeating at least in part the purpose of the Data Storage Module. In practice, it makes sense to identify a scoped set of queries that the other modules require and implement only those.

Considering the requirements of the other modules described in previous subsections, the following queries and respective message types qualify:

- **MSG_GET_APPLIANCES**: Tasks, automation profiles and energy consumption refer to one or several appliances via their ids, so it is sensible to be able to retrieve them by those. The response maps the ids to the respective appliances.
- **MSG_GET_APPLIANCES_ALL**: The User Interface Module and the Automation and Recommendation Module need to be able to have an overview of the registered appliances, e.g. to present them to the user or for task scheduling. Therefore, these modules should be able to request all appliances in bulk and have them delivered in the response. They may also want to filter based on whether appliances are schedulable or not.
- **MSG_GET_TASKS_BY_ID**: Appliances refer to one or several tasks via their ids, so modules should be able to retrieve tasks by their ids. The response maps the ids to the respective tasks.
- **MSG_GET_TASKS_BY_TIME**: The User Interface Module may want to present a view of all future, past or currently running tasks, i.e. all tasks filtered by a time frame. The Automation and Recommendation Module likewise requires

scoped knowledge of existing tasks by time frames instead of only in their totality. Both modules may also want to filter by whether the task was created by the user - the automation module in particular needs to know this, as user-declared tasks are respected in automation. The response maps times to the respective tasks that start there.

- **MSG_GET_TASKS_ALL**: The user may want an overview of all existing tasks, so the User Interface Module should be able to request all tasks in bulk and have them delivered in the response. As above, modules using this message may filter by whether the task was created manually.
- **MSG_GET_AUTO_PROFILES**: Tasks can refer to an automation profile via its id, so modules should be able to retrieve automation profiles by their ids. The response maps the ids to the respective automation profiles.
- **MSG_GET_AUTO_PROFILES_ALL**: Like with appliances and tasks, modules and in particular the User Interface Module may want to get all automation profiles in bulk, and have them included in the response.
- **MSG_GET_ENERGY_PRODUCTION**: Getting all energy production data within a given time frame is required for presentation to the user by the User Interface Module, to train the model (see 3.5.5) or to infer from the model (see 3.5.6). To this end, a message with a time frame is sent and the response maps time points (in the configured fixed cadence) to their respective energy production.
- **MSG_GET_ENERGY_PRODUCTION_ALL**: The user may want to review energy production throughout the entire time where the HEMS was operated in order to diagnose problems and inform purchases, so it should also be possible to get all energy production data as a response to a message. The response contains a map like for the previous message.
- **MSG_GET_ENERGY_CONSUMPTION**: The automation module can optionally update each appliance's power rating as a rolling average, for which it needs to retrieve historical values for each appliance and for a given time frame. The response lets the module choose energy consumption by appliance, and then by time. The times follow the configured fixed cadence. This message may also be used for presentation purposes in the user interface.
- **MSG_GET_ENERGY_CONSUMPTION_ALL**: Analogously to energy production data, the user may want to see all available energy consumption data of their devices to diagnose problems and inform decisions, so getting all energy consumption data for all appliances (or a subset) should be possible. The response follows the same format as for the previous message.
- **MSG_GET_WEATHER**: Weather data is needed by the Knowledge Inference Module and the Model Training Module in order to infer from and train the model.

Both modules would want all weather data for all stations within a given time frame. The response lets the sender choose weather by weather station or time.

A special message retrieves settings from storage:

- **MSG_GET_SETTINGS**: Retrieve the previous session's settings from storage at the next session initialization (3.6.1). Must only be used by the Launcher Module.

## 3.6. Program sequences

As a modular application, most of the HEMS' functionality and scenarios (see 3.1) require the interaction of several modules and the exchange of messages. A detailed exploration of the different kinds of exchanges and each modules' individual functionalities in 3.5 allows in turn to explore the most essential scenarios as program sequences.

### 3.6.1. Session initialization

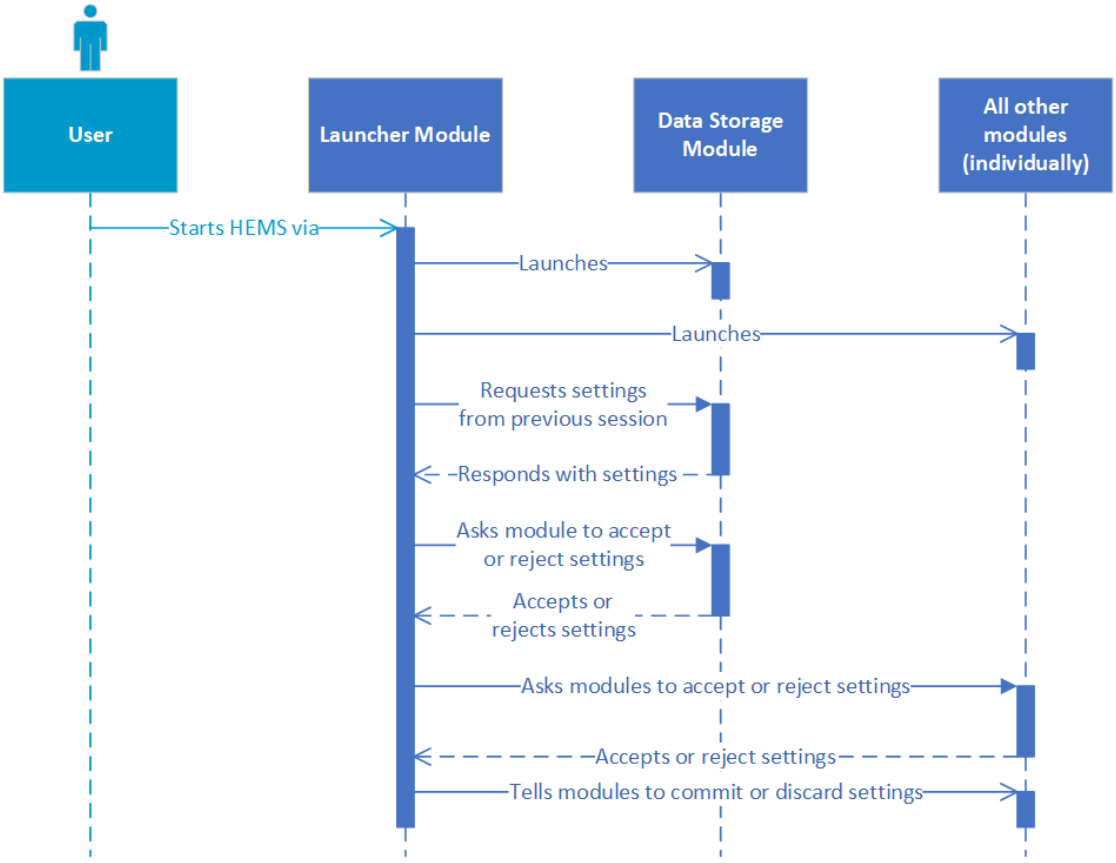



Figure 5: Session initialization

Sessions are initialized by the Launcher Module when the user starts the HEMS application (figure 5). The HEMS sets up inter-module communication and launches all

modules. The launch order is not crucial, as at this point all modules only set up communication and then wait to receive the last session's settings from the HEMS. After the HEMS verifies that all modules have launched through standard process management interfaces, it requests the previous session's settings from the Data Storage Module. If there are none, the launcher constructs new settings out of defaults and call arguments. These new settings are sent to all modules, which can then choose to either accept or reject them.[7] If all modules accept, the launcher then tells all modules to commit those settings, otherwise if one module rejects them, the launcher tells all modules to discard the settings. After settings have been committed, each module can start all its operations that are dependent on the settings and start listening for other messages from modules.

### 3.6.2. Getting recommendations and committing tasks

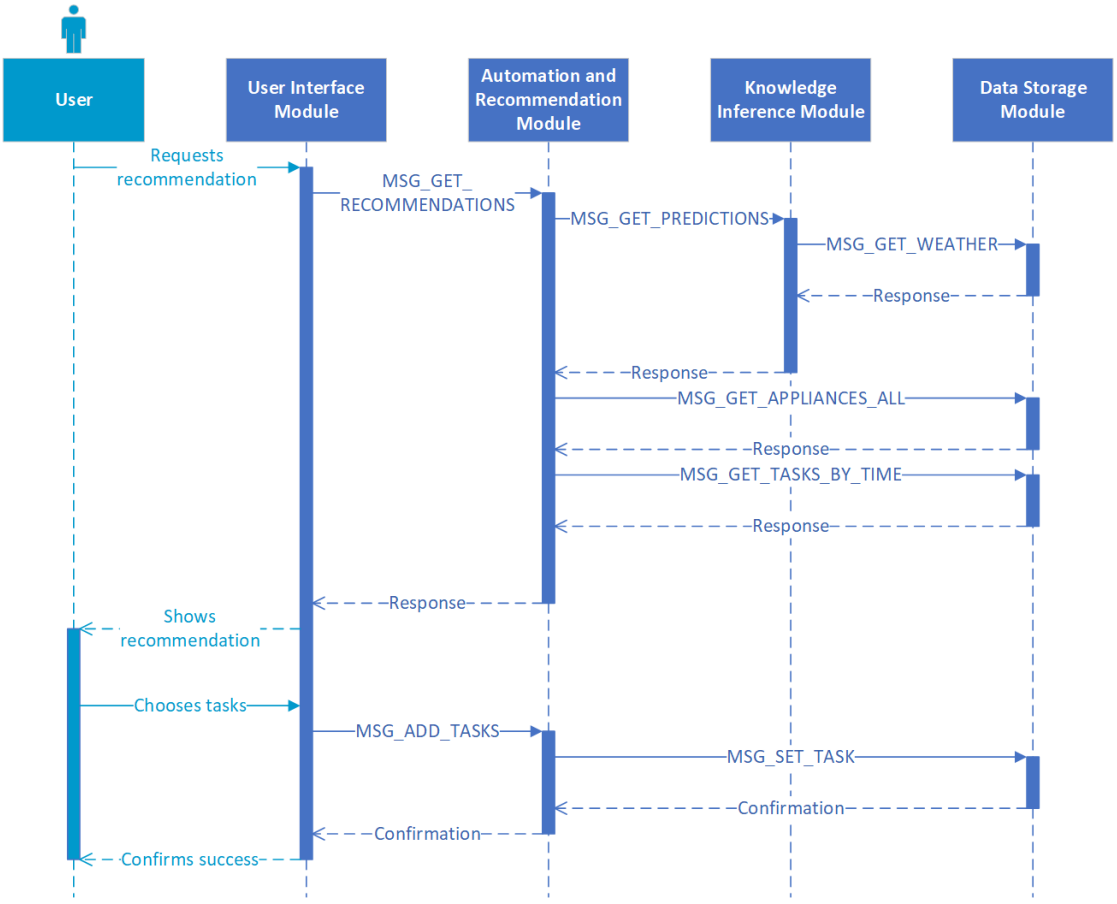



Figure 6: Getting recommendations and committing tasks

[7] It can be assumed that neither the defaults nor the previous settings have any reason to be rejected, so rejection is only expected during runtime when a module wants to change settings.

When the user requests task recommendations for the week ahead, this initiates a sequence (figure 6) that spans a total of four other modules (not counting when these modules send log messages to the Launcher Module). The User Interface Module requests these recommendations from the Automation and Recommendation Module (3.5.4). To provide them, the automation module requires:

- Energy production predictions from the Knowledge Inference Module.
- An overview of all appliances.
- An overview of all tasks that are already scheduled for the week ahead.

In turn, to make these predictions from the model, the inference module needs at least (see 3.5.6 and 5):

- Weather forecasts for the week ahead (acquired from the Data Storage Module).

When inference is done, it can send the predictions back to the automation module that requested them. The automation module then gets the appliances and tasks from the storage module (3.5.7).

The automation module now has all the data it needs to make recommendations. These are sent back to the User Interface Module, which displays them to the user. The user can now choose to commit an arbitrary subset of tasks from this recommendation, which the User Interface Module then lets the automation module know. The tasks that the user chose are sent to the storage module for persistence.

### 3.6.3. Setting appliances

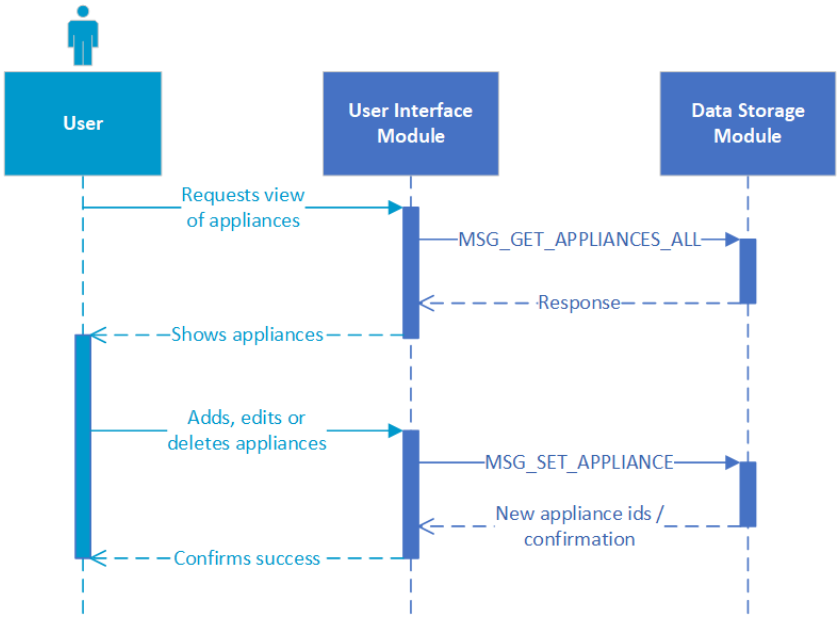



Figure 7: Setting appliances

Setting appliances (figure 7) is straightforward: First, the user requests a view of all appliances through the user interface, which the User Interface Module gets from the Data Storage Module. The appliances are then shown to the user, who chooses to add new appliances or edit or delete existing appliances as shown fit. The User Interface Module sends these to the storage module, which confirms the process or returns the new appliances' ids if new appliances were added. The user interface then confirms the success of this action to the user.

### 3.6.4. Setting tasks

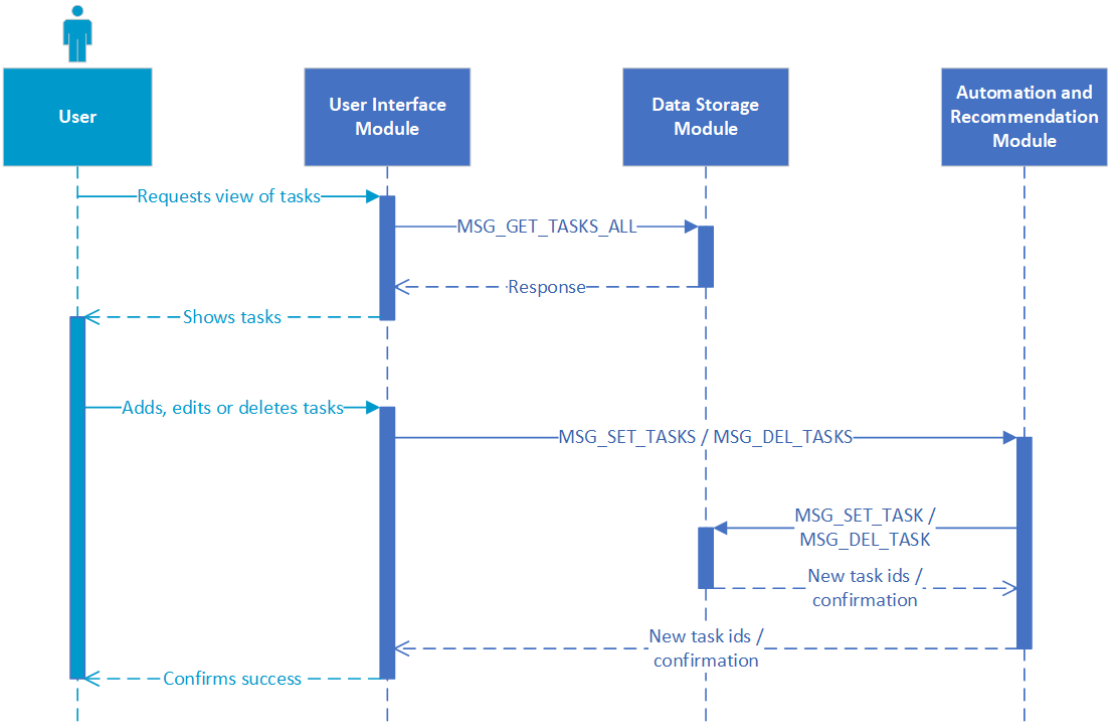



Figure 8: Setting tasks

The process of setting tasks (figure 8) starts analogously to setting appliances: The user first requests a view of all tasks, which the User Interface Module gets from the Data Storage Module. Then the user may choose to add new tasks or modify or remove some.

From here on, the process is more involved because the Automation and Recommendation Module is responsible of automating appliances on their respective schedules, which is why it must be aware of changes or additions to tasks immediately. For this reason, the User Interface Module cannot tell the storage module directly to set tasks but must ask the automation module instead (see 3.5.4). The automation module, now aware of the changes to tasks, tells the Data Storage Module to persist these changes (3.5.7). The storage module responds with a confirmation or with the new tasks' ids if

new tasks were added. Then the automation module can get back the User Interface Module and confirm the process on its end, so that the success of the user's action is displayed on the user interface.

### 3.6.5. Setting automation profiles

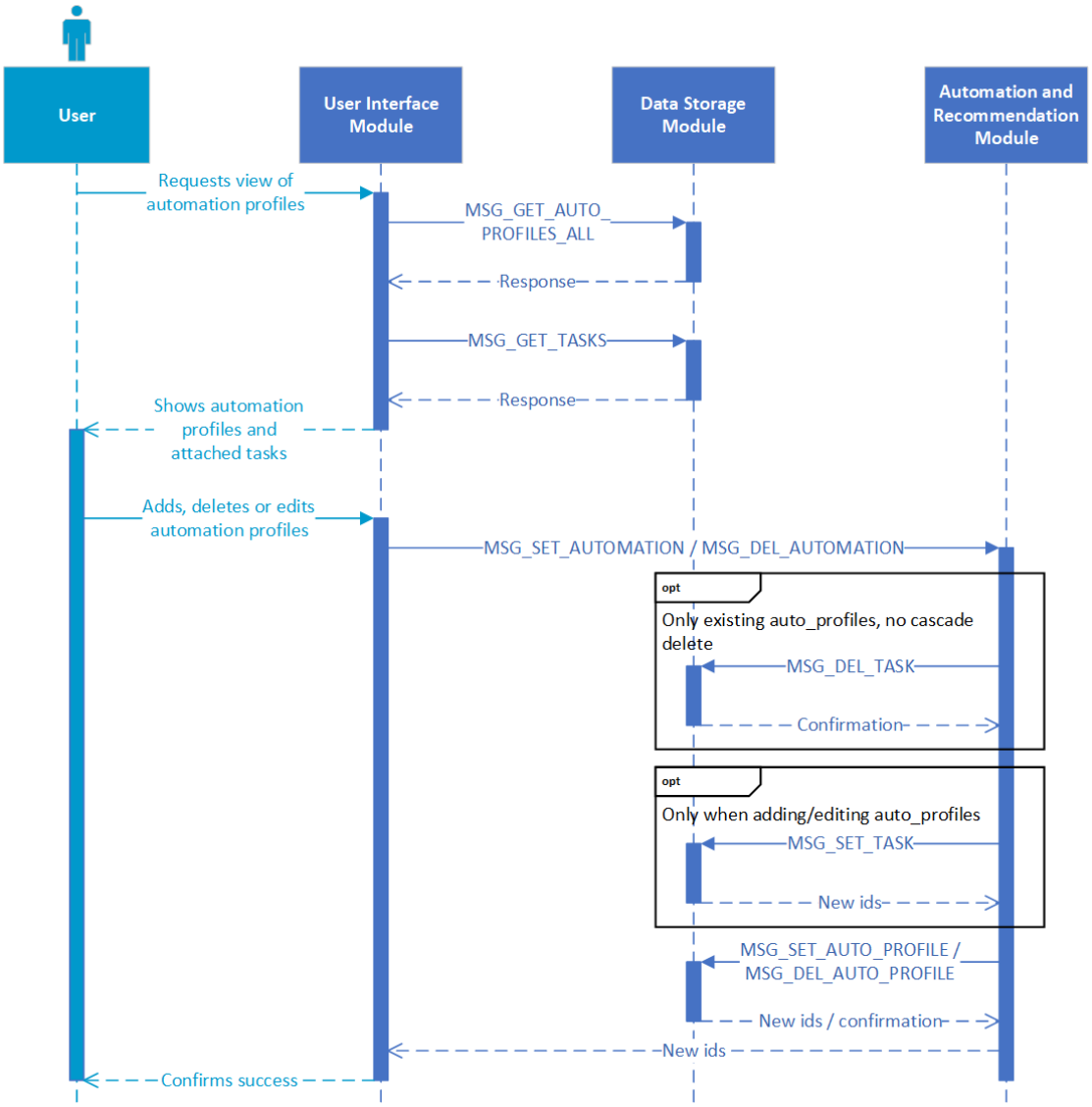



Figure 9: Setting automation profiles

Automation profiles spawn tasks (3.3.2), which is why in order for the user to make informed decisions on which automation profiles to keep, delete, edit or add, they must see which tasks are created out of each automation profile. Therefore, the process of setting automation profiles (figure 9) starts with the User Interface Module requesting

not only all automation profiles from the Data Storage Module, but also all tasks that have been created out of each respective automation profile.

When the user chooses to add, modify or remove automation profiles, the User Interface Module must go to the Automation and Recommendation Module (see 3.5.4) for the same reason as with tasks: Because the automation module must be aware of additions or changes to automation profiles immediately. If an existing automation profile is modified or deleted, all its spawned tasks are first deleted from storage - this may happen automatically if the database system and scheme include a deletion cascade mechanism, or otherwise manually via deletion messages to the Data Storage Module.

If a new automation profile is added or an existing one is modified, the automation module parses the *profile* field in iCalendar syntax (see 3.3.2) and creates corresponding tasks. Then the automation profile is added or deleted. The Data Storage Module responds with a confirmation or with the id of the new automation profile. All ids of new automation profiles are then returned by the automation module to the User Interface Module, which confirms the success of the operation to the user through the user interface.

### 3.6.6. Collection of energy data

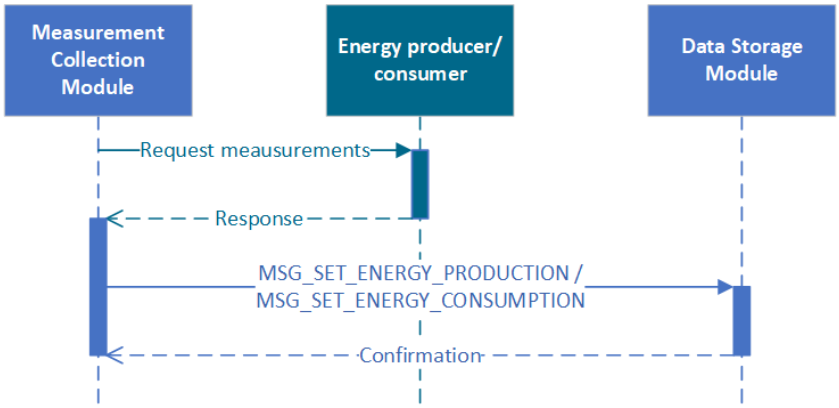



Figure 10: Collection of energy data

To collect energy data, the Measurement and Collection Module requests measurements from the energy producer or a consumer (depending on the kind of energy data). The communication with the producer/consumer leaves the boundaries of the HEMS, as the consumers and the producer are separate components of the infrastructure (see 1). Therefore, this communication does not happen through the inter-module communication mechanism of the HEMS, and is entirely dependent on the specific producer's / consumer's interface. When the measurements have been collected, they are sent directly to the Data Storage Module.

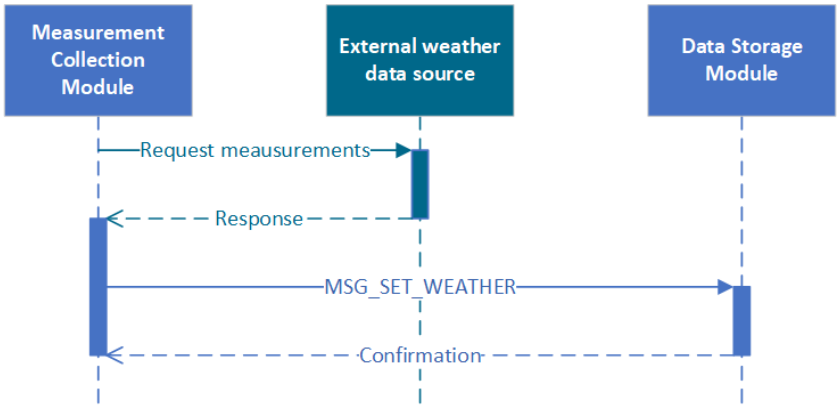



Figure 11: Collection of weather data

### 3.6.7. Collection of weather data

Weather data is collected by communicating with an external weather data source. This source should offer historical data and forecasts. It is a separate component of the infrastructure (see 1) and therefore, communication with the source happens outside the boundaries of the HEMS and does not use its inter-module communication mechanism. When the data has been collected, it is sent directly to the Data Storage Module for persistence.

# 4. Datasets and Data Collection

The HEMS collects energy production and weather data, and optionally energy consumption data. During normal operation, the Measurement Collection Module would collect this data continuously and send it to the Data Storage Module for persistence, where other modules can then request it. However, in the context of this thesis, energy production and weather data was collected in advance. The collection of this data employed prototypical hardware deployments and software interfaces. In the following, the data sources are described and the available data is characterized for each type.

## 4.1. Energy production data

### 4.1.1. Source and collection

Energy production data was collected from two locations: Four roof-mounted panels in Wijchen, Netherlands, and one panel from *SunMan* with a rating of 235 Wp mounted on a balcony in Berlin, Germany (see figure 12). Both installations were connected to a *FRITZ!DECT 200*[8], a smart outlet from *AVM* that can measure energy production or consumption of a connected device. It ties into the AVM *FRITZ!* home network and smart home ecosystem that includes routers, WiFi repeaters, telephones and smart

[8] `https://avm.de/produkte/fritzdect/fritzdect-200/`

thermostats[9]. The smart outlet connects to a router of the *FRITZ!Box* family via *DECT* (Digital Enhanced Cordless Telecommunications). It can then be controlled and interacted with through the FRITZ!Box's graphical web-based interface or through a dedicated HTTP "AHA" (AVM Home Automation) interface [22].

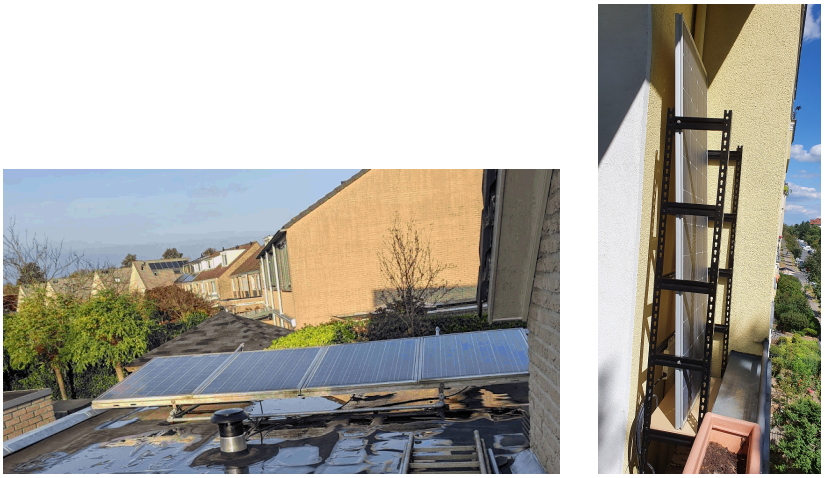


Figure 12: The two PV systems used for this thesis (left: Wijchen, right: Berlin)

Presently, this interface offers a way to get an outlet's current power throughput and its accumulated energy throughput since first use or since the last reset of this statistic. However, unlike the web-based interface it does not offer a way to get energy values with a fixed cadence. The web-based interface does offer an overview of energy consumption in Wh for the past 24 hours on a 15-minute cadence, with a precision of +/- 100 mW (up to 5 Watts) or +/- 2% (5 Watts and above).[10] This data can be acquired in JSON format via an undocumented (but openly available) HTTP interface. A prototype tool was built that calls this interface and collects energy production data once every day, then sends it as a CSV file to a specified email address. In both locations, this tool was deployed on a Raspberry Pi connected to the home network and the internet. The code of this tool can be found in this thesis' GitHub repository under the *tools/energy_collector* directory[11], and the raw data in CSV format collected through it is in the *data* directory[12]. Because this tool uses an undocumented interface, it is not suitable for a production environment and was built exclusively for the context of the thesis.

Table 9 lists additional characteristics of each data source. It should be noted that the high number of small interruptions in Wijchen's dataset is because of unrelated problems on the device that the collection tool was deployed on.

[9] `https://avm.de/produkte/`
[10] `https://avm.de/produkte/fritzdect/fritzdect-200/technische-daten/`
[11] `https://github.com/adrianghc/HEMS/tree/master/tools/energy_collector`
[12] `https://github.com/adrianghc/HEMS/tree/master/data`

| | Source 1 (Wijchen, NL) | Source 2 (Berlin, DE) |
|---|---|---|
| Latitude | 51.8235504 | 52.4652025 |
| Longitude | 5.7329005 | 13.3412466 |
| Time zone | GMT+1 (+ DST) | GMT+1 (+ DST) |
| Orientation | Horizontal | 75°, pointing south |
| Rating | N/A | 235 Wp |
| Brand/model | N/A | SunMan Solarmodul Monokristallin |
| No. of panels | 4 | 1 |
| Time periods of collection | 26-06-20 22:15 - 29-06-20 00:00<br>29-06-20 07:15 - 01-07-20 07:00<br>01-07-20 07:45 - 03-07-20 07:00<br>03-07-20 16:30 - 04-07-20 16:15<br>04-07-20 19:15 - 06-07-20 06:30<br>10-07-20 15:45 - 11-07-20 15:30<br>12-07-20 07:45 - 13-07-20 19:15<br>17-07-20 15:15 - 18-07-20 15:00<br>20-07-20 22:15 - 21-07-20 22:00<br>17-08-20 07:15 - 29-08-20 00:00<br>31-08-20 00:15 - 01-09-20 00:00<br>03-09-20 00:15 - 19-09-20 00:00<br>23-09-20 00:15 - 04-10-20 00:00<br>06-10-20 17:45 - 24-10-20 00:00 | 31-07-20 00:15 - 08-08-20 00:00<br>09-08-20 00:15 - 19-09-20 00:00 |
| Interval of collection | 15 minutes | 15 minutes |

Table 9: Characteristics of the energy production data sources

### 4.1.2. Charaterization and analysis

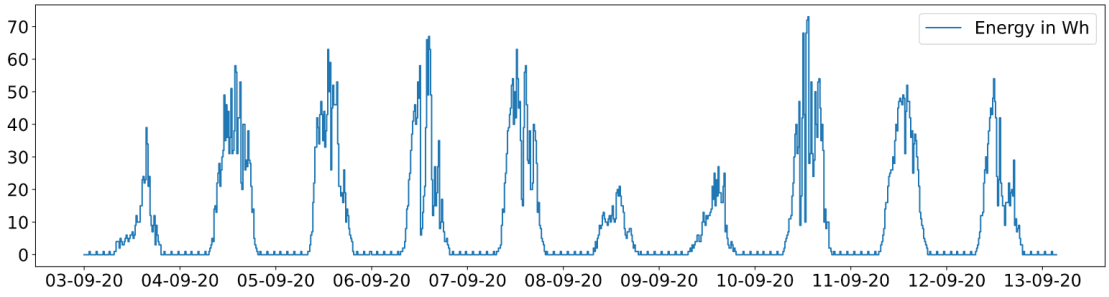



Figure 13: Energy production (Wh) per 15 minutes in Wijchen between 3 September and 13 September 2020

As shown in figures 13 and 14, energy production follows a step function that flattens out at night and reaches its peak around the middle of the day. It can also be seen that there can be significant differences in energy production between very immediate time points, and also between different days. This demonstrates a high responsiveness of the solar installations to changes in environmental factors.

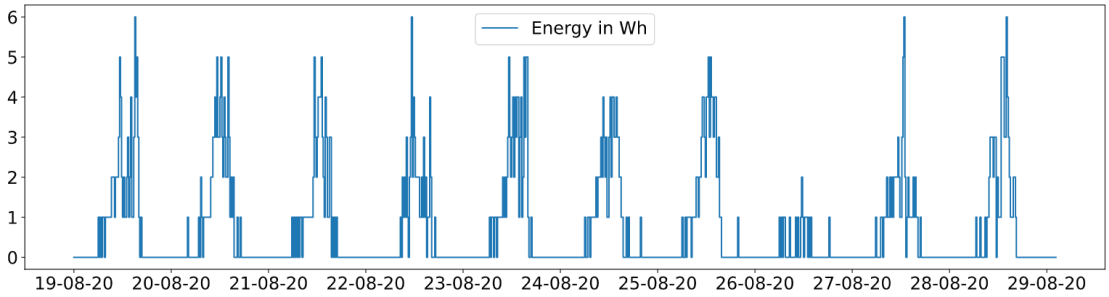



Figure 14: Energy production (Wh) per 15 minutes in Berlin between 19 August and 29 August 2020

Comparing the two datasets (figure 15), it becomes apparent that the Wijchen installation has produced much more energy than the Berlin installation. The greatest value measured is **74** Wh between 13:00 and 13:15 on 12 July 2020, and the average 15-minute value is **9.6** Wh. In contrast, the greatest value measured for the Berlin installation is merely **22** Wh between 11:15 and 11:30 on 31 July 2020, and the average is as low as **0.79** Wh. These numbers are not directly comparable because of different environmental conditions and different time periods for data collection, but even when considering only common times, Wijchen's maximum and average energy are **73** Wh and **11.76** Wh, respectively, in contrast to Berlin's maximum and average energy of **8** Wh and **0.72** Wh.

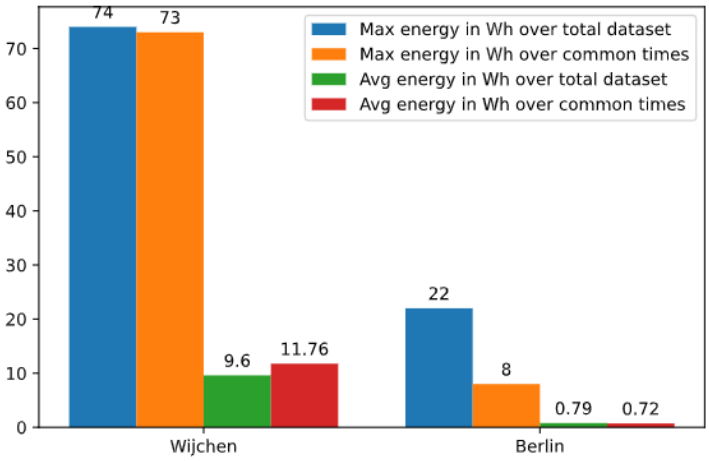



Figure 15: Average and maximum energy values of both datasets

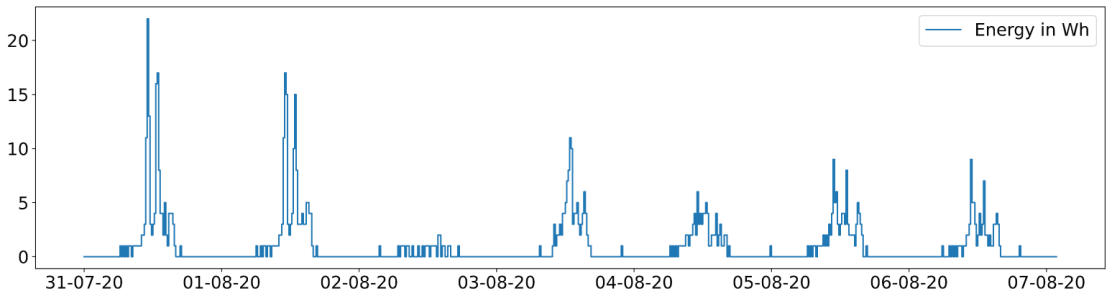



Figure 16: Berlin's peak for maximum energy production in late July / early August

An additional observation (figure 16) is that Berlin's maximum measurement of **22** Wh on 31 July is considerably higher than the maximum over all measurements that begin just two days later, which sinks to just half of that with **11** Wh on 3 August 2020 between 13:00 and 13:15. This change in performance is the result of a change in the panel's orientation that was made because of space constraints. Due to the limitations in the precision of the FRITZ!Dect 200, the lower performance of the Berlin installation also means that its measured energy values have a lower granularity.

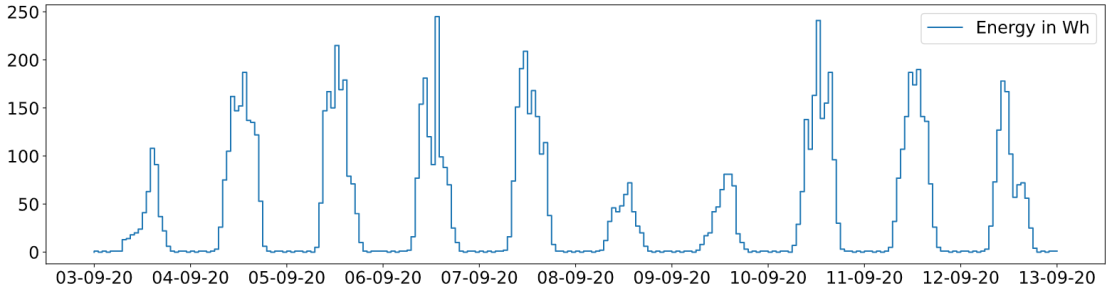



Figure 17: Energy production (Wh) per 1 hour in Wijchen between 3 September and 13 September 2020

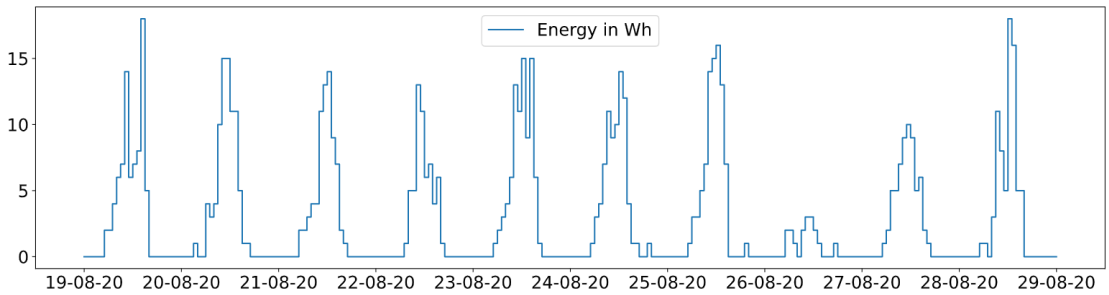



Figure 18: Energy production (Wh) per 1 hour in Berlin between 19 August and 29 August 2020

Figures 17 and 18 show a slice of the collected energy production data for the locations in Wijchen and Berlin, respectively, but integrated over a one-hour cadence. From here on, energy production data will always be used in this one-hour cadence to ensure compatibility with the granularity of available weather forecasts (see 4.2).

## 4.2. Weather data

### 4.2.1. Source and collection

Kachelmannwetter[13] is a Swiss weather site that offers an immense and exhaustive variety of weather data, including satellite or radar pictures, animations and tables, ranging from historical weather measurements to weather forecasts from several computer models. Weather data is available in a large variety of types, including but not limited to temperature (air and surface), precipitation, cloud cover, wind speed, air pressure, humidity and more. It is available for a dense net of weather stations with cadences of up to ten minutes, depending on the type of weather data and on the station.

| | Volkel[14] | Deelen[15] |
|---|---|---|
| Latitude | 51.65 | 52.0667 |
| Longitude | 5.7 | 5.8833 |
| Height | 22 m | 48 m |
| SYNOP station code[16] | 06375 | 06275 |

Table 10: Weather stations for the energy production location in Wijchen

| | Berlin-Dahlem[17] | Berlin-Tempelhof[18] |
|---|---|---|
| Latitude | 52.46 | 52.28 |
| Longitude | 13.30 | 13.24 |
| Height | 51 m | 49 m |
| SYNOP station code | 10381 | 10384 |

Table 11: Weather stations for the energy production location in Berlin

While the site is focused on German-speaking markets, measurements are available for other markets as well, including the Netherlands. Kachelmannwetter offers its services on demand for businesses to integrate into their products on an individual basis.[19] Un-

[13] https://kachelmannwetter.com
[14] https://meteostat.net/en/station/06375
[15] https://meteostat.net/en/station/06275
[16] https://de.wikipedia.org/wiki/SYNOP
[17] https://www.dwd.de/DE/leistungen/klimadatendeutschland/stationsuebersicht.html
[18] https://www.dwd.de/DE/leistungen/klimadatendeutschland/stationsuebersicht.html
[19] https://business.kachelmannwetter.com/produkte-dienstleistungen/

fortunately, an inquiry for data access through a well-defined interface for the purposes of this thesis was declined due to lack of capacity on Kachelmannwetter's end. Therefore, while weather data from Kachelmannwetter was used in the context of this thesis, no tool to download this data can be publicly provided.

For each of the two locations where energy production data was collected (see 4.1), two weather stations were identified that were deemed to be in sufficient proximity to each location, listed in tables 10 and 11.

### 4.2.2. Characterization and analysis

The results of this section were derived with the *scikit-learn* machine learning framework for Python (version 0.24).[20]

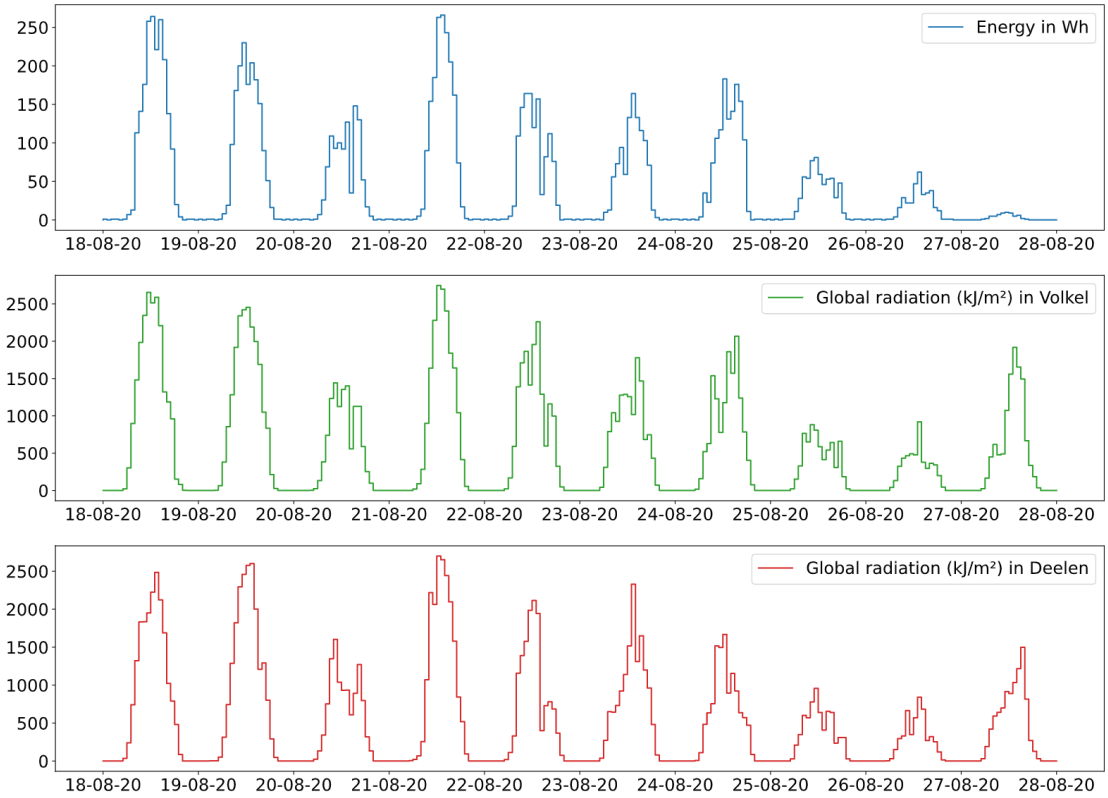



Figure 19: Energy production and global radiation (source in appendix A) in Volkel and Deelen between 18 August and 28 August 2020

A number of environmental factors, particularly weather factors, can influence the performance of the PV system and its energy output. The **global radiation**[27][28], measured in kJ or Wh per $m^2$, stands out in particular. Global radiation is measured with surface-based pyranometers or with satellite measurements of cloud cover and other

[20] `https://scikit-learn.org/stable/index.html`

parameters such as water vapor and aerosol amounts in the atmosphere. It is a more advanced type of weather data for which e.g. Kachelmannwetter restricts forecasts behind a paywall, but for which it makes historical measurements freely available.

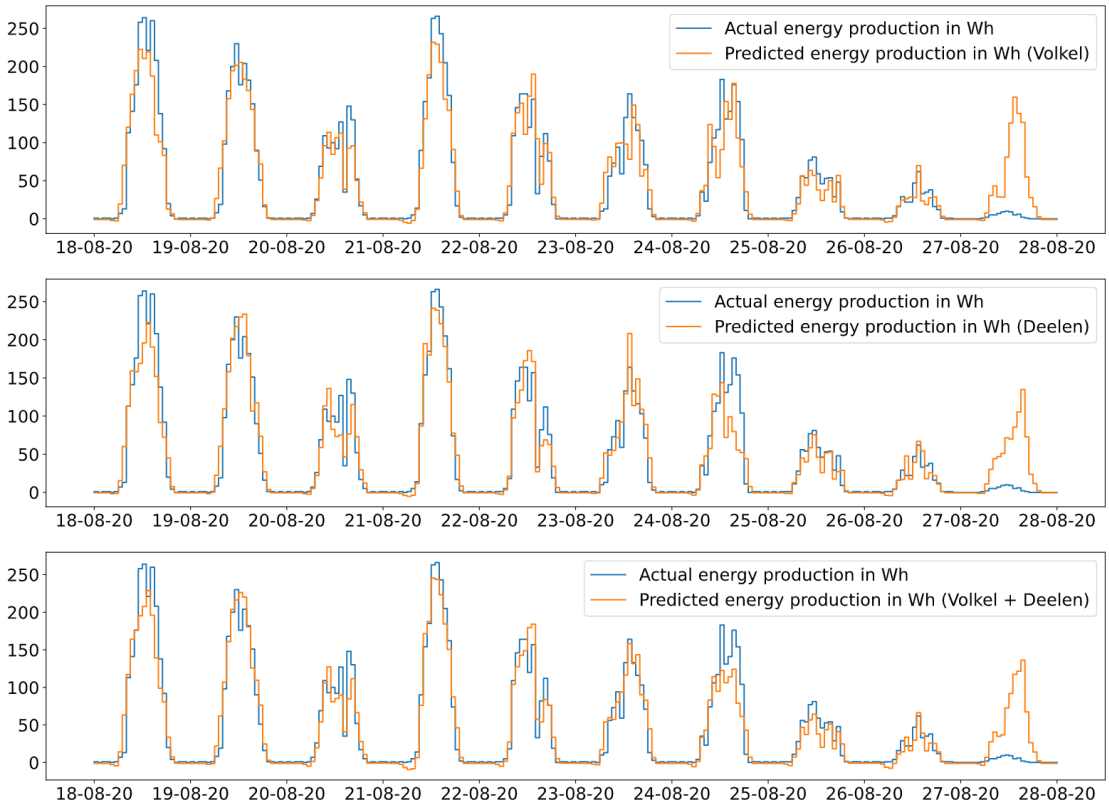



Figure 20: Linear regression of sunlight angles and global radiation (source in appendix A) from Volken and Deelen between 18 August and 28 August 2020

Indeed, for the Wijchen location it appears there is a high correlation between the energy production and the observed global radiation at Volkel and Deelen, as shown in figure 19. Furthermore, figure 20 shows the observed energy production signal over ten days and the predicted energy production signal for linear regression models using the sunlight angle and global radiation from Volkel, from Deelen, and both. The coefficients of determination (scores) $R^2$ go as high as **0.8356**, **0.8558** and **0.8739**, respectively[21]. Not only are these very good scores, they even increase to **0.9123**, **0.9029** and **0.9345**, respectively, when ignoring what appears to be an outlier in energy production on 27 August.

[21]The coefficient of determination $R^2$ of a model is defined as $1 - \frac{u}{v}$, where $u = \sum (y_i - \hat{y}_i)^2$ and $v = \sum (y_i - \overline{y})^2$. $y_i$ are the observed values, $\overline{y}$ is the arithmetic mean and $\hat{y}_i$ are the predicted values. The best possible value (score) for $R^2$ is 1, and worse values are less than 1 and can get arbitrarily big in the negative range. A constant model that always predicts $\overline{y}$ and ignores the input has a score of 0. The score describes the proportion of variance in the observed values that can be predicted by the model.

In contrast, the correlation between energy production and observed radiation appears much lower for the Berlin location and the Berlin-Tempelhof weather station, as shown in figure 21. A linear regression as shown in figure 22 using the sunlight angle and global radiation confirms this, producing an $R^2$ score of just **0.6299**. This suggests that basing a model on global radiation may not be the best approach for all locations and environments, depending on the proximity of weather stations to the user's location, on the quality of the solar installation, and other factors. Therefore, it may be sensible to look at weather parameters beyond the global radiation in order to improve the model.

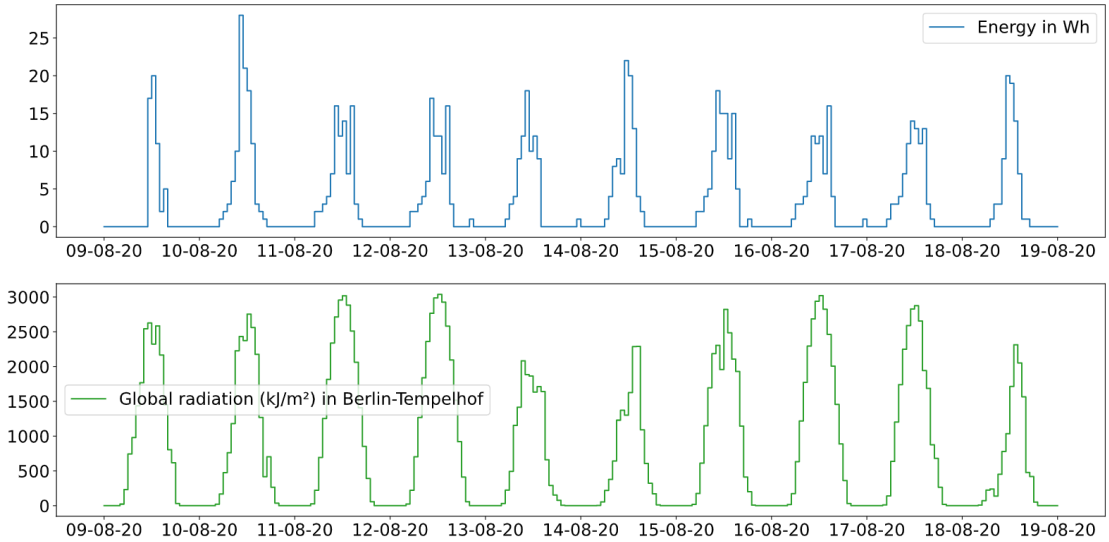



Figure 21: Energy production and global radiation (source in appendix A) in Berlin-Tempelhof between 9 August and 19 August 2020

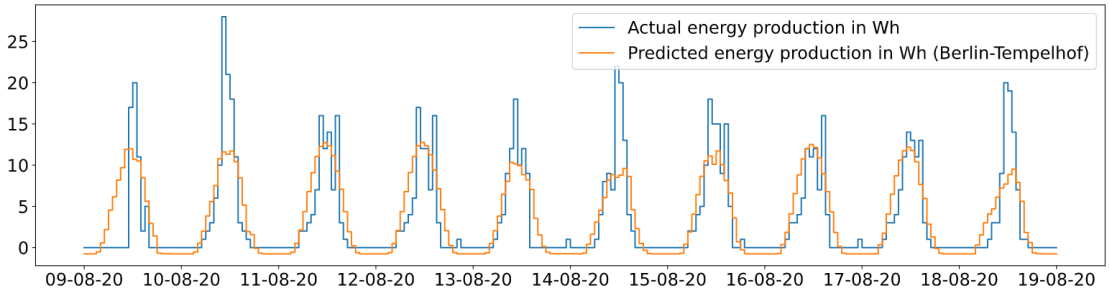



Figure 22: Linear regression of sunlight angles and global radiation (source in appendix A) from Berlin-Tempelhof between 9 August and 19 August 2020

Additionally, beyond the consideration of improving the predictions, the choice of weather parameters may be constrained by pricing and licensing of weather sources for more advanced parameters. Due to the nature of the HEMS as an application for a home environment, this can be a limitation both for private, independent deployments and for

products intended for a residential environment. There can therefore be additional value in attempting to predict energy production based on a scoped set of common weather parameters that are more accessible than global radiation.

An obvious and readily available one is cloud cover, i.e. the fraction of sky freely visible from a location (given in percent or in *okta*[24]): The greater the cloud cover is, the less sunlight can potentially reach the installation. In fact, since it is the main cause of reduction in solar radiation, short term predicting of cloud cover can improve the benefits of solar power plant operation [25]. Therefore, this raises the question whether energy production could not be inferred only from cloud cover and flattened sunlight angles.


Figure 23: Energy production and flattened sunlight angle in Wijchen and cloud cover (see appendix A) in Volkel and Deelen between 18 August to 28 August 2020

Figure 23 shows the following signals from 18 August to 28 August 2020:

1. The first signal represents energy production in the Wijchen location.
2. The second signal represents the flattened sunlight angle.
3. The third signal represents cloud cover from the Volkel station.
4. The fourth signal represents cloud cover from the Deelen station.

Assuming that the energy production signal can be constructed as a linear combination of the cloud cover and sunlight angle signals, a linear regression should be able to find coefficients for which this is the case. Figure 24 shows the actual, observed energy production signal over ten days and the predicted energy production signal for linear regression models using the sunlight angle and cloud cover data from Volkel and from Deelen, respectively. The $R^2$ scores are **0.4941** and **0.4642**, respectively, which point to poor prediction accuracies.

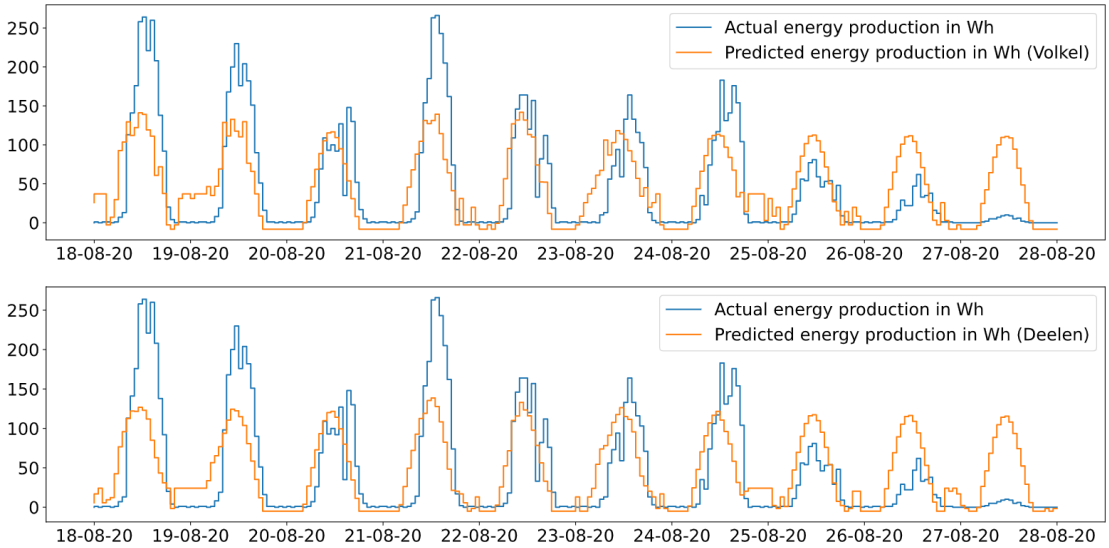



Figure 24: Linear regression of sunlight angles and cloud cover (source in appendix A) from Volken and Deelen between 18 August and 28 August 2020

Two possible explanations are:

1. The cloud cover signals from Volkel and from Deelen may both differ too much from what would be the cloud cover signal at the user's location if it were known. In fact, as figure 25 shows, there can be considerable differences between the cloud cover signals for the Volkel and Deelen stations.
2. Even the cloud cover signal at the user's location could only approximately describe the amount of blocked sunlight, as it does not differentiate between cloud types that may allow different amounts of direct or diffuse radiation through [24][26].

These observations reinforce the need to not rely on one single station for weather data, and to not rely on cloud cover exclusively. Using additional types of weather data for each station could improve the prediction of energy production by adding more environmental signals. Possible weather parameters are e.g. **temperature**, **humidity** and **pressure**. Temperature and humidity in particular have a direct effect on solar PV performance, where higher values lead to lower performance [29][30][31][32]. All three may have indirect effects on PV performance individually or through their interplay, e.g. a lower air pressure drives a higher cloud cover, and temperature influences air pressure. Short term changes in temperature can also be a result of short term changes in cloud cover.

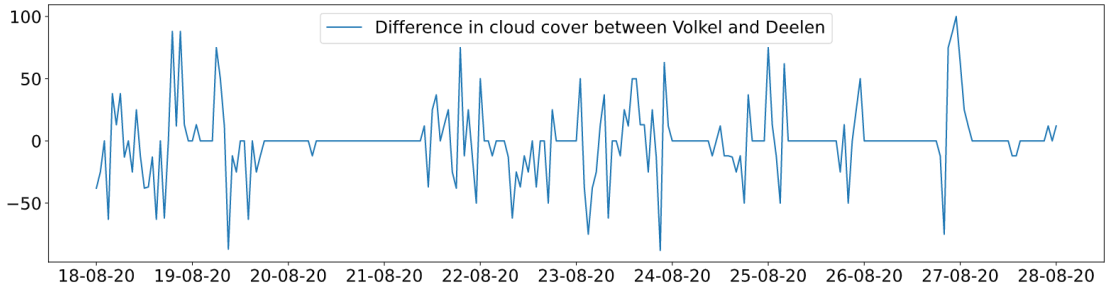



Figure 25: Difference in cloud cover (%) between Volkel and Deelen, 18 August and 28 August 2020

Kachelmannwetter offers all the hitherto discussed weather parameters for most stations as historical measurements and as forecasts from a variety of models, including but not limited to *ECMWF* (European Centre for Medium-Range Weather Forecasts)[22] and *GFS* (Global Forecast System)[23] (except global radiation forecasts). Table 12 shows the respective availability of measurements and forecasts for the Volkel, Deelen, Berlin-Dahlem and Berlin-Tempelhof stations. Weather forecasts extend at least ten days into the future for each weather parameter. Historical measurements, where available, always have a cadence of at least one hour. While higher cadences exist for several weather parameters, forecast cadences are never lower than one hour, and go up to three hours after some days (depending on the model). This limits the cadence of historical measurements that can be used to train models.

Historical measurements of weather parameters must also be available as forecasts and vice versa. This is why despite there being cloud cover forecasts that differentiate between low, median and high clouds (see the weather forecast sources above), this differentiation cannot be used as it does not exist for historical measurements.

Since Berlin-Tempelhof has no cloud cover measurements, the measurements from nearby Berlin-Dahlem are used for both stations. The same is done for global radiation measurements, which are only available for Berlin-Tempelhof but not Berlin-Dahlem.

During regular operation of the HEMS, weather forecasts are used like future mea-

[22] `https://www.ecmwf.int/en/about`
[23] `https://www.emc.ncep.noaa.gov/emc/pages/numerical_forecast_systems/gfs.php`

surements, i.e. no provisions are made to account for forecast inaccuracies and they are instead treated as accurate. However, the collected weather data for any future time becomes more accurate as that time approaches, since weather forecasts are updated continuously (see 3.5.3). For the purposes of this thesis and particularly model development and evaluation, only historical measurements were used, and further possible considerations to account for differences between historical measurements and forecasts beyond the already stated are out of scope.

| | Volkel | Deelen | Berlin-Dahlem | Berlin-Tempelhof |
|---|---|---|---|---|
| Global radiation ($kJ/m^2$) | Yes | Yes | No | Yes |
| Cloud cover (%) | Yes | Yes | Yes | No |
| Temperature (°C) (2m) | Yes | Yes | Yes | Yes |
| Humidity (%) | Yes | Yes | Yes | Yes |
| Pressure (hPa) (station altitude) | Yes | Yes | Yes | Yes |

Table 12: Availability of weather data (historical measurements & forecasts) per station (all sources in appendix A)

## 4.3. Executive summary

Sections 4.1 and 4.2 discussed the specific energy production and weather data that was collected in the context of this thesis. Energy production was measured in two locations with different solar installations and characteristics. Weather measurements were collected from two weather stations for each location. The most important weather parameter is global radiation, as that proves to have the greatest linear correlation to energy production, especially for the Wijchen dataset. However, it was established that availability of global radiation measurements and in particular forecasts may be restricted, in which case other weather parameters, most notably cloud cover, may be used in order to attempt to achieve models of similar quality as when using global radiation, or in oder to improve on models based on global radiation.

The collection of weather and energy production data serves two purposes: On the one hand, they are the data that the HEMS requires for operation, so the data that was collected a priori can be used to simulate the operation a posteriori. On the other hand, the data and the analysis of its characteristics serves to inform how exactly it should be used to achieve the HEMS' functionality of recommending tasks through predictions of energy production. Specifically, this question must be answered: *Which models are the most effective in inferring energy production from weather data?* To answer this, linear and non-linear models are evaluated for different configurations of weather parameters, different training and validation time frames, and more.

# 5. Predicting Energy Production

## 5.1. Linear regression

The Python-based *scikit-learn* machine learning framework (version 0.24) was used for training and inference of linear regression models of this section. The code for this can be found in this thesis' GitHub repository under the *models/energy_production_lr* directory.[24]

It was demonstrated in section 4.2 that linear regression can show very good or just mixed results, depending on the energy production data and whether cloud cover or radiation data was used. The introduction of additional types of weather data poses the question of how linear regression may fare with those additional variables for each energy production dataset.

The HEMS scenario (see 3.1) assumes that a user deploying an HEMS application would not want it to collect data for too long before producing output for the first time. Therefore, models should be able to infer predictions as soon as possible, preferably after training with datasets of a length between one week or no more than two weeks. A length of ten days presents a suitable middle ground.

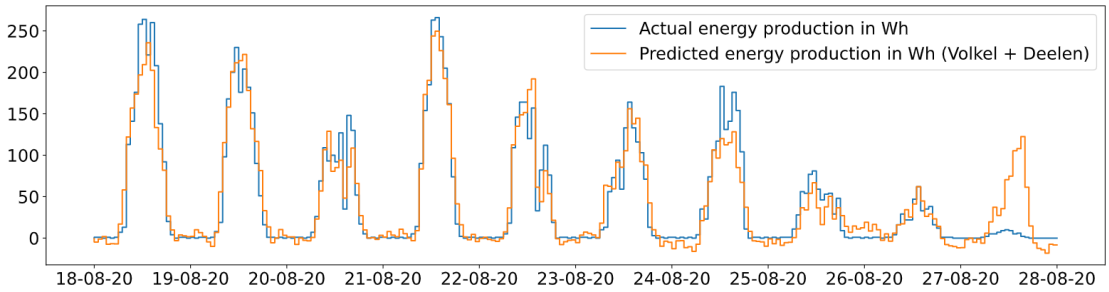



Figure 26: Linear regression of sunlight angles and all weather parameters from Volkel and Deelen between 18 August to 28 August 2020

Recall that linear regression on a ten-day time span (18 August to 28 August 2020) of sunlight angles and global radiation achieved $R^2$ scores of over **0.83** for Wijchen and the Deelen and Volkel stations, demonstrating a high correlation between global radiation and energy production in Wijchen. These scores improve when adding more weather parameters to the linear regression, reaching **0.8447**, **0.8621** and **0.8847** when using data from Volkel, from Deelen, and from both stations, respectively, when keeping the apparent energy production outlier on 28 August. Figure 26 shows real and predicted energy production signals for Wijchen between 18 August and 28 August 2020.

Applying this model to a validation time frame of 23 September to 3 October (see figure 27) shows scores of **0.8485**, **0.8734** and **0.8670**, which are roughly as good as or even slightly better than the training scores due to the lack of outliers in this time range. This shows that the linear regression model is able to generalize and performs well on

[24] `https://github.com/adrianghc/HEMS/tree/master/models/energy_production_lr`

validation if it does so on training, suggesting that in such cases continuous retraining of the model to maintain accuracy may not or only rarely be needed because the linear correlation factors between radiation and energy production change only little over time.

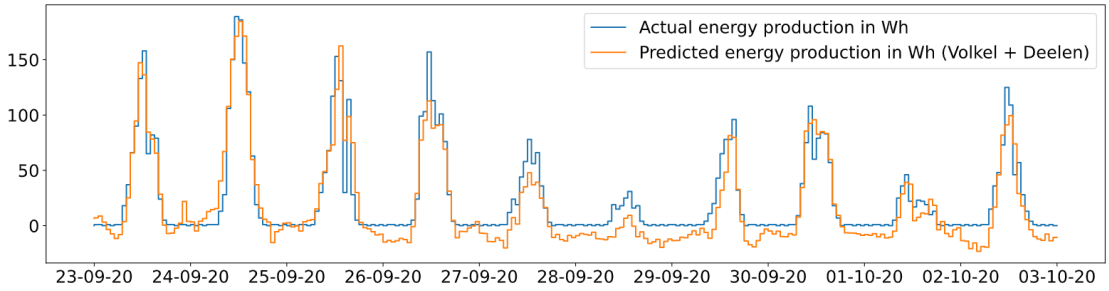



Figure 27: Linear regression of sunlight angles and all weather parameters from Volkel and Deelen between 23 September and 3 October 2020, using the model from 18 August to 28 August

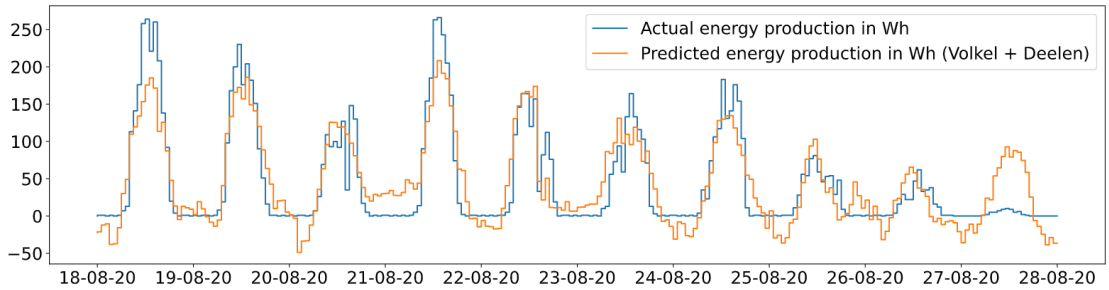



Figure 28: Linear regression of sunlight angles and all weather parameters but radiation from Volkel and Deelen between 18 August to 28 August 2020

Figure 28 shows the linear regression over the time span of 18 August and 28 August 2020 using all weather parameters **except radiation**. The $R^2$ scores are **0.7317**, **0.7419** and **0.7673** when using data from Volkel, from Deelen, and both, respectively.

However, as figure 29 shows, the prediction quality greatly decreases when applying this model to a different time frame, with $R^2$ scores of **-0.7204**, **0.4313** and **-0.1365**, respectively. Indeed, the predicted energy production signal appears markedly shifted down to the negative range, suggesting that the model overfits to the sunlight angles, which for the time frame of 23 September to 3 October range up to 37.4622 degrees, as opposed to 50.8342 degrees for the original time frame of 18 August to 28 August 2020. However, forgoing the sunlight angle signal in addition to the radiation signal is not a good option, as that yields $R^2$ training scores as low as **0.5141**, **0.4946** and **0.5702**, respectively, for the time frame of 18 August to 28 August 2020. These observations imply that there is a weak linear correlation between the non-radiation weather parameters and energy production. Figure 30 demonstrates example predictions obtained without the sunlight angle signal.

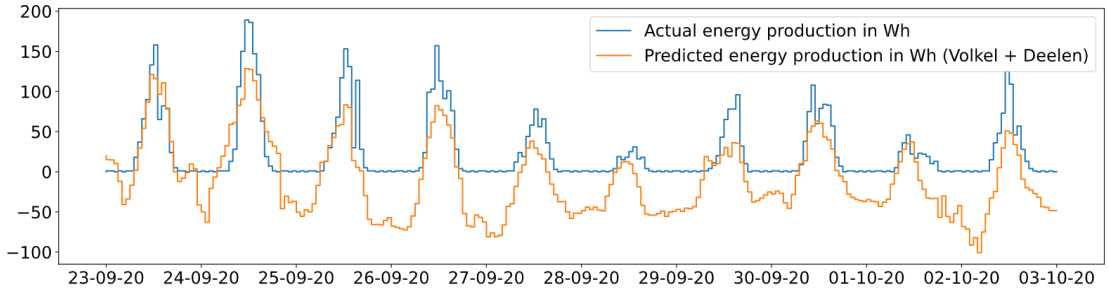



Figure 29: Linear regression of sunlight angles and all weather parameters but radiation from Volkel and Deelen between 23 September and 3 October 2020, using the model from 18 August to 28 August 2020

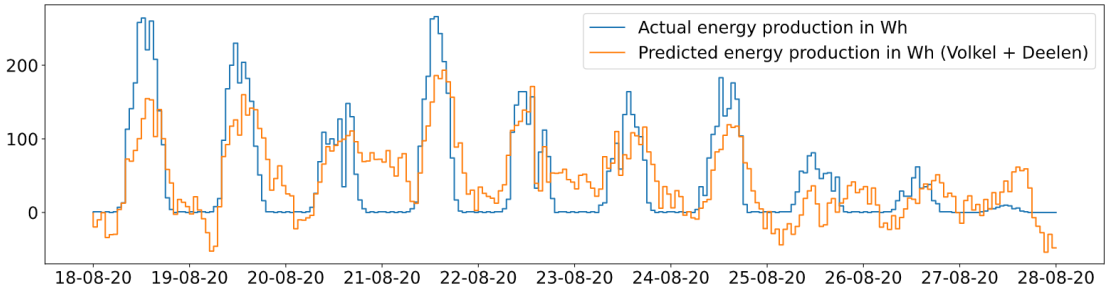



Figure 30: Linear regression of all weather parameters but radiation and no sunlight angles from Volkel and Deelen between 18 August to 28 August 2020

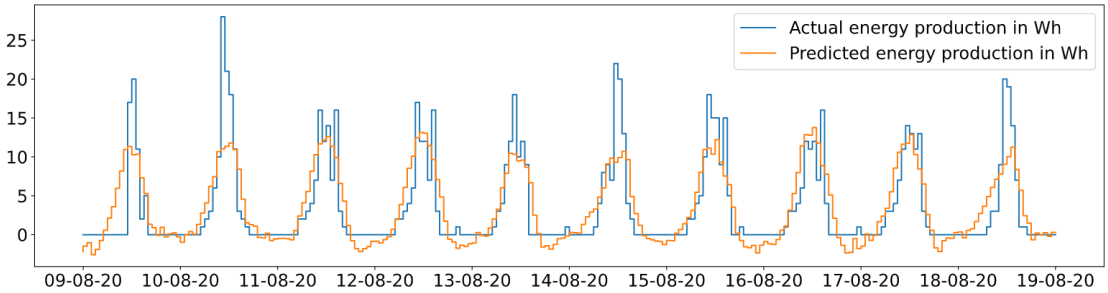



Figure 31: Linear regression of sunlight angles and all weather parameters from Berlin-Dahlem and Berlin-Tempelhof between 9 August and 19 August 2020

Recall that the Berlin energy production dataset had a much weaker correlation to global radiation with an $R^2$ score of just **0.6579** for 9 August to 19 August 2020. Adding all other weather parameters from Berlin-Dahlem and Berlin-Tempelhof increases this score to just **0.6792** (see figure 31), underscoring the weak linear correlation between non-radiation weather parameters and energy production, and highlighting the limitations of this model. It is particularly notable how the model fails to predict the

distinguished peaks around most mid-days of the original data.

| | Wijchen | | | Berlin | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 1+2 | 1 | 2 | 1+2 |
| **Only temperature** | | | | | | |
| Training | 0.6282 | 0.6199 | 0.6405 | 0.6282 | 0.6199 | 0.6405 |
| Validation 1 | 0.3945 | 0.4157 | 0.3882 | 0.5745 | 0.6025 | 0.5604 |
| Validation 2 | -0.2594 | -0.2555 | -0.2207 | 0.5822 | 0.6413 | 0.5847 |
| **Only humidity** | | | | | | |
| Training | 0.6792 | 0.7172 | 0.7178 | 0.6222 | 0.6185 | 0.6230 |
| Validation 1 | 0.7618 | 0.7441 | 0.7484 | 0.6248 | 0.6353 | 0.6165 |
| Validation 2 | 0.5482 | 0.5805 | 0.5750 | 0.7288 | 0.7396 | 0.7232 |
| **Only pressure** | | | | | | |
| Training | 0.4439 | 0.4429 | 0.4684 | 0.6071 | 0.6072 | 0.6099 |
| Validation 1 | 0.5336 | 0.5449 | 0.5717 | 0.5865 | 0.5857 | 0.6031 |
| Validation 2 | 0.4628 | 0.4898 | 0.2936 | 0.7142 | 0.7124 | 0.7064 |
| **Only cloud cover** | | | | | | |
| Training | 0.4941 | 0.4642 | 0.4942 | 0.6065 | N/A | N/A |
| Validation 1 | 0.5225 | 0.5554 | 0.5205 | 0.6093 | N/A | N/A |
| Validation 2 | 0.4525 | 0.4959 | 0.4527 | 0.7403 | N/A | N/A |
| **Only radiation** | | | | | | |
| Training | 0.8356 | 0.8558 | 0.8739 | N/A | 0.6579 | N/A |
| Validation 1 | 0.8995 | 0.8966 | 0.9197 | N/A | 0.7160 | N/A |
| Validation 2 | 0.9062 | 0.9059 | 0.9352 | N/A | 0.8331 | N/A |
| **All but radiation** | | | | | | |
| Training | 0.7317 | 0.7419 | 0.7673 | 0.6338 | 0.6238 | 0.6531 |
| Validation 1 | 0.7718 | 0.8182 | 0.7954 | 0.5422 | 0.5908 | 0.5435 |
| Validation 2 | -0.7204 | 0.4313 | -0.1365 | 0.1968 | 0.4197 | 0.3109 |
| **All with radiation** | | | | | | |
| Training | 0.8447 | 0.8621 | 0.8847 | 0.6631 | 0.6681 | 0.6792 |
| Validation 1 | 0.8648 | 0.8784 | 0.8761 | 0.6942 | 0.6360 | 0.6499 |
| Validation 2 | 0.8485 | 0.8734 | 0.8670 | 0.8472 | 0.8152 | 0.8114 |

Table 13: $R^2$ scores for a variety of constellations of weather parameters and stations (10-day linear regression training). The best three training and validation scores per location are marked in green, the worst three are marked in red.

Table 13 shows training and validation $R^2$ scores for a number of constellations across weather parameters (always with flattened angles) for the Wijchen and Berlin locations. Training and validation data time frames are listed in table 14.

The best three training and validation scores per location are marked in green while the worst three are marked in red. Some key observations are as follows:

1. While the best training scores are achieved with all weather parameters, using only radiation yields a more general model that achieves better validation scores

and only slightly worse training scores, pointing to slight overfitting when using all weather parameters.

2. Notably, training scores when using only cloud cover are only on the lower end.
3. When using global radiation is not an option, using all other weather parameters together achieves the best training scores. However, for the Berlin dataset there is only a small margin to scores of other constellations, and validation scores are notably much worse than for any weather parameter individually.
4. Scores are not entirely comparable between Wijchen and Berlin even for weather parameters besides radiation, e.g. validation scores are consistently much worse for Wijchen than for Berlin.
5. Remarkably, validation scores are sometimes better for time frames that are further away from the training time frame than they are for the closer validation time frames.

| | Wijchen | Berlin |
|---|---|---|
| Training | 18 August - 28 August | 9 August - 19 August |
| Validation 1 | 3 September - 13 September | 22 August - 1 September |
| Validation 2 | 23 September - 3 October | 8 September - 18 September |

Table 14: Training and validation time frames (10-day linear regression training)

Even though the application scenario demands that the user does not wait too long before receiving output for the first time, the model is less constrained in the amount of data it can use when the HEMS has been in operation for longer and has already been producing recommendations. E.g, while the HEMS may collect data for ten days before training the first model, later models may have more data at their disposal. Table 16 explores whether a longer time frame for training data of around a month may be beneficial when using only radiation and flattened angles, when using all weather parameters except for radiation, or when using all weather parameters. For better comparability, models are validated over the same ten-day time frame as the second validation time frame from before, and they are trained over a one-month time frame as shown in table 15.

| | Wijchen | Berlin |
|---|---|---|
| Training | 18 August - 19 September | 9 August - 8 September |
| Validation | 23 September - 3 October | 8 September - 18 September |

Table 15: Training and validation time frames (1-month linear regression training)

Both training and validation scores are consistently better than when training over ten days. Therefore, the main limitation of the linear regression model remains even with

| | Wijchen | | | Berlin | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 1+2 | 1 | 2 | 1+2 |
| **Only radiation** | | | | | | |
| Training | 0.8933 | 0.9017 | 0.9170 | N/A | 0.6943 | N/A |
| Validation | 0.9132 | 0.9026 | 0.9410 | N/A | 0.8425 | N/A |
| **All but radiation** | | | | | | |
| Training | 0.7644 | 0.7751 | 0.8911 | 0.6399 | 0.6398 | 0.6431 |
| Validation | 0.3947 | 0.5596 | 0.2805 | 0.7775 | 0.7816 | 0.7829 |
| **All with radiation** | | | | | | |
| Training | 0.8934 | 0.9028 | 0.9189 | 0.6982 | 0.6986 | 0.6995 |
| Validation | 0.9147 | 0.9047 | 0.9312 | 0.8499 | 0.8493 | 0.8477 |

Table 16: $R^2$ scores for a variety of constellations of weather parameters and stations (1-month linear regression training)

bigger training datasets: It is very effective when the global radiation data correlates highly with energy production, but when this is not the case, linear regression is unable to make up for it through the inclusion of additional weather parameters.

Two possible reasons for this are:

1. Weather parameters and energy production may be correlated, but not linearly, e.g. there may be a linear correlation between energy production and the squared weather signal. Linear regression would be unable to model these non-linear correlations if they exist.

2. There may be other unknown signals, not all of which may be measurable through observation of the weather. For example, depending on the PV system's position and orientation, nearby objects such as other buildings may cast shadows that block sunlight. This type of "invisible" signal could be a singular interference or reoccur periodically. The linear regression model would not be able to learn these "invisible" signals that are not linearly dependent of the measured weather signals unless they were linear functions of the time.

## 5.2. Neural networks

Neural network models were trained, validated and evaluated with TensorFlow 2.4.1. The code for this can be found in the thesis' GitHub repository under the *models/energy_production_nn* directory[25].

A neural network may be able to model non-linear correlations between weather signals and energy production, and it may also be capable of modelling "invisible" signals. The latter would require a recurrent neural network (RNN) architecture [33] that is capable of inferring from time sequences, such as an LSTM (long short-term memory)

[25] `https://github.com/adrianghc/HEMS/tree/master/models/energy_production_nn`

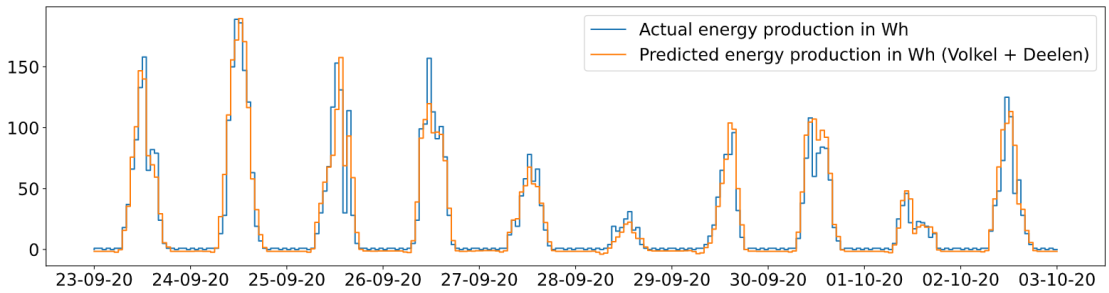



Figure 32: Linear regression of sunlight angles and radiation from Volkel and Deelen between 23 September and 3 October 2020, using the model from 18 August to 19 September 2020

[34]. Whether any significant invisible signals exist is highly dependent on the user's environment, including factors such as the solar installation's positioning, orientation, nearby objects and the PV hardware itself. The high correlation between radiation and energy production for Wijchen in particular suggests a lack of this kind of signal apart from single outliers such as the one on 27 August mentioned before and shown in figure 20. Therefore, further analysis of the Wijchen and Berlin datasets to this regard is out of the scope of this thesis.

However, modelling non-linear correlations between signals could be achieved by a simple feed-forward neural network.

The main question is whether a neural network may be able to improve on the linear regression models' predictions. For the highest comparability this is tested by training and validating with the same time frames as in table 14 (for 10-day training) and table 15 (for 1-month training).

| | Minimum | Maximum |
|---|---|---|
| Energy production (Wh) | 0 | 291 (Wijchen)<br>49 (Berlin) |
| Sunlight angles (°) | -90[26] | 90 |
| Radiation (kJ/m²) | 0 | 4000 |
| Cloud cover (%) | 0 | 100 |
| Temperature (°C) | -50 | 50 |
| Humidity (%) | 0 | 100 |
| Pressure (hPa) | 870 | 1085 |

Table 17: Minimum and maximum values used to normalize the datasets

[26] Because sunlight angles are flattened to not allow negative values, 0 would be a more sensible value for the minimum. However, due to an oversight, -90 was used for the models that were trained. This does not affect model performance in a significant or measurable way.

Models are trained with normalized values. A simple min-max-norm is appropriate in this case, as minimum and maximum values are known a priori for all weather parameters. For energy production, the highest measured values were used, but this would not be possible during normal operation of the HEMS as these values would not be known beforehand. The solar installation's rating, if it is known, could be used in this case but energy production might exceed it under rare circumstances, so an appropriate buffer should be added on top of the rating as the maximum. If the rating is not known, a possible upper bound for energy production could be inferred by the HEMS e.g. from historic measurements of radiation at or near the user's location. Table 17 lists the minimum and maximum values that were used to normalize the datasets.

Choices are explored for the following dimensions of the hyperparameter space:

1. The **optimizing algorithm** and corresponding parameters (e.g. step size or **learning rate**).
2. The **loss** function.
3. The number of **epochs** for training.
4. The **size of batches** propagated through the network in each epoch.
5. The **number of layers** and the respective type of each layer.
6. For dense layers, the **number of neurons** in each layer.
7. For dense layers, each layer's **activation function**.

Adam (Adaptive Moment Estimation) is used as the **optimizer** with a **learning rate** between $10^{-3}$ and $10^{-2}$, which trial and error shows to yield the most consistent learning and to avoid convergence to non-global local minima for the datasets.

A mean-squared error loss is used as the **loss function** for all constellations. $R^2$ scores are calculated in parallel to MSE losses for each period.

For all constellations, training and validation scores fully converge after around 1000 to 2000 **epochs** or less, though scores can continue to increase slightly even beyond that number.

Different **batch sizes** achieve different results depending on the training and validation datasets. When training with month-long datasets, there are batch sizes (32 for Wijchen and 16 for Berlin) that yield smooth, stable curves for the training and validation scores (see e.g. figure 33), whereas smaller batches slow down learning and bigger batches result in instability. Additionally, in neither case do the different batch sizes result in significantly different scores. However, when training with just ten days of data, training is much more unstable and inconsistent, and for the same location there could be different batch sizes that achieve the best results. Batches are always shuffled in all cases.

For the **activation functions**, sigmoid and ReLU are explored. Generally, training with sigmoid is found to result in slightly higher scores than ReLU, and more stable,

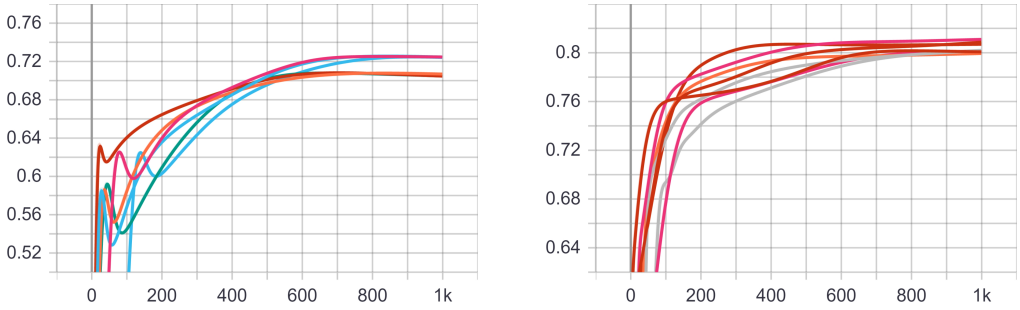



Figure 33: Validation $R^2$ scores over epochs for several neural network training runs with one-month training and ten-day validation datasets without radiation for Wijchen and Berlin

smoother curves without jumps. In addition, there is a smaller variance between different training runs when using sigmoid, and sigmoid is better capable of generalizing as reflected by a smaller difference between training and validation scores than with ReLU.

Appendix B shows an exploration of the model hyperparameter space for all constellations. For each constellation of weather parameters and locations (whereby for each location, data from both stations was used where available), the tables list the best validation $R^2$ scores that have been achieved and the corresponding training scores. Neither the scores nor the number of layers or neurons per layer are to be understood as hard limits, and better scores may be possible after a different number of epochs, with more layers and with a different number of neurons. However, it is found that generally, increasing the **number of layers** beyond the stated offers no or negligible benefits for the datasets in question. Likewise, increasing the **number of neurons per layer** beyond the numbers stated in the tables offers no tangibly better results, and neither does the addition of non-dense layers such as batch normalization.

It becomes clear that there is not a single set of model hyperparameters that achieves the best performance for all datasets. Therefore, a HEMS product should explore hyperparameters before settling on the most suitable choice for the user's environment. The values listed in appendix B provide a general guidance as to what values may be sensible, and suggests that shallow networks with no more than 500 trainable parameters may be sufficient in most if not all cases. The best-performing architectures e.g. when training without radiation have a number of parameters between just 23 and 301. This low number of parameters is also beneficial in terms of computation and memory costs, and therefore also for the cost of hardware that the Model Training Module (3.5.5) and Knowledge Inference Module (3.5.6) are deployed on. Similarly, a lower number of parameters reduces the time cost of training and makes more frequent retraining more economical.

Training and validation scores that are better for the same constellation using a feed-forward neural network compared to using a linear regression model are highlighted in the tables in green. In particular, validation scores for the neural network are consistently

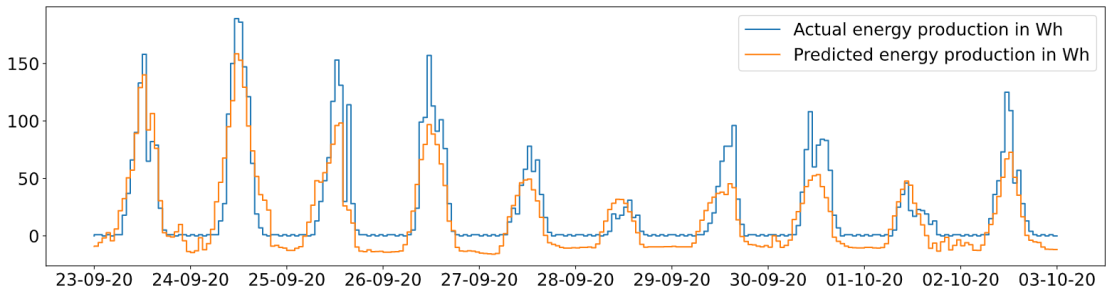



Figure 34: Model evaluation between 23 September and 3 October 2020, using a neural network model trained with sunlight angles and all weather parameters but radiation from Volkel and Deelen between 18 August and 28 August 2020

better or roughly as good when using all weather parameters except for radiation, but not when including radiation or when training exclusively with it. At the same time, models trained with all weather parameters except radiation do not perform better than linear regression models trained with radiation. These observations reinforce previous conclusions: Global radiation, if it is available, exhibits the highest correlation to energy production. However, if it is not available, alternative weather parameters can serve to predict the energy production if non-linear correlations are taken into account by the model. Figure 34 shows an exemplary prediction of energy production using a model that was trained with a one-month dataset without radiation data.

Tables 18 and 19 list scores, hyperparameters and architectures when training with a one-month and a ten-day dataset without radiation, respectively, and validating with datasets as in table 15 and 14, respectively.

| | Score | Architecture |
|---|---|---|
| Wijchen | 0.7047-0.7294 (validation)<br>0.7634-0.7780 (training) | Dense layer (5-16 neurons, sigmoid)<br>Dense layer (1 neuron, linear)<br>56 - 177 parameters<br>Batch size 32<br>1000 epochs |
| Berlin | 0.7994-0.8087 (validation)<br>0.5078-0.6487 (training) | Dense layer (5-16 neurons, sigmoid)<br>Dense layer (1 neuron, linear)<br>56 - 177 parameters<br>Batch size 16<br>1000 epochs |

Table 18: Hyperparameter exploration for neural network training with a one-month training and a ten-day validation dataset without radiation as in table 15

| | Score | Architecture |
|---|---|---|
| Wijchen 1 | 0.7660-0.7776 (validation)<br>0.7382-0.7928 (training) | Dense layer (12-20 neurons, sigmoid)<br>Dense layer (1 neuron, linear)<br>133 - 221 parameters<br>Batch size 18<br>1500 epochs |
| Wijchen 2 | 0.6025-0.6816 (validation)<br>0.6914-0.7489 (training) | Dense layer (2-20 neurons, sigmoid)<br>Dense layer (1 neuron, linear)<br>23 - 221 parameters<br>Batch size 18<br>2000 epochs |
| Berlin 1 | 0.6438-0.7338 (validation)<br>0.3809-0.6545 (training) | Dense layer (16-20 neurons, sigmoid)<br>Dense layer (5-10 neurons, sigmoid)<br>Dense layer (1 neuron, linear)<br>177 - 301 parameters<br>Batch size 18<br>1000 epochs |
| Berlin 2 | 0.6971-0.7685 (validation)<br>0.5883-0.6767 (training) | Dense layer (5-10 neurons, sigmoid)<br>Dense layer (1 neuron, linear)<br>56 - 111 parameters<br>Batch size 32<br>2000 epochs |

Table 19: Hyperparameter exploration for neural network training with a ten-day dataset without radiation and validating with the datasets of table 14

## 5.3. Executive summary

When performing linear regression over global radiation, models can achieve validation $R^2$ scores of over 0.9, as is the case for the Wijchen datasets regardless of whether the training time frame is ten days or a month. Even for the Berlin location, where global radiation correlates less strongly with the measured energy production, linear regression over global radiation can achieve validation $R^2$ scores of up to 0.85.

When global radiation data is unavailable, non-linear models perform best. In this case, validation $R^2$ scores can go as high as 0.8 like they do for the Berlin dataset with a one-month training time frame, and are generally found to be in a range between about 0.7 and 0.8. While these too are good scores and models can perform better, they can potentially also perform worse. A wide range of factors can affect model performance, including solar installation hardware, weather stations, locations and time frames. The past sections demonstrate that **effective models can be found** for the two example locations and the presented exploration of the hyperparameter space. However, it may be found that different, unexplored models (e.g. LSTM-based ones) or different hyperparameters yield better results for other configurations.

**NB**: As can be seen in several figures of the previous sections, model output for all variations can often be negative where it should be zero. These errors are are a special case with two implications: One, that they are easy to catch and the HEMS can simply flatten negative values to zero during inference. Two, that as a result of this easy error correction during inference, the true errors of the models in practice are actually slightly *smaller* than what the scores suggest.

Good energy predictions are the basis of the HEMS' main function of recommending tasks to the user. While the quality of a task recommendation algorithm given a predicted energy budget is independent of the quality of an energy production model, only a good model can enable the HEMS to generate useful results to the user. Equipped with the finding that such models exist, task recommendation algorithms and their performance can be examined.

# 6. Task Recommendations

## 6.1. Tasks and appliance loads

Ogunjuyigbe et al.'s work on an online ad hoc load management scheme for home appliances (see 2.1.2) evaluates said scheme with the energy audit of a "typical 4-bedroom residential building of a medium income earner located in Ibadan, Nigeria [...]"[18] The choice for that building was made because it had "most electrical appliances found in buildings of both the high and low income earners", thus making it representative to a degree.

Table 20 lists a selection of controllable, i.e. schedulable loads under that audit:

| Appliance | Number in use | Rating (W) | Total power[27] (W) | Daily duty cycle (h) |
|---|---|---|---|---|
| Refrigerator | 1 | 400 | 400 | 9 |
| Indoor light | 6 | 15 | 90 | 4 |
| Passageway light | 3 | 15 | 45 | 9 |
| Security light | 5 | 25 | 125 | 6 |
| Ceiling fan | 3 | 100 | 300 | 3 |
| Washing machine | 1 | 650 | 650 | 1 |
| Water pump | 1 | 750 | 375 | 1 |
| Blender | 1 | 300 | 75 | 1 |
| Microwave oven | 1 | 850 | 425 | 1 |
| Water heater | 1 | 800 | 800 | 1 |

Table 20: A selection of schedulable appliance loads [18]

[27]The power rating is what the appliance's manufacturer states as the power input that the appliance is

Not all of these are freely schedulable and they may be constrained in when they can run. E.g., indoor lights and passageway lights would not run during daytime, and a microwave oven or a blender would only run when the user is expected to be in the kitchen. Other appliances like a lawnmower may not be used during nighttime, but this does not require special consideration because the lack of energy production during nighttime would automatically schedule this appliance at daylight.

For further considerations, it is assumed that all tasks are freely schedulable.

## 6.2. Algorithms and heuristics for task allocation

The goal of recommending tasks is to allocate a selection over a given time frame of one week such that energy self-consumption is maximized and grid consumption is minimized.

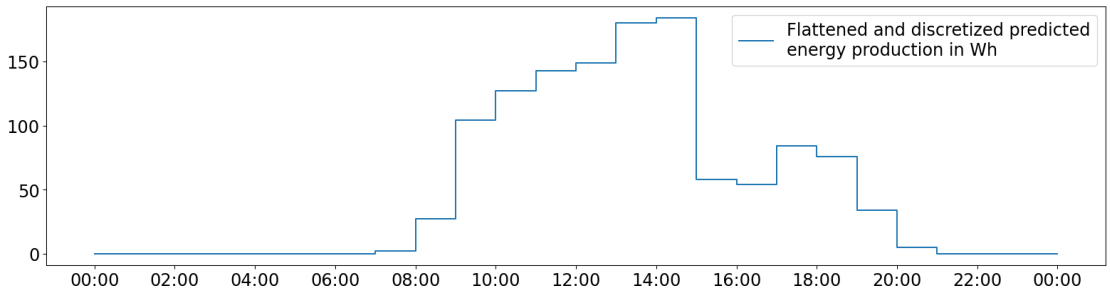



Figure 35: An example of a step function for energy production predictions for one day

Energy production measurements and predictions consist of hourly energy values. Since energy is not a punctual unit but instead describes an integral over time, this means that energy production results in a step function of energy over time, as shown e.g. in figure 35. Energy production drops to zero at night, so it can be seen as one long step function or as a sequence of step functions, with one function for each day.

At the same time, tasks consist of appliances, each with a power rating, running over a certain period of time (3.3.2). While the power rating of an appliance can be updated over time (3.3.3), it remains constant during one task. Therefore, tasks result in constant functions of energy over time, and these functions are shaped like rectangles (or step functions with two steps).

The distribution of tasks can be described as the allocation of the space under the energy production step function with task rectangles. Under this metaphor, the maximization of energy self-consumption means that the total amount of "empty space" under the step function, i.e. space that no task rectangle has been packed in, is minimized. The minimization of grid consumption means that the total space of task rectangles that have been partially or fully placed above the step function is minimized.

designed for. An appliance may tolerate a slightly higher input but it may also function with a lower input. The total power is the actual power that was used to power the appliance in the household, or all appliances of that type. Some appliances can function with a lower power input than stated by the rating, in which case this is reflected by a lower number for the total power.

There are two broad strategies for this, depending on which goal is more strongly prioritized:

1. If minimizing grid consumption is the primary goal, then tasks can be left unallocated even if this means that some produced energy is left unconsumed, as long as it is guaranteed that no grid energy is consumed.

2. If maximizing self-consumption is the primary goal, then as many tasks as necessary and available are allocated to make sure that none of the produced energy is left unconsumed, even at the expense of some tasks possibly consuming grid energy as well.

The former strategy is more straightforward because tasks either fit under the step function or they don't. The latter strategy is more complex because the decision of whether a task is allocated does not only hinge on whether it fits: Since tasks can be allocated even if part of them would be placed outside the step function, additional criteria must be developed for when this is acceptable. This work focuses on the former strategy.

The allocation of tasks can be interpreted as a variation of a two-dimensional bin packing problem with free guillotine cutting and oriented items [35]. Two-dimensional bin packing consists of finding an allocation of rectangular items $j_i$, each with width $w_i$ and height $h_i$, into an unlimited number of finite-size rectangular bins of identical width $W$ and height $H$ such that the items do not overlap and the number of bins is minimized. Similarly, the present problem can be interpreted as a variation of a rectangle packing problem or of a two-dimensional knapsack problem. However, in this variation, the bins are not rectangles but step functions, are not identical, and the number of bins is limited, whereby there is one "bin" (i.e. step function) for each day. An alternative interpretation is that there is one single bin whose height is the maximum energy production and that is filled with "holes".

Due to the different constraints of this variation, heuristics and exact algorithms to solve the regular two-dimensional bin packing, rectangle packing or knapsack problems are not directly applicable. Instead, a different heuristic approach is required.

Energy production consists of one "hill-shaped" step function for each day. While the time is discretized into one-hour steps, prediction values are not discretized if the corresponding model has no integer constraint (as is the case for the models in 5.1 and 5.2). The step function can be simplified by discretizing its values into "levels" of 1 Wh steps, i.e. each level has a height of 1 Wh.

Define the total area $A_i$ of a level $i$ as the area of the rectangle that spans the height of that level and the width of the entire day, minus the area of that rectangle that sits above the step function:

$$A_i = \sum_{t=1}^{24} \delta_i, \text{ where } \delta_i = \left\{ \begin{array}{ll} 1, & \text{if } e(t) \geq i \\ 0, & \text{if } e(t) < i \end{array} \right\} \tag{1}$$

where $e(t)$ is the discretized energy production (i.e. multiple of 1 Wh) between $t-1$ and $t$ hours. An example can be seen in figure 36.

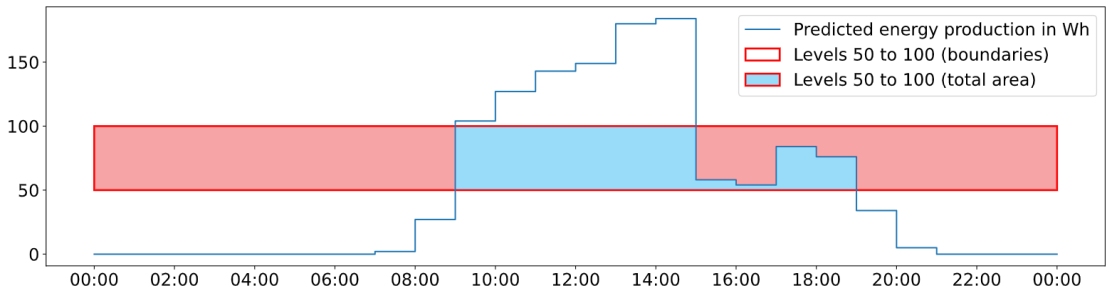



Figure 36: Example for the total area of levels

It then holds that for each level $i$, the total area $A_i$ is no greater than the area $A_{i-1}$ of the previous level $i-1$. In fact, due to the "hill shape" of the step function, each level's total area is more likely to be smaller than that of the previous level than it is to be of equal size.

As a consequence of this, the wider a rectangle is, the less likely it is to fit near the top of the step function. Therefore, rectangles should be allocated closer to the bottom the wider they are, as narrower rectangles are more likely than wider ones to still fit at higher levels. A possible heuristic allocation approach follows from this where:

1. **Wide rectangles** (i.e. long-running tasks) are prioritizedm i.e. rectangles are allocated from the bottom up by **decreasing width**.

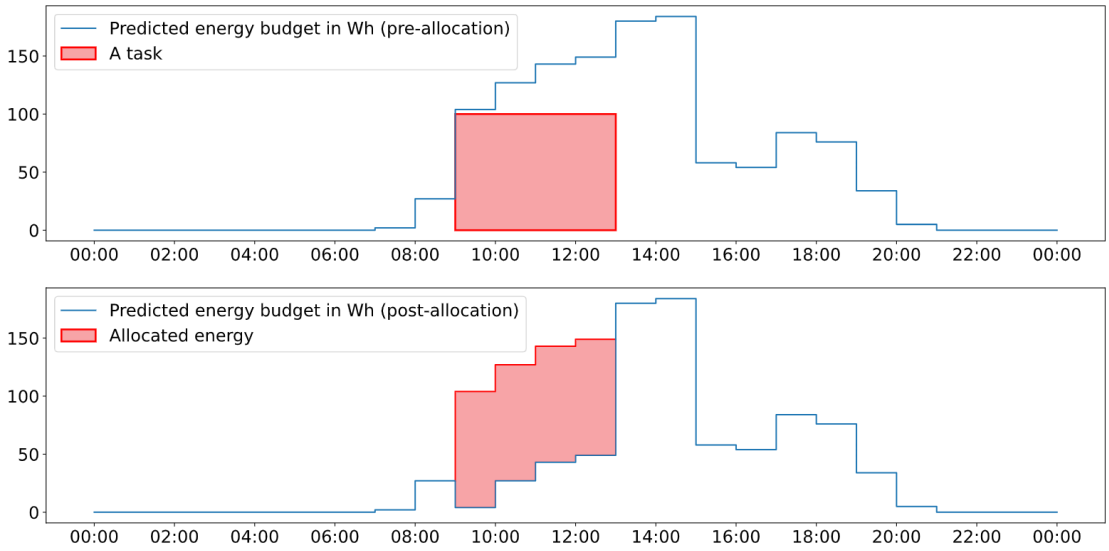



Figure 37: Example for updated step function after allocation

Allocation happens from the bottom up because of the monotonously decreasing total area of levels. However, the exact vertical position of a rectangle is irrelevant because

vertical positions do not translate into any meaning in the problem space of task allocation. Algorithmically, this is reflected as follows: When a rectangle is allocated, its height is subtracted from the step function for the length that it covers, thus resulting in a new step function as shown in figure 37. This reflects the reduced energy budget that is available for the remaining tasks when one is allocated.

Prioritizing wide rectangles comes at the expense of high rectangles, i.e. power-intensive tasks, for which there may not be enough vertical space left. This effect is even more undesired due to the fact that typical schedulable home appliance loads are more likely to be power-intensive than long-running, as shown in 6.1. In this light, it may be desirable to allocate high rectangles as long as sufficient vertical space is still available. Another possible heuristic allocation approach follows:

2. **High rectangles** (i.e. power-intensive tasks) are prioritized, i.e. rectangles are allocated by **decreasing height**.

However, this carries the risk of high rectangles fragmenting the horizontal space and therefore making it harder for wider rectangles to fit.

Prioritizing high rectangles and prioritizing wide rectangles are at odds with each other. A third possible heuristic can attempt to balance these two objectives:

3. **Big rectangles** (i.e. with a large area) are prioritized, i.e. rectangles are allocated by **decreasing area**.

This means that tasks are prioritized according to their total energy consumption. In addition to balancing out allocation of wide and of high rectangles, prioritizing bigger rectangles is sensible because the user is generally more flexible with shorter, less energy-intensive tasks.

The user may want to prioritize some tasks differently so that they would be allocated earlier than they would based on only their height, width or area. This would ensure that some specific tasks always run even if e.g. the energy budget is not high enough for all tasks. Specific use cases for such prioritization are out of the scope, but it would be realized by not sorting rectangles by height, width or area but by a score that those parameters are only one part of.

The hitherto discussed are heuristics for the *order* in which rectangles are allocated. However, another heuristic is required for rectangles' placement, i.e. an *allocation strategy*:

1. **First fit**: Hereby all daily step functions are traversed from left to right (i.e. from earlier to later) and the rectangle is placed on the first horizontal position where it fits (the vertical position is chosen as described above). This can lead to a higher concentration of prioritized rectangles (tasks) in the earlier step functions (days).

2. **Next fit**: Analogous to first fit, but instead of starting the traversal from the first step function, it is started from the last step function to which a rectangle was allocated, thus avoiding a concentration of prioritized rectangles in the first step functions.

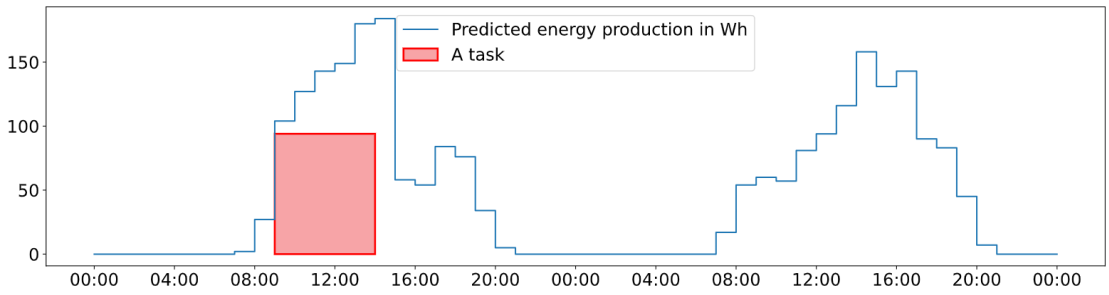



Figure 38: Example for first fit allocation

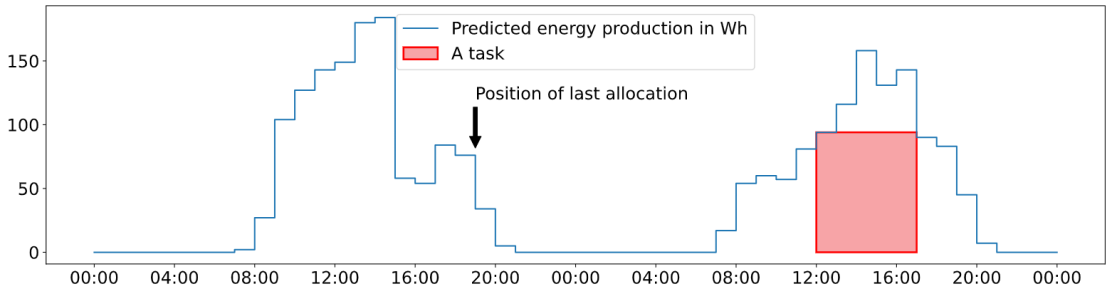



Figure 39: Example for next fit allocation

3. **Best fit**: Rectangles are allocated such that the free space above them is maximized. This minimizes fragmentation of a step function when it is updated after allocation.

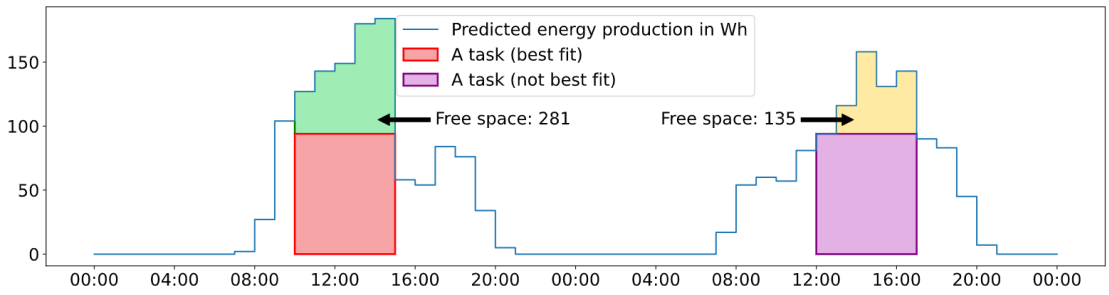



Figure 40: Example for best fit allocation

All strategies are in algorithmic form in appendix C.

The algorithms have a time complexity of $\mathcal{O}(n \times m)$, where $n$ is the number of tasks and $m$ is the number of time points over all step functions. Since tasks are always recommended for a week, there are always $7 \times 24$ time points, and therefore the complexity is $\mathcal{O}(n)$ for this application.

User-defined tasks or tasks spawned by an automation profile (3.3.2) are not allocated through these algorithms but are instead allocated a priori. To reflect this, their required

energy is subtracted from the energy step function for the times they are allocated at, analogously to how this is done for tasks during the algorithms.

## 6.3. Performance analysis

A way to evaluate these heuristics and strategies and analyze their performance is by trying them out and measuring their performance for some step function (or some seven step functions) and some set of tasks. A combination of task allocation order heuristic and allocation strategy is maximally effective when as many tasks as possible were allocated, and minimally effective when no tasks could be allocated at all.

If the total area of all task rectangles is smaller than the total area of the step function, then in the best case all task rectangles could be allocated. However, if the total area of all task rectangles exceeds that of the step function, then even in the best case some tasks would remain unallocated, but there would be no free space remaining in the step function.

From this, two possible evaluation metrics of efficacy follow. In the first case, where all task rectangles could fit under the step function, one may define a metric $m_{t<s}$ such that

$$m_{t<s} = \left\{ \begin{array}{ll} 1, & \text{if all tasks were allocated} \\ 0, & \text{if no tasks were allocated} \end{array} \right\} \tag{2}$$

Define $t$ as the **total area of all tasks**, and $s$ and $\tilde{s}$ as the **total area of the step function before and after allocation**, respectively. Then if all tasks were allocated, it holds that $s - \tilde{s} = t$, for which $m_{t<s}$ should be 1, while if no tasks were allocated, it holds that $s - \tilde{s} = 0$, for which $m_{t<s}$ should be 0. It follows that a fitting metric is

$$m_{t<s} = \frac{s - \tilde{s}}{t} \tag{3}$$

In the second case, where not all task rectangles could fit under the step function, one may define a metric $m_{t>s}$ such that

$$m_{t>s} = \left\{ \begin{array}{ll} 1, & \text{if the step function after allocation is zero} \\ 0, & \text{if the step function after allocation is unchanged} \end{array} \right\} \tag{4}$$

Then in the first case, it holds that $\tilde{s} = 0 \implies s - \tilde{s} = s$, while in the second case, it holds that $\tilde{s} = s \implies s - \tilde{s} = 0$. It follows that a fitting metric is

$$m_{t>s} = \frac{s - \tilde{s}}{s} \tag{5}$$

This metric can also be interpreted such that a score of $x$ means that $x\%$ of the total area of the step function could be allocated, as $s - \tilde{s}$ is the allocated area.

In the rare case that $t = s$, i.e. if the total area of all tasks is equal to the area of the step function before allocation, then both metrics apply, as then in the best case all tasks would be allocated and the step function would be zero after allocation. Analogously, both worst cases would apply.

A generalized metric of efficacy $m$ follows:

$$m = \frac{s - \tilde{s}}{\min\{s, t\}} \tag{6}$$

The more often a heuristic and strategy combination is evaluated, the more conclusive the aggregate of those evaluations can be. To achieve this, one can generate both a random step function and a set of random tasks for each evaluation. Random step functions and tasks should approximate their "real-world" occurrences in this domain.

The "hill-shaped" daily step functions of energy production have pronounced peaks and fall roughly symmetrically to both sides of the peak. The step functions are not monotonically increasing up to the peak (and not monotonically decreasing thereafter), which can be modelled with random noise.

The following sampling method is simple and yields plausible results:

1. The step function has 24 indices, each index $i$ representing energy production from hour $i - 1$ to $i$.

2. A value for the peak is uniformly sampled from an interval, e.g. between 100 and 400. The location of the peak is sampled between indices/hours 13 and 15.

3. The step function is assumed to be 0 before and after certain times, e.g. before 9 and after 21.

4. Nonzero values left/right from the peak are sampled from normal distributions with linearly increasing/decreasing means and standard deviations, the means ranging from 0 to the peak value. (Standard deviations can range from e.g. 14 to 56.)

5. All values are rounded and thus discretized, and negative values are flattened at 0.

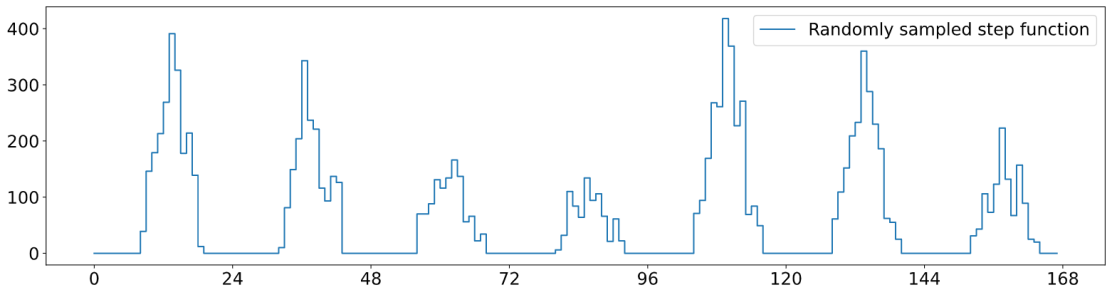



Figure 41: Example for a seven-day step function sampled with the described method

Sampling a seven-day step function is equivalent to sampling seven one-day step functions as described above. Figure 41 shows an example for a seven-day step function sampled with the described method.

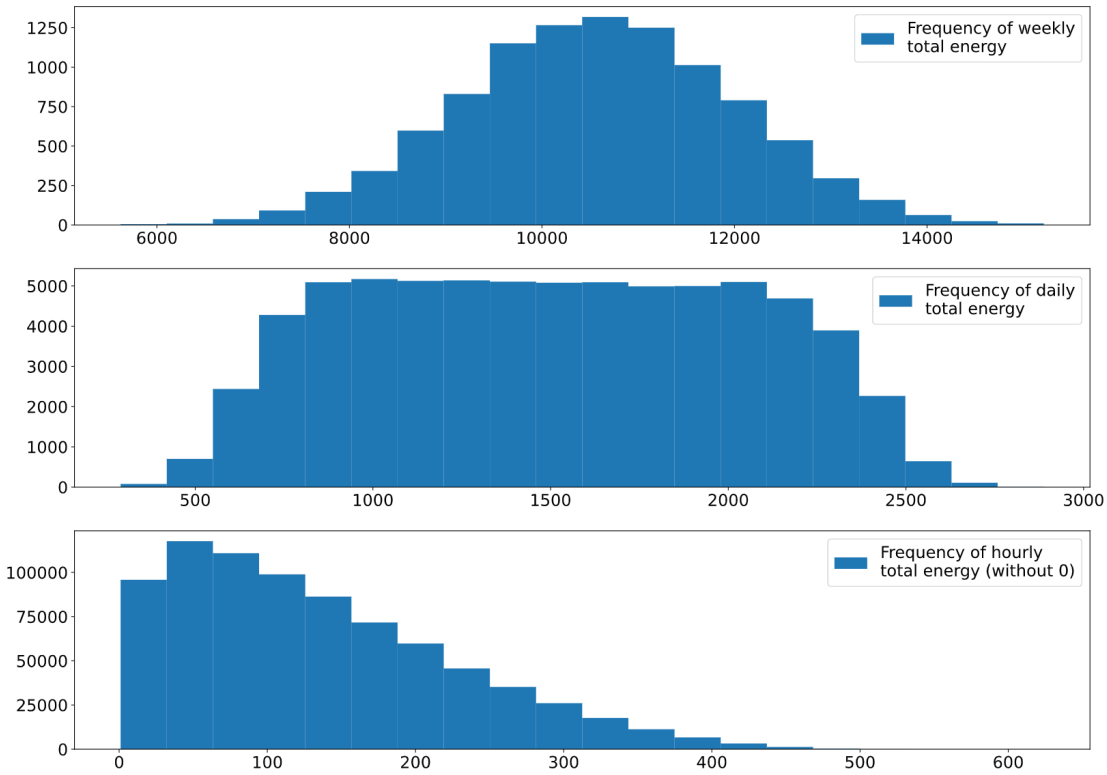



Figure 42: Histogram of weekly, daily and hourly energy levels for 10000 sampled step functions (peak between 100 and 400 at index 14)

Figure 42 shows a histogram of weekly, daily and hourly energy levels for 10000 sampled step functions where the peak is between 100 and 400 at index 14. The distribution of weekly energy levels in particular makes it possible to determine whether the total area of a set of random tasks is expected to exceed or fall short of the area of the step function.

The rectangular shape of tasks makes for straightforward sampling rules:

1. Task widths (i.e. appliance duty cycles) are sampled from a normal distribution with mean 1 and standard deviation 2.5. This ensures that shorter tasks are more likely and that 99.73% of tasks have a duty cycle of up to 8.5 hours. All values are rounded and values under tasks' minimum duration of 1 are discarded.

2. Task heights (i.e. appliance power ratings) are sampled from a normal distribution, the mean and standard deviation of which can be tailored so that the total area of all sampled tasks is expected to either exceed or fall short of the step function's total area.

For all further considerations, step functions are assumed to be sampled with the example values for peaks, means and standard deviations. Under this assumption, the

mean total area of step functions sampled this way was found in testing to be around **10420** over 10000 samples. The mean step height (i.e. hourly energy) was around **32** and the maximum step height was **624**. The maximum in particular is important for the parameter choice of task sampling because tasks can never be allocated if their height exceeds the maximum step height, so the distribution of tasks should be such that such tasks become unlikely.

Table 21 shows some parameters for mean and standard deviation of task heights and the mean total task areas they result in, determined by sampling 50000 sets of tasks.

| Number of tasks | Mean (task height) | Standard deviation (task height) | Mean total task area | Proportion of step function area |
|---|---|---|---|---|
| 50 | 50 | 130 | 15744 | 1.5109 |
| 20 | 20 | 100 | 4126 | 0.3960 |

Table 21: Example parameters for random task sampling

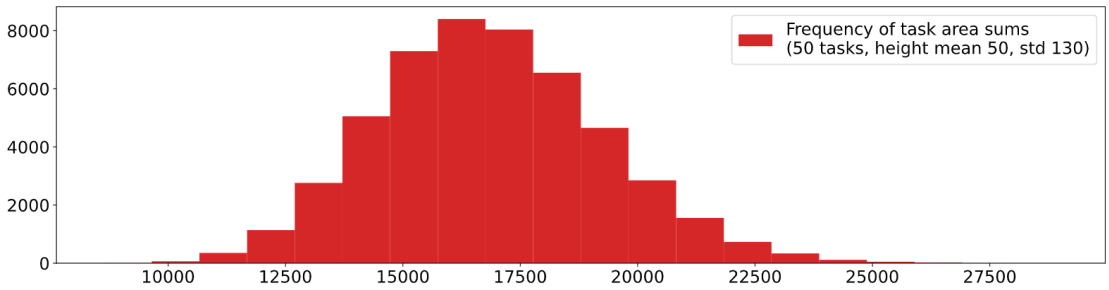



Figure 43: Frequency of task area sums with parameters such that the total area of all tasks is expected to exceed that of the step function

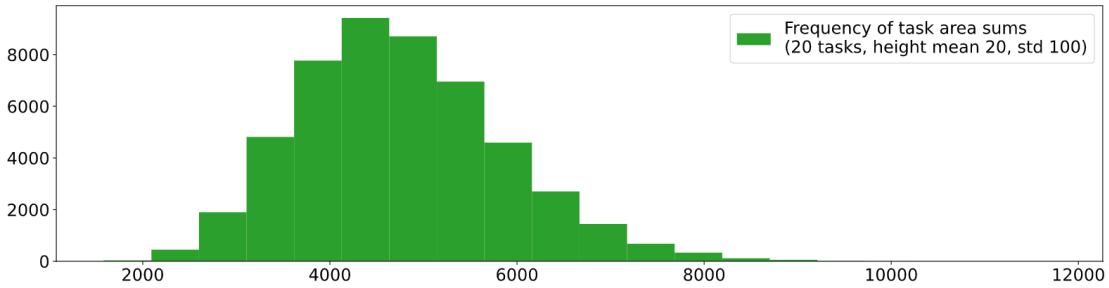



Figure 44: Frequency of task area sums with parameters such that all tasks are expected to fit under the step function

Figures 43 and 44 show the frequency of task area sums when using the parameters for which the total area of all tasks is expected to exceed or fall below that of the step function, respectively.

Figure 45 shows the frequency of efficacy scores when the total area of all tasks is expected to exceed that of the step function, for all combinations of task sort orders and allocation strategies. The score frequencies follow roughly normal distributions with pronounced means, which are listed in table 22 for all combinations. Sorting by descending width yields the best results by a small margin but notably, none of the allocation strategies performs better than the others.

| | First fit | Next fit | Best fit |
|---|---|---|---|
| Descending width | 0.6259 | 0.6040 | 0.6061 |
| Descending height | 0.5652 | 0.5766 | 0.5801 |
| Descending area | 0.5536 | 0.5497 | 0.5331 |

Table 22: Mean efficacy scores when the total task area is expected to exceed the step function area

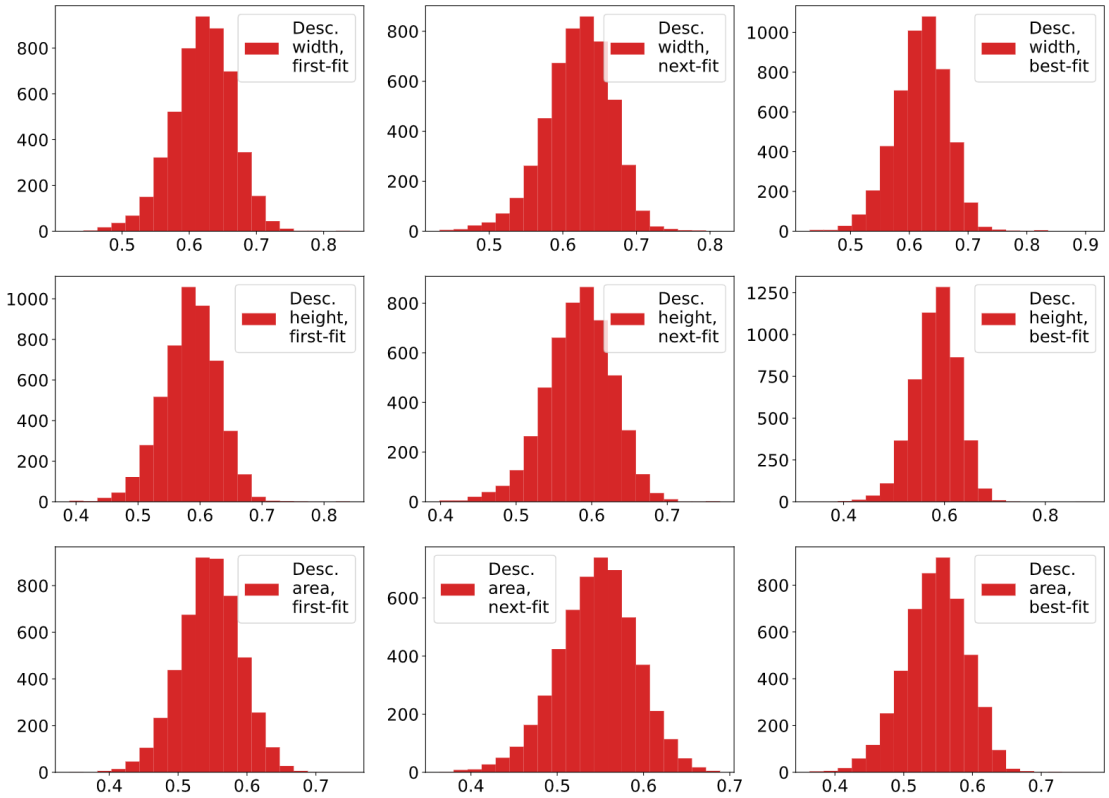



Figure 45: Frequency of efficacy scores when the total task area is expected to exceed the step function area

However, all the allocation strategies are fundamentally restricted by the requirement that tasks fit precisely in order to be allocated. The more energy-intensive the tasks are

expected to be and thus the bigger the rectangles, the more likely there are tasks that are limited in this way. In particular, this applies to tasks with a high power rating: Due to the "hill shape" and the monotonically decreasing total area of step function levels, the less likely a higher rectangle is to fit.

A mitigating measure would be allowing tasks to "stretch" over time, whereby only for allocation purposes, a task's rectangle is stretched or split into consecutive smaller polygons with the same total area, as shown in figure 46. This would presuppose the presence of a *buffer battery* because then the appliance is consuming energy that is available over some time frame that is longer than the time frame during which the task actually runs. The size of the battery would determine to which degree task stretching may be carried out. In turn, the choice of battery size could be informed by determining which amount of stretching yields the best allocation results, although further examinations to this regard are outside the scope of this work.

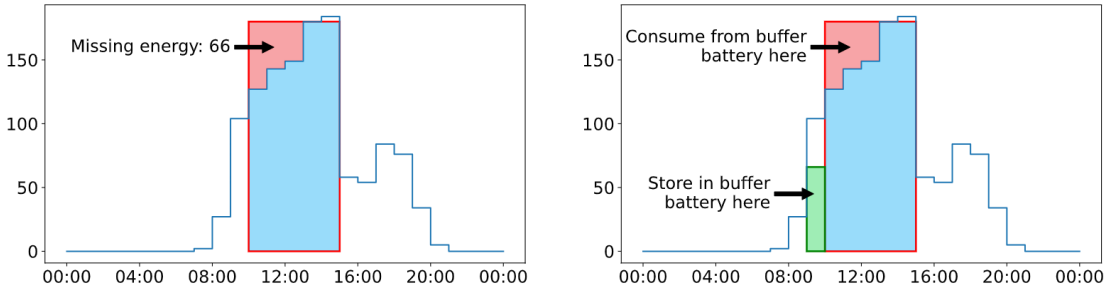



Figure 46: Example for task "stretching" over time

Figure 47 shows the frequency of efficacy scores when the total area of all tasks is expected to fall below that of the step function. Some observations from before are confirmed, such as which combinations perform best. However, in this case the score frequencies do not follow a normal distribution. It can be expected that all or practically all tasks can be allocated, especially when sorting by descending width, as shown in table 23, and high scores are achieved even when this is not the case.

| | First fit | Next fit | Best fit |
|---|---|---|---|
| Descending width | 0.9625 | 0.9637 | 0.9603 |
| Descending height | 0.9617 | 0.9610 | 0.9630 |
| Descending area | 0.9612 | 0.9642 | 0.9627 |

Table 23: Most frequent efficacy scores when the total task area is expected to fall below the step function area

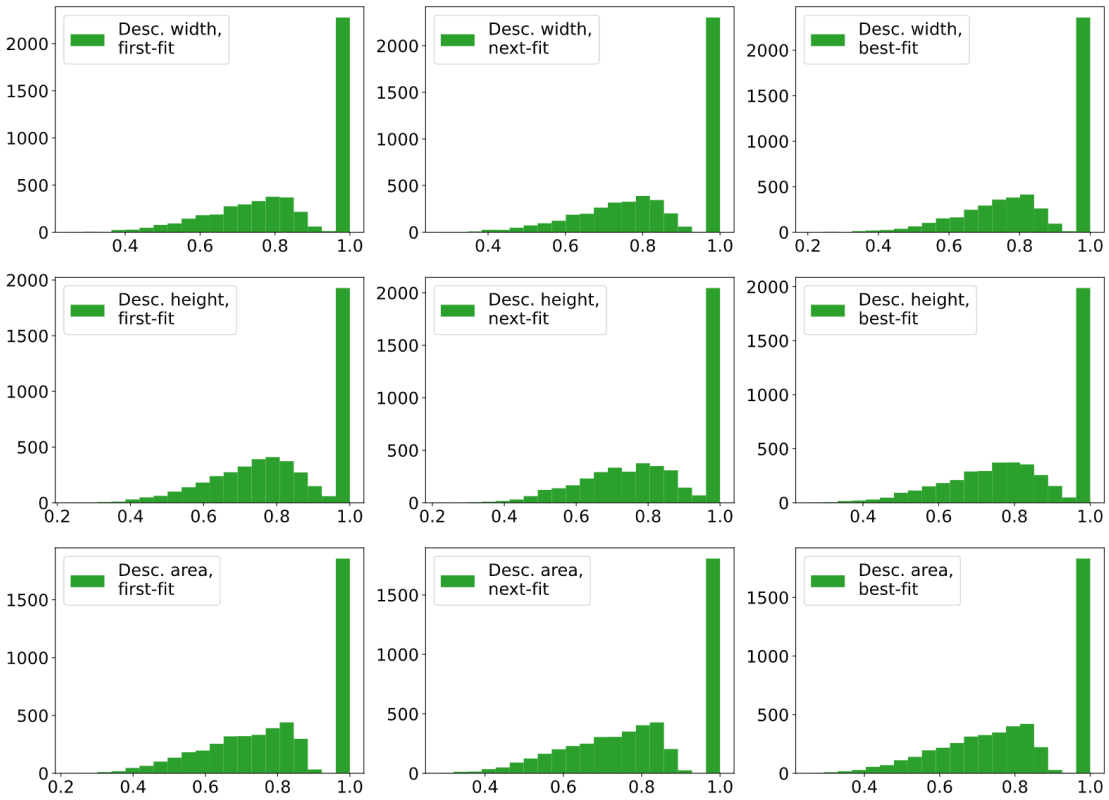



Figure 47: Frequency of efficacy scores when the total task area is expected to fall below the step function area

## 6.4. Executive summary

The allocation of tasks can reach near-perfect and perfect efficacy if sufficient energy for all appliances is expected to be available, meaning that one can expect almost all or all tasks to be allocated. If total appliance energy consumption is expected to exceed total available energy by as much as 50%, efficacy can reach up to about 0.63, meaning that 63% of the total area of the step function can be expected to be allocated.

These scores highlight both the potential and the limitations of the discussed approaches for task allocation. Refinements such as task "stretching" may improve on these scores. The scores also suggest that in general, better task allocation efficacy can be achieved and expected the greater the solar installation's performance and expected energy output is in relation to the tasks' total energy requirements. Users therefore benefit threefold from more powerful PV systems:

1. The bigger energy budget generally lowers the user's requirement for energy from the grid - independently of whether a HEMS is deployed.

2. The potentially much better task allocation efficacy enables the user to consume energy **in a maximally targeted way**.

3. A near-perfect and perfect task allocation efficacy renders additional measures such as task stretching superfluous, thus also saving on buffer battery size that this approach would require.

However, even in cases where tasks' energy requirements exceed the budget and despite the room for improved efficacy in this case, the examined approach enables a targeted scheduling of tasks. Therefore, the HEMS can provide value in all cases, regardless of the energy budget.

# 7. Implementation and Prototype Software

The HEMS has been partly implemented in the context of this thesis. This partial prototype implementation consists of a C++ HEMS "core" program for Linux-based operating systems and a set of accompanying Python-based tools.

## 7.1. Core program

The C++-based core portion of the HEMS program targets two platforms:

1. x64 and x86 processors,
2. A FRITZ!Box 7530 with an ARM-based Qualcomm 'Dakota' SoC (717MHz Quad Core ARM Cortex A7 WiSoC + 2x 2x2 MAC/Baseband/Radio).[28]

x64 and x86 binaries can be built using the standard GCC compilation toolchain through GNU Make, while ARM binaries for the FRITZ!Box 7530 target require a proprietary custom build toolchain from AVM. C++ 14 was chosen as the language for implementation in order to support both these platforms and the FRITZ!Box platform in particular using a common codebase.

Both builds have the Boost C++ library[29] as the only prerequisite, whereby builds were produced and tested with Boost version 1.73.0. For x64 and x86 targets, Boost can be installed through standard channels, e.g. as pre-built library files via

```
sudo apt install libboost-all-dev
```

or

```
sudo apt install libboost1.73-all-dev
```

on Debian-based distributions or by building from the source available at [30] or [31] for version 1.73.0. For the ARM platform target, Boost must be built from source with the custom toolchain from AVM.

[28] https://boxmatrix.info/wiki/FRITZ!Box_7530
[29] https://www.boost.org/
[30] https://www.boost.org/users/download/
[31] https://dl.bintray.com/boostorg/release/1.73.0/source/

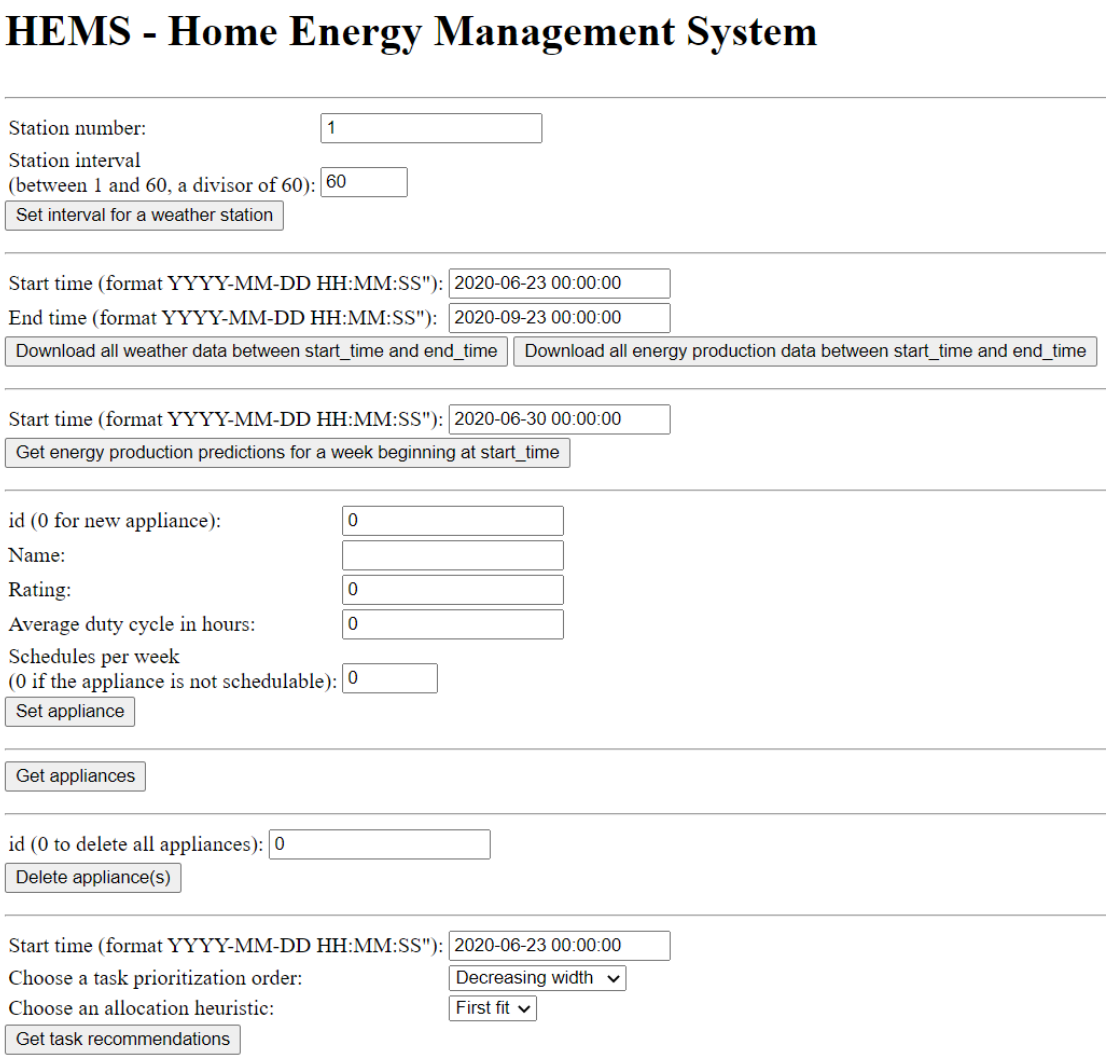



Figure 48: The web-based user interface of the partial HEMS implementation

Included in the core program of this partial implementation is the following:

1. All data types as discussed in section 3.3, including string representations, serialization and ==/!= operators.

2. A *messenger* class shared by all modules that implements communication between modules in a unified manner. Messages follow a standardized format and are exchanged in serializable form through POSIX message queues. They can be asynchronous (*COMMAND* messages) or synchronous (*REQUEST*), in which case a response (*RESPONSE*) is expected. Message and response payloads are shared between modules through shared memory objects. The messenger also implements the settings mechanism discussed in 3.3.5: New settings are sent to all modules

to check, which accept or reject them individually. If all modules accept, the new settings are sent to all modules to commit.

3. A fully functional **Launcher Module** as in section 3.5.1 that can be started via command line. It establishes the common communication infrastructure of modules via POSIX message queues[32], launches the modules in separate processes and watches their current execution status. Module paths can be provided as command line arguments to the launcher. All output generated by the modules is sent to the launcher, which prints it to the console and optionally to a log file.

4. A partially implemented **User Interface Module** that launches an HTTP server and enables user interaction with the HEMS through a web-based interface. Supported interaction includes the following:
   a) Requesting the HEMS to download all weather or energy production data between given start and end times from respective external HTTP servers (see 7.2).
   b) Requesting the HEMS to generate and show a textual view of energy production predictions for a week beginning at a given start time.
   c) Adding new appliances or modifying existing ones.
   d) Getting a textual view of all existing appliances.
   e) Deleting appliances.
   f) Requesting the HEMS to generate and show a textual view of task recommendations for a week beginning at a given start time, choosing a task prioritization order and an allocation heuristic as discussed in 6.2.

5. A partially implemented **Measurement Collection Module** as in section 3.5.3 that supports collecting energy production and weather data from respective external HTTP servers (see 7.2) at user request through the user interface.

6. A partially implemented **Automation and Recommendation Module** as in section 3.5.4 that supports generating task recommendations at user request through the user interface. All task prioritization orders and allocation heuristics discussed in 6.2 are supported.

7. A functional **Knowledge Inference Module** as in section 3.5.6 that can request model inference from an external HTTP server that interfaces with the model (see 7.2).

8. A functional **Data Storage Module** as in section 3.5.7 that creates an SQLite database with the scheme as in 3.4 and supports:
   a) Adding new entries or modifying existing ones for all data types discussed in 3.3.

[32] `https://man7.org/linux/man-pages/man7/mq_overview.7.html`

b) Deleting entries for all data types discussed in 3.3.

c) Getting existing entries for settings, appliances, energy production and weather data.

This core program covers the following program sequences from 3.6:

1. **Session initialization** (3.6.1) is fully implemented.
2. **Getting recommendations** (3.6.2) is implemented, except for committing the tasks that the user chooses and without considering already scheduled tasks for the recommendation.
3. **Setting appliances** (3.6.3) is fully implemented.
4. **Collection of energy production data** (3.6.6) is fully implemented.
5. **Collection of weather data** (3.6.7) is fully implemented.

## 7.2. Tools

In addition to the core program, a number of tools based on Python 3.8 has been implemented in the context of the thesis. These tools are the following:

1. A small tool to collect energy production measurements from a solar installation with a FRITZ!DECT 200 (see 4.1.1).
2. A program that loads the energy production model and offers an HTTP interface for the HEMS core program to infer from it. The model can be a TensorFlow model file for a feed-forward neural network (see 5.2), a Python pickle file for a linear regression model (see 5.1) or a CSV file defining an artificial step function to test the HEMS and task recommendations.
3. A program that offers an HTTP interface for the HEMS core program to download weather data. Weather data is loaded from a NumPy array file for a given time, if available.[33]
4. A program that offers an HTTP interface for the HEMS core program to download energy production data. This data has been collected in advance from the energy collector and is read from CSV files.
5. A program to test the task allocation heuristics and algorithms with real energy production data or with artificial step functions as described in 6.3.
6. A program to generate artificial step functions and save them into a CSV file to use with the energy production model interface.

[33] In order to comply with Kachelmannwetter's terms of use, no code to obtain weather data from the site is made available in the context of this thesis.

These tools may require additional Python modules. Each tool lists its dependencies in a corresponding *requirements.txt* file, which can be installed via

```
pip3 install -r requirements.txt
```

## 7.3. Project structure

The partial implementation of the HEMS is available in the thesis' GitHub repository at `https://github.com/adrianghc/HEMS`, where all code and data produced in the context of this thesis is hosted.

The project contains workspace files for Visual Studio Code, including convenience functions like launch configurations (for debugging of HEMS modules, for all tools, and for tests) and tasks for building and tests. While it is recommended, Visual Studio Code is not in any way required to view, modify, build or run any parts of the project.

The directory structure of the project is as follows:

1. **data**: This directory contains the raw energy production data that was collected from the Wijchen and Berlin locations with the energy collector.[34]
2. **models**: This contains two well-performing example models:
   a) A TensorFlow-based neural network model. This model was trained with 30 days of weather data (without radiation) from Wijchen on a three-layer architecture, including an input layer, a 16-neuron dense layer with sigmoid activations and an output layer with linear activations. The model was trained on 1000 epochs using the Adam optimizer with a learning rate of $10^{-3}$.
   b) A linear regression model trained with 30 days of weather data (including radiation) from Wijchen.

   The directory also includes a template CSV file for artificial step functions, and Jupyter notebooks for the training of neural network and linear regression models.
3. **src**:
   a) **/hems**: The entire C++ source code for the HEMS core program.
   b) **/sqlite**: The SQLite source code.
4. **include**:
   a) **/hems**: This includes all C++ header files that are part of the HEMS core program.
   b) **/extras**: Contains third-party code used in the HEMS core program.
   c) **/sqlite**: Includes headers for SQLite.

[34] Weather data that was used in the context of this thesis is not included in order to comply with Kachelmannwetter's terms of use. If you are interested in reproducing the results of this thesis that presume the availability of weather data, please write a free-form email to adrian[dot]ghc[at]outlook[dot]com including [MASTERS-THESIS] in the subject.

5. **resources**: This contains resources for the web-based user interface.
6. **tests**: This directory contains unit tests for the HEMS core program. These tests cover typical and edge cases for a number of scenarios, including:
   a) Data types, the messenger class and the settings mechanism.
   b) **Launcher Module**: The management of POSIX message queues and watching modules' execution status.
   c) **Measurement Collection Module**: Energy production and weather data download.
   d) **Data Storage Module**: Creating the database, checking and committing new settings, setting entries for all data types, deleting entries for all data types, and getting entries for settings, appliances, energy production and weather data.
7. **doc**: This thesis manuscript.

In order to build the HEMS core program for an x86 or x64 platform, simply run

```
make
```

from the project's root directory (build targets for individual modules are also available). This generates binaries for all modules in the *build/native/hems* directory, where the HEMS can be launched by running

```
./launcher
```

Each of the tools can be started by running

```
./run.sh
```

from its corresponding directory.

All files and contents of the project are made available under The MIT License, unless explicitly stated otherwise on third-party code files, and except for this thesis manuscript.

# 8. Summary

## 8.1. Conclusion

The HEMS enables users to schedule appliances with the goal of increasing energy self-consumption, based on predictions of energy production. These predictions can achieve $R^2$ scores of over 0.9 when using global radiation data, or up to 0.8 when global radiation data is not available. The allocation of tasks can reach near-perfect and perfect efficacy if sufficient energy for all appliances is expected to be available, and an efficacy of up to about 63% allocated energy if total task energy consumption is expected to be as high as 150% of the total available energy budget.

Based on these results, can it be said **if a battery-less HEMS appears feasible or promising**, particularly in the geographical regions for which data was collected?

A quantitative answer to this question is difficult. Generally, the performance of the HEMS depends on a very wide range of factors: Environmental factors such as the geographical location and surrounding objects determine the maximum amount of energy that can be produced. The solar installation hardware itself affects how much of this potential energy can be harnessed at any given moment. Weather conditions can limit energy production only slightly or very severely, and with great variability. And finally, whether a given energy budget can be used to maximal effectiveness depends on the specific appliances that users wants to schedule. All these factors span a vast array of dimensions over which performance is affected.

Additionally, the factors for which performance can be and have been quantitatively measured in a general way - namely energy production predictions (section 5) and task scheduling algorithms (section 6) - are independent of one another. Therefore, performance of one does not affect performance of the other in any way, as the task allocation algorithm - the part of the HEMS that produces the output to the user - is not concerned with whether the predicted energy budget it receives as input is close to the actual energy budget or not. Consequently, the overall performance of the HEMS cannot be quantitatively evaluated generally but only on a case-by-case basis. Nonetheless, a general answer to the question of feasibility can be sought.

When power is consumed from a solar installation without a management system, there is only one metric that is being optimized, which is user comfort based on the ability to instantaneously meet energy demand whenever it arises. However, this comes at the expense of a full lack of optimization for maximum self-consumption, minimum grid consumption and minimum grid exports. The only way to achieve these sans task scheduling is through the addition of a battery, which in turn comes at the expense of a lack of optimization for minimum battery size.

The HEMS contributes a way for users to schedule appliances, and thus consume energy, **in a targeted way** that aims to optimize the stated goals, *including optimizing for minimum battery size*, provided that the user can accept a loss in flexibility in when appliances are used. The previous sections of this thesis demonstrate effectiveness of both energy production predictions and task allocation, both of which form the basis for overall effectiveness.

These findings cannot conclude definitively whether a battery-less HEMS is feasible or promising under any and all circumstances. They do demonstrate, however, that **effectiveness of the HEMS can be plausibly expected** under a range of circumstances, including greater or smaller correlation between energy production and radiation (affecting model performance) and greater or smaller energy requirements for appliances in relation to the energy budget (affecting task allocation effectiveness). It is therefore concluded that **a battery-less HEMS as explored in this thesis can, indeed, be feasible**, even for regions such as the ones studied - recognizing that performance can fall below the demonstrated, but also exceed it. The demonstrated software architecture and prototype implementation offer a workable basis on which to further develop a HEMS, and evaluate and experiment with the suggested models and allocation algorithms for

other locations and datasets.

## 8.2. Outlook and future work

Despite these results, it is clear that there remains room for improvement on all fronts even while maintaining most of the concepts that were discussed in this thesis. Energy production predictions may improve with additional weather parameters, or with architectures such as LSTMs that infer from time series instead of only singular points, which may allow networks to detect periodic signals or detect changes to the development of energy production in advance. The allocation of tasks may increase in efficacy through more complex strategies and heuristics, such as through the method of task "stretching".

A greater emphasis on the existence of a buffer battery and possible implications thereof may also be explored, including but not limited to the interaction between task "stretching" and buffer battery size. The functionality of the HEMS may be expanded beyond recommending tasks to the recommendation of battery sizes as well, enabled by machine learning using energy production and energy consumption measurements over time.

Some greater deviations from the concepts that have been discussed in this thesis may also be worth exploring. For instance, energy prices acquired in advance and in real-time via smart grid may enable other allocation strategies that look beyond maximizing self-consumption or minimizing grid consumption as objectives to include pricing as an additional factor. Additionally, better allocation efficacy may be achieved through the enhanced flexibility that appliances with non-binary automation can provide, i.e. appliances that can not only be turned on or off but can also consume more or less energy, such as a fridge or freezer that can regulate its temperature. Supporting such appliances would greatly enhance the complexity of both task allocation and appliance automation.

Finally, while the application and the concepts of this thesis have been discussed in the context of a home environment and a residential solar installation, they may be beneficially applied to other settings as well, such as industrial environments.

## References


[1] Global Market Outlook for Solar Power 2016-2020 - Solar Power Europe (SPE) (https://www.webcitation.org/6nRc4lsKn?url=http://www.solareb2b.it/wp-content/uploads/2016/06/SPE_GMO2016_full_version.pdf)

[2] Global Market Outlook for Solar Power 2015-2019 - Solar Power Europe (SPE) (https://www.webcitation.org/6ZA1o2aLo?url=http://www.solarpowereurope.org/fileadmin/user_upload/documents/Publications/Global_Market_Outlook_2015_-2019_lr_v23.pdf)

[3] Global Market Outlook for Photovoltaics 2014-2018 - European Photovoltaic Industry Association (EPIA) (https://www.webcitation.org/6QGSvAF7w?url=http://www.epia.org/fileadmin/user_upload/Publications/EPIA_Global_Market_Outlook_for_Photovoltaics_2014-2018_-_Medium_Res.pdf)

[4] A Snapshot of Global PV (1992-2014) - European Photovoltaic Industry Association (EPIA) (https://www.webcitation.org/6XPpb1fai?url=http://www.iea-pvps.org/index.php?id=92&eID=dam_frontend_push&docID=2430)

[5] The projections for the future and quality in the past of the World Energy Outlook for solar PV and other renewable energy technologies - Matthieu Metayer, Christian Breyer and Hans-Josef Fell, September 2015 (https://web.archive.org/web/20160915150103/http://energywatchgroup.org/wp-content/uploads/2015/09/EWG_WEO-Study_2015.pdf)

[6] How the IEA exaggerates the costs and underestimates the growth of solar power (https://energypost.eu/iea-exaggerates-costs-underestimates-growth-solar-power/)

[7] Why Have IEA Renewables Growth Projections Been So Much Lower Than the Out-Turn? (https://energycentral.com/c/ec/why-have-iea-renewables-growth-projections-been-so-much-lower-out-turn)

[8] The role of residential rooftop photovoltaic in long-term energy and climate scenarios - David E.H.J. Gernaat, Harmen-Sytze de Boer, Louise C. Dammeier, Detlef P. van Vuuren, 1 December 2020 (https://www.sciencedirect.com/science/article/abs/pii/S0306261920312009)

[9] BDEW-Strompreisanalyse Januar 2020 - Bundesverband der Energie- und Wasserwirtschaft e.V., 7 January 2020 (https://www.bdew.de/media/documents/20200107_BDEW-Strompreisanalyse_Januar_2020.pdf)

[10] Stromgestehungskosten erneuerbare Energie - Fraunhofer-Institut für solare Energiesysteme, March 201 (https://www.ise.fraunhofer.de/content/dam/ise/de/documents/publications/studies/DE2018_ISE_Studie_Stromgestehungskosten_Erneuerbare_Energien.pdf)

[11] EEG-Registerdaten und -Fördersätze (https://www.bundesnetzagentur.de/DE/Sachgebiete/ElektrizitaetundGas/Unternehmen_Institutionen/ErneuerbareEnergien/ZahlenDatenInformationen/EEG_Registerdaten/EEG_Registerdaten_node.html)

[12] Archivierte EEG-Vergütungssätze und Datenmeldungen (https://www.bundesnetzagentur.de/DE/Sachgebiete/ElektrizitaetundGas/Unternehmen_Institutionen/ErneuerbareEnergien/ZahlenDatenInformationen/EEG_Registerdaten/ArchivDatenMeldgn/ArchivDatenMeldgn_node.html)

[13] Solarstrom und Energiespeicher - Planung und Kauf einer Photovoltaikanlage - Verbraucherzentrale Nordrhein-Westfalen, September 2019 (https://www.verbraucherzentrale.nrw/sites/default/files/2019-09/brosch%C3%BCre-solaranlage-batteriespeicher-kaufen-kosten-f%C3%B6rderung.pdf)

[14] Artificial Immune System in the Management of Complex Small Scale Cogeneration Systems - Fabio Freschi, Maurizio Repetto, 2009 (https://www.igi-global.com/chapter/artificial-immune-system-management-complex/19643)

[15] A Stochastic Home Energy Management System Considering Satisfaction Cost and Response Fatigue - Miadreza Shafie-Khah, Pierluigi Siano, 19 July 2017 (https://ieeexplore.ieee.org/document/7984899)

[16] Smart home energy management systems: Concept, configurations, and scheduling strategies - Bin Zhou, Wentao Li, Ka Wing Chan, Yijia Cao, Yonghong Kuang, Xi Liu, Xiong Wang, August 2016 (https://www.sciencedirect.com/science/article/abs/pii/S1364032116002823)

[17] An Optimal Power Scheduling Method for Demand Response in Home Energy Management System - Zhuang Zhao, Won Cheol Lee, Yoan Shin, Kyung-Bin Song, 6 June 2013 (https://ieeexplore.ieee.org/document/6525433)

[18] Management of loads in residential buildings installed with PV system under intermittent solar irradiation using mixed integer linear programming - A.S.O. Ogunjuyigbe, T.R. Ayodele, O.E. Oladimeji, 15 October 2016 (https://www.sciencedirect.com/science/article/abs/pii/S0378778816307381)

[19] Solar plus: Optimization of distributed solar PV through battery storage and dispatchable load in residential buildings - Eric O'Shaughnessy, Dylan Cutler, Kristen Ardani, Robert Margolis, 1 March 2018 (https://www.sciencedirect.com/science/article/abs/pii/S0306261917318421)

[20] Photovoltaic self-consumption in buildings: A review - Rasmus Luthander, Joakim Widén, Daniel Nilsson, Jenny Palm, March 2015 (https://www.researchgate.net/publication/270825726_Photovoltaic_self-consumption_in_buildings_A_review)

[21] ENERGY INDEPENDENCE AND SECURITY ACT OF 2007 (https://www.govinfo.gov/content/pkg/PLAW-110publ140/html/PLAW-110publ140.htm)

[22] AVM Home Automation - HTTP Interface, 24 April 2020 (https://avm.de/fileadmin/user_upload/Global/Service/Schnittstellen/AHA-HTTP-Interface.pdf)

[23] How Reliable Are Weather Forecasts? (https://scijinks.gov/forecast-reliability/)

[24] Wetter und Klima - Deutscher Wetterdienst - Glossar - Gesamtbedeckungsgrad (https://www.dwd.de/DE/service/lexikon/Functions/glossar.html?lv2=100932&lv3=101016)

[25] Cloud Cover Forecasting from METEOSAT Data - Francisco Javier Batlles, Joaquín Alonso, Gabriel López, 2013 (https://www.sciencedirect.com/science/article/pii/S1876610214014891)

[26] Wetter und Klima - Deutscher Wetterdienst - Glossar - Globalstrahlung (https://www.dwd.de/DE/service/lexikon/Functions/glossar.html?nn=103346&lv2=100932&lv3=101042)

[27] Wetter und Klima - Deutscher Wetterdienst - Our services - Solar radiation - solar energy (https://www.dwd.de/EN/ourservices/solarenergy/solarenergy.html)

[28] Wetter und Klima - Deutscher Wetterdienst - Our services - Satellite-based retrieval of surface solar radiation (https://www.dwd.de/EN/ourservices/solarenergy/satellite_solarradiation.html?nn=495490&lsbId=570780)

[29] Investigation of the Effect Temperature on Photovoltaic (PV) Panel Output Performance - Amelia Razak, Mohd Irwan Yusoff, Leow Wai Zhe, Muhammad Irwanto, October 2016 (https://www.researchgate.net/publication/309600416_Investigation_of_the_Effect_Temperature_on_Photovoltaic_PV_Panel_Output_Performance)

[30] Performance evaluation of a solar photovoltaic system - Wael Charfi, Monia Chaabane, Hatem Mhiri, Philippe Bournot, November 2018 (https://www.sciencedirect.com/science/article/pii/S2352484717303323)

[31] Evaluation of solar PV panel performance under humid atmosphere - Abhishek Kumar Tripathi, Shashwati Ray, Mangalpady Aruna, Sandeep Prasad, 10 October 2020 (https://www.sciencedirect.com/science/article/pii/S2214785320366591)

[32] Effect of Humidity on the PV Performance in Oman - Miqdam Tariq Chaichan, September 2012 (https://www.researchgate.net/publication/257840932_Effect_of_Humidity_on_the_PV_Performance_in_Oman)

[33] Learning representations by back-propagating errors - David E. Rumelhart, Geoffrey E. Hinton, Ronald J. Williams, October 1986 (https://www.nature.com/articles/323533a0)

[34] Long Short-term Memory - Sepp Hochreiter, Jürgen Schmidhuber, December 1997 (https://www.researchgate.net/publication/13853244_Long_Short-term_Memory)

[35] Heuristic and Metaheuristic Approaches for a Class of Two-Dimensional Bin Packing Problems - Andrea Lodi, Silvano Martello, Daniele Vigo, 1 November 1999 (https://pubsonline.informs.org/doi/abs/10.1287/ijoc.11.4.345)

# A. Weather data sources

| | Volkel[35] | Deelen[36] | Dahlem[37] | Tempelhof[38] |
|---|---|---|---|---|
| Global radiation (kJ/m²) | Yes[39] | Yes | No | Yes[40] |
| Cloud cover (%) | Yes[41] | Yes | Yes[42] | No |
| Temperature (°C) (2m) | Yes[43] | Yes | Yes[44] | Yes |
| Humidity (%) | Yes[45] | Yes | Yes[46] | Yes |
| Pressure (hPa) (station altitude) | Yes[47] | Yes | Yes[48] | Yes |

Table 24: Availability of weather data (historical measurements & forecasts) per station

[35] https://kachelmannwetter.com/de/vorhersage/2745338-volkel/xltrend/euro, https://kachelmannwetter.com/de/vorhersage/2745338-volkel/xltrend/usa, https://kachelmannwetter.com/de/modellkarten/euro/gelderland, https://kachelmannwetter.com/de/modellkarten/standard/gelderland

[36] https://kachelmannwetter.com/de/vorhersage/2757667-deelen/xltrend/euro, https://kachelmannwetter.com/de/vorhersage/2757667-deelen/xltrend/usa

[37] https://kachelmannwetter.com/de/vorhersage/2939440-berlin-dahlem/xltrend/euro, https://kachelmannwetter.com/de/vorhersage/2939440-berlin-dahlem/xltrend/usa, https://kachelmannwetter.com/de/modellkarten/euro/brandenburg, https://kachelmannwetter.com/de/modellkarten/standard/brandenburg

[38] https://kachelmannwetter.com/de/vorhersage/2823538-berlin-tempelhof/xltrend/euro, https://kachelmannwetter.com/de/vorhersage/2823538-berlin-tempelhof/xltrend/usa

[39] https://kachelmannwetter.com/de/messwerte/gelderland/globalstrahlung/20200620-2000z.html

[40] https://kachelmannwetter.com/de/messwerte/berlin/globalstrahlung/20200730-2200z.html

[41] https://kachelmannwetter.com/de/messwerte/gelderland/bewoelkungsgrad/20200620-2000z.html

[42] https://kachelmannwetter.com/de/messwerte/berlin/bewoelkungsgrad/20200730-2200z.html

[43] https://kachelmannwetter.com/de/messwerte/gelderland/temperatur/20200620-2000z.html

[44] https://kachelmannwetter.com/de/messwerte/berlin/temperatur/20200730-2200z.html

[45] https://kachelmannwetter.com/de/messwerte/gelderland/luftfeuchtigkeit/20200620-2000z.html

[46] https://kachelmannwetter.com/de/messwerte/berlin/luftfeuchtigkeit/20200730-2200z.html

[47] https://kachelmannwetter.com/de/messwerte/gelderland/luftdruck-stationshoehe/20200620-2000z.html

[48] https://kachelmannwetter.com/de/messwerte/berlin/luftdruck-stationshoehe/20200730-2200z.html

## B. Energy production model scores

| | Score | Architecture |
|---|---|---|
| Wijchen 1 | | |
| Only radiation | 0.5586-0.5606 (validation)<br>0.4195-0.4587 (training) | Dense layer (5-20 neurons, sigmoid)<br>Dense layer (1 neuron, linear)<br>26 - 101 parameters<br>Batch size 18<br>1000 epochs |
| All but radiation | 0.7660-0.7776 (validation)<br>0.7382-0.7928 (training) | Dense layer (12-20 neurons, sigmoid)<br>Dense layer (1 neuron, linear)<br>133 - 221 parameters<br>Batch size 18<br>1500 epochs |
| All with radiation | 0.7954-0.8106 (validation)<br>0.6191-0.7031 (training) | Dense layer (12-20 neurons, sigmoid)<br>Dense layer (1 neuron, linear)<br>157 - 261 parameters<br>Batch size 18<br>1500 epochs |
| Wijchen 2 | | |
| Only radiation | 0.5186-0.5254 (validation)<br>0.4609-0.5105 (training) | Dense layer (5-20 neurons, sigmoid)<br>Dense layer (1 neuron, linear)<br>26 - 101 parameters<br>Batch size 32<br>2000 epochs |
| All but radiation | 0.6025-0.6816 (validation)<br>0.6914-0.7489 (training) | Dense layer (2-20 neurons, sigmoid)<br>Dense layer (1 neuron, linear)<br>23 - 221 parameters<br>Batch size 18<br>2000 epochs |
| All with radiation | 0.6251-0.6837 (validation)<br>0.7889-0.8167 (training) | Dense layer (12-20 neurons, sigmoid)<br>Dense layer (5 neurons, sigmoid)<br>Dense layer (1 neuron, linear)<br>215 - 351 parameters<br>Batch size 18<br>1500 epochs |

Table 25: Hyperparameter exploration for neural network training with a ten-day dataset as in table 14 and validating with the first and second validation dataset of the same table (Wijchen)

| | Score | Architecture |
|---|---|---|
| Berlin 1 | | |
| Only radiation | 0.6443-0.6735 (validation)<br>0.3250-0.5725 (training) | Dense layer (1-5 neurons, sigmoid)<br>Dense layer (1 neuron, linear)<br>6 - 26 parameters<br>Batch size 18<br>2000 epochs |
| All but radiation | 0.6438-0.7338 (validation)<br>0.3809-0.6545 (training) | Dense layer (16-20 neurons, sigmoid)<br>Dense layer (5-10 neurons, sigmoid)<br>Dense layer (1 neuron, linear)<br>177 - 301 parameters<br>Batch size 18<br>1000 epochs |
| All with radiation | 0.6717-0.7174 (validation)<br>0.3199-0.6182 (training) | Dense layer (1-16 neurons, sigmoid)<br>Dense layer (1 neuron, linear)<br>14 - 209 parameters<br>Batch size 18<br>1000 epochs |
| Berlin 2 | | |
| Only radiation | 0.7159-0.7493 (validation)<br>0.3277-0.5240 (training) | Dense layer (10-20 neurons, sigmoid)<br>Dense layer (1 neuron, linear)<br>51 - 101 parameters<br>Batch size 18<br>1000 epochs |
| All but radiation | 0.6971-0.7685 (validation)<br>0.5883-0.6767 (training) | Dense layer (5-10 neurons, sigmoid)<br>Dense layer (1 neuron, linear)<br>56 - 111 parameters<br>Batch size 32<br>2000 epochs |
| All with radiation | 0.6859-0.7174 (validation)<br>0.5692-0.6798 (training) | Dense layer (16-20 neurons, sigmoid)<br>Dense layer (0-10 neurons, sigmoid)<br>Dense layer (1 neuron, linear)<br>209 - 461 parameters<br>Batch size 18<br>1000 epochs |

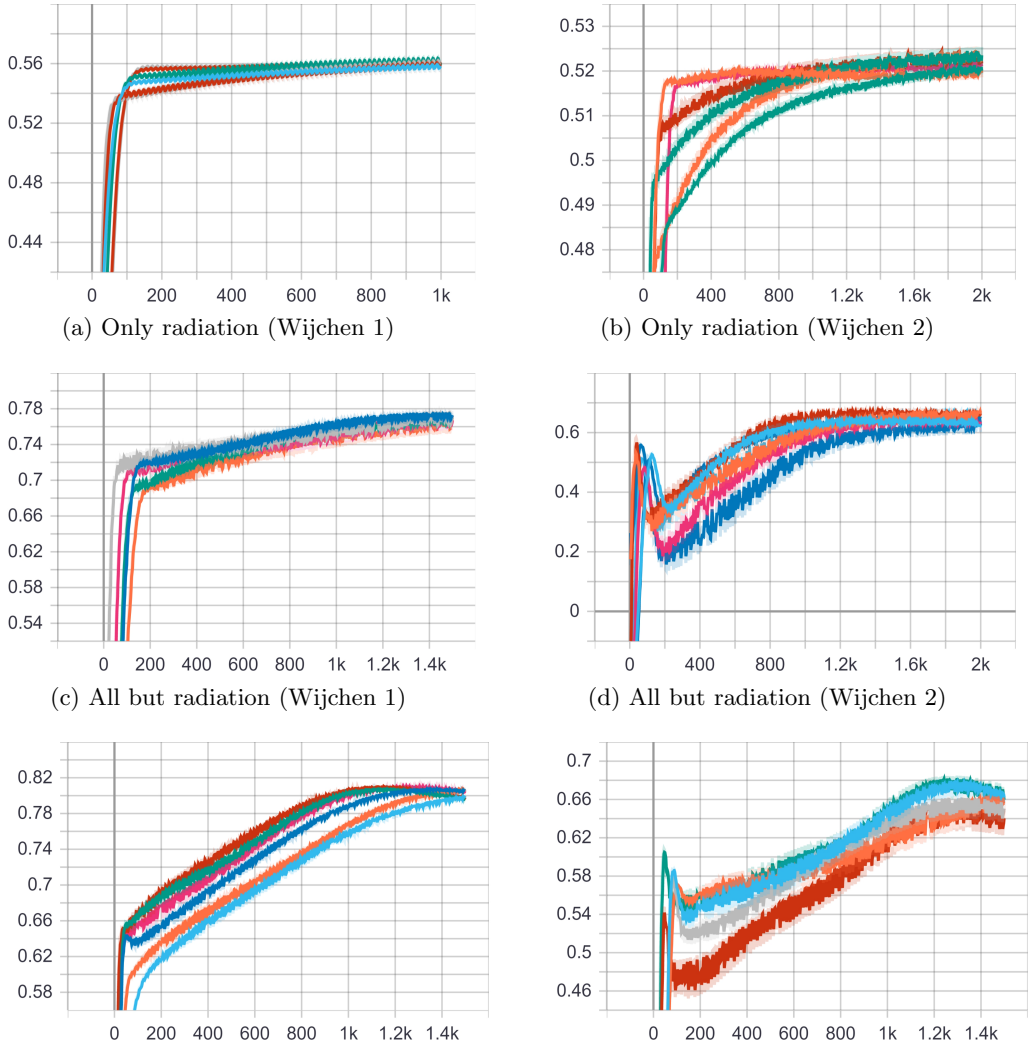
Table 26: Hyperparameter exploration for neural network training with a ten-day dataset as in table 14 and validating with the first and second validation dataset of the same table (Berlin)

| | Score | Architecture |
|---|---|---|
| Only radiation | 0.4952-0.5291 (validation)<br>0.5175-0.5325 (training) | Dense layer (1-20 neurons, sigmoid)<br>Dense layer (1 neuron, linear)<br>6 - 101 parameters<br>Batch size 32<br>1000 epochs |
| All but radiation | 0.7047-0.7294 (validation)<br>0.7634-0.7780 (training) | Dense layer (5-16 neurons, sigmoid)<br>Dense layer (1 neuron, linear)<br>56 - 177 parameters<br>Batch size 32<br>1000 epochs |
| All with radiation | 0.7107-0.7218 (validation)<br>0.7684-0.8076 (training) | Dense layer (5-16 neurons, sigmoid)<br>Dense layer (1 neuron, linear)<br>66 - 209 parameters<br>Batch size 32<br>1000 epochs |

Table 27: Hyperparameter exploration for neural network training with a one-month training and a ten-day validation dataset as in table 15 (Wijchen)

| | Score | Architecture |
|---|---|---|
| Only radiation | 0.7321-0.7402 (validation)<br>0.4133-0.5641 (training) | Dense layer (10-20 neurons, sigmoid)<br>Dense layer (1 neuron, linear)<br>51 - 151 parameters<br>Batch size 16<br>1000 epochs |
| All but radiation | 0.7994-0.8087 (validation)<br>0.5078-0.6487 (training) | Dense layer (5-16 neurons, sigmoid)<br>Dense layer (1 neuron, linear)<br>56 - 177 parameters<br>Batch size 16<br>1000 epochs |
| All with radiation | 0.8154-0.8351 (validation)<br>0.6154-0.7164 (training) | Dense layer (5-20 neurons, sigmoid)<br>Dense layer (1 neuron, linear)<br>66 - 261 parameters<br>Batch size 16<br>1000 epochs |

Table 28: Hyperparameter exploration for neural network training with a one-month training and a ten-day validation dataset as in table 15 (Berlin)




(a) Only radiation (Wijchen 1)

(b) Only radiation (Wijchen 2)

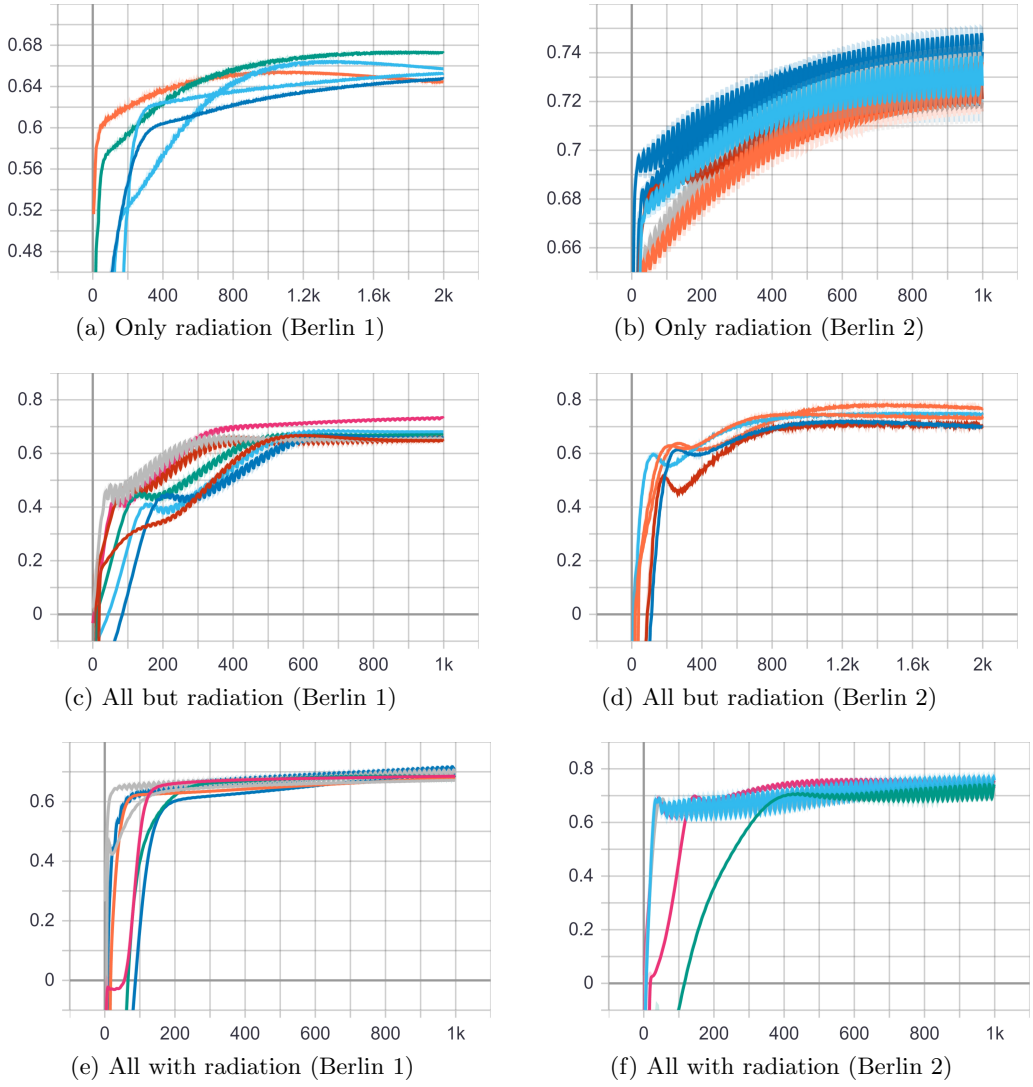
(c) All but radiation (Wijchen 1)

(d) All but radiation (Wijchen 2)

(e) All with radiation (Wijchen 1)

(f) All with radiation (Wijchen 2)

Figure 49: Validation $R^2$ scores over epochs for several neural network training runs with a ten-day dataset as in table 14 and validating with the first and second validation dataset of the same table (Wijchen)




(a) Only radiation (Berlin 1)

(b) Only radiation (Berlin 2)

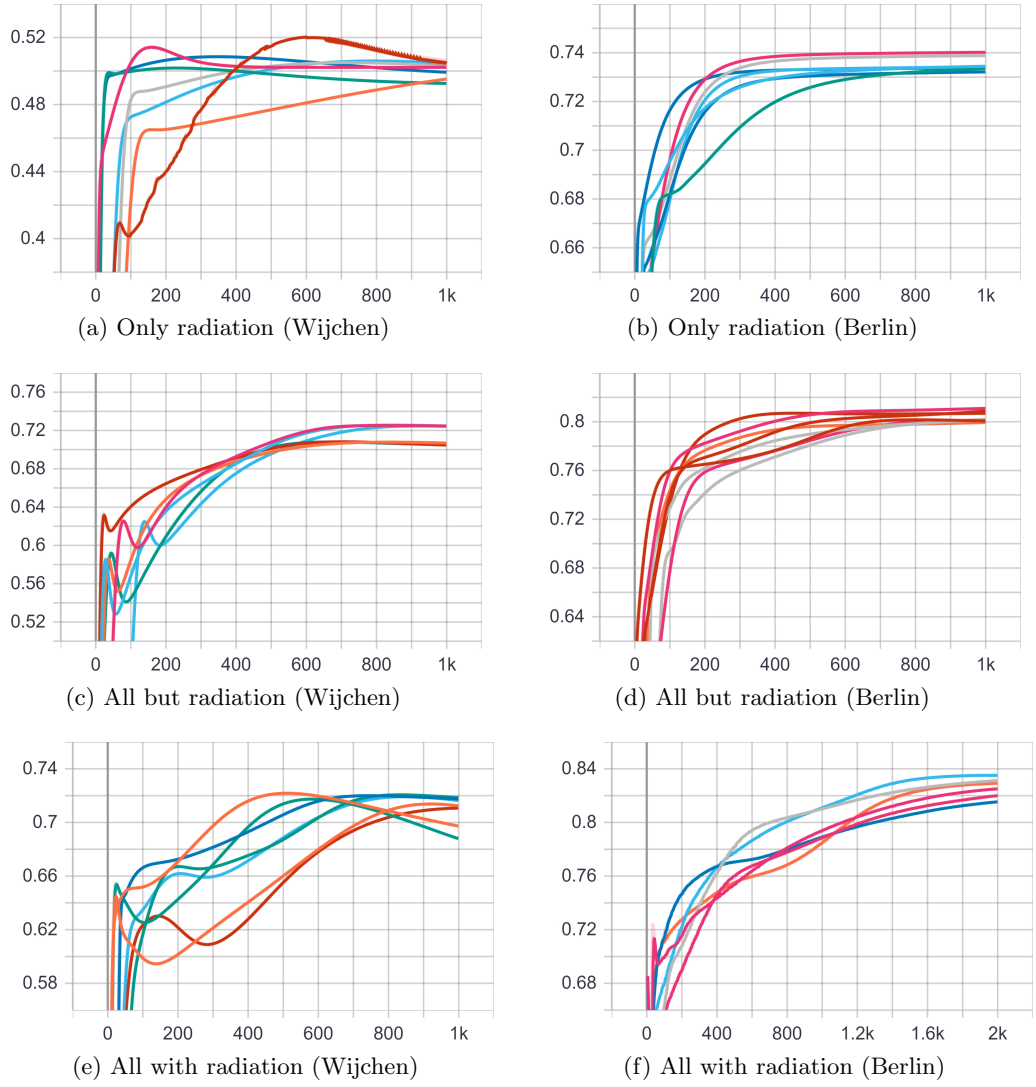
(c) All but radiation (Berlin 1)

(d) All but radiation (Berlin 2)

(e) All with radiation (Berlin 1)

(f) All with radiation (Berlin 2)

Figure 50: Validation $R^2$ scores over epochs for several neural network training runs with a ten-day dataset as in table 14 and validating with the first and second validation dataset of the same table (Berlin)




(a) Only radiation (Wijchen)

(b) Only radiation (Berlin)

(c) All but radiation (Wijchen)

(d) All but radiation (Berlin)

(e) All with radiation (Wijchen)

(f) All with radiation (Berlin)

Figure 51: Validation $R^2$ scores over epochs for several neural network training runs with a one-month training and a ten-day validation dataset as in table 15

# C. Task allocation algorithms

**Algorithm 1** Heuristic task allocation (first fit)

```
tasks = list of all tasks in consideration (ordered by width/height/area)
funcs = list of all step functions
for task x from tasks do
    for step function s from funcs do
        for t in [0 : 23-x.width] do
            if s[t : t+x.width].value ≥ x.height then
                x.start = t
                s[t : t+x.width].value -= x.height
            end if
        end for
    end for
end for
```

[x:y] denotes the integer range between x and y (inclusive).
z[x:y] denotes the values of z at indices x to y (inclusive).
z[x:y] - k denotes the element-wise subtraction of k from every value of z[x:y].

**Algorithm 2** Heuristic task allocation (next fit)

```
tasks = list of all tasks in consideration (ordered by width/height/area)
funcs = list of all step functions

last_func = 0
last_time = 0

for task x from tasks do
    for step function s from (funcs[last_func:] + funcs[0 : last_func-1]) do
        for t in ([last_time : 23-x.width] + [0 : last_time-1]) do
            if s[t : t+x.width].value ≥ x.height then
                x.start = t
                s[t : t+x.width].value -= x.height
                last_func = s
                last_time = t+x.width
            end if
        end for
    end for
end for
```

[x:y] denotes the integer range between x and y (inclusive).
z[x:y] denotes the values of z at indices x to y (inclusive).
z[x:y] - k denotes the element-wise subtraction of k from every value of z[x:y].
z1[:] + z2[:] denotes the concatenation of z1 and z2.

**Algorithm 3** Heuristic task allocation (best fit)

```
tasks = list of all tasks in consideration (ordered by width/height/area)
funcs = list of all step functions

for task x from tasks do
    best_pos = 0
    max_area = 0
    for step function s from funcs do
        for t in [0 : 23-x.width] do
            if sum(s[t : t+x.width].value - x.height) > max_area then
                best_pos = t
                max_area = sum(s[t : t+x.width].value - x.height)
            end if
        end for
    end for
    x.start = best_pos
    s[best_pos : best_pos+x.width].value -= x.height
end for
```

[x:y] denotes the integer range between x and y (inclusive).
z[x:y] denotes the values of z at indices x to y (inclusive).
z[x:y] - k denotes the element-wise subtraction of k from every value of z[x:y].