

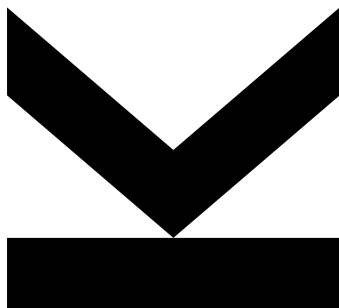
Author
Dominik Birngruber
12206237

Submission
Institute for System
Software

Thesis Supervisor
Dipl.-Ing. Dr. **Markus**
Weninger, BSc

October 2026

Photovoltaic System Forecasting and Electric Vehicle Charging Optimization



Bachelor's Thesis

to confer the academic degree of

Bachelor of Science

in the Bachelor's Program

Informatik

Abstract

As the number of electric vehicles (EVs) and photovoltaic (PV) systems installed at private homes grows, using self-produced solar power for EV charging is becoming increasingly attractive, especially as feed-in tariffs and grid electricity prices keep drifting apart. Wallboxes can already adapt their charging around PV excess power, but only if they are physically connected to the solar inverter, a connection that is not always available. This thesis, conducted together with KEBA Energy Automation, investigates whether solar production forecasts from third-party APIs can be used to approximate this behavior without an inverter connection. The resulting forecast is used to generate a charging schedule ahead of time that varies the charging power throughout the session to follow the predicted PV surplus, approximating the behavior of a directly connected wallbox without requiring an inverter connection.

Twelve forecasting API providers were evaluated for pricing, data granularity, and availability, narrowing the selection down to two commercially viable candidates: Forecast.Solar and Meteoblue. Over several months, their forecasts were compared against actual production data from two residential PV systems in Upper Austria using a custom-built data collection pipeline and analysis frontend. For forecasts made one hour ahead, both APIs deviated from the systems' actual production by 63% to 102% on average (MAPE), with Meteoblue consistently more accurate than Forecast.Solar. Grouping the days by cloud cover showed that the relative error of both APIs increases from 27–67% on clear days to 115–158% on overcast days, while the absolute error remains in a range that is not negligible relative to a typical household base load in all weather conditions. Based on these findings, generic forecasting APIs are not yet precise enough to reliably drive charging decisions for small-scale, distributed charging stations.

Despite this, an API-agnostic prototype backend and frontend were implemented that generate OCPP 1.6 charging profiles from forecast data. Because the underlying data source is fully exchangeable, the system provides a ready foundation for future work, such as custom-trained models or newly available providers, once a sufficiently accurate and commercially viable data source becomes available.

Kurzfassung

Mit der wachsenden Zahl an Elektrofahrzeugen (EVs) und Photovoltaikanlagen (PV) in Privathaushalten wird die Nutzung des selbst erzeugten Solarstroms zum Laden von Elektrofahrzeugen zunehmend attraktiv, insbesondere da sich Einspeisevergütungen und Strombezugpreise immer weiter auseinanderentwickeln. Wallboxen können den Ladevorgang bereits heute an den PV-Überschuss anpassen, allerdings nur, wenn eine physische Verbindung zum Solarwechselrichter besteht, eine Verbindung, die nicht immer verfügbar ist. Diese Arbeit, durchgeführt in Zusammenarbeit mit KEBA Energy Automation, untersucht, ob Solarprognosen aus APIs Dritter genutzt werden können, um dieses Verhalten auch ohne Wechselrichteranbindung nachzubilden. Die daraus resultierende Prognose wird genutzt, um im Vorhinein einen Ladeplan zu erstellen, der die Ladeleistung während des Ladevorgangs an den prognostizierten PV-Überschuss anpasst und so das Verhalten einer direkt angebotenen Wallbox auch ohne Wechselrichteranbindung annähert.

Zwölf Anbieter von Prognose-APIs wurden hinsichtlich Preisgestaltung, Datenauflösung und Verfügbarkeit evaluiert, wodurch sich die Auswahl auf zwei kommerziell einsetzbare Kandidaten, Forecast.Solar und Meteoblue, reduzierte. Über mehrere Monate hinweg wurden deren Prognosen mithilfe einer eigens entwickelten Datenerfassungs- und Analyseanwendung anhand realer Produktionsdaten zweier privater PV-Anlagen in Oberösterreich verglichen. Bei Prognosen eine Stunde im Voraus wichen beide APIs im Mittel um 63% bis 102% (MAPE) von der realen Produktion ab, wobei Meteoblue durchgehend genauer war als Forecast.Solar. Eine Gruppierung der Tage nach Bewölkung zeigte, dass der relative Fehler beider APIs von 27–67% an klaren Tagen auf 115–158% an bedeckten Tagen ansteigt, während der absolute Fehler bei jeder Wetterlage in einem Bereich bleibt, der im Verhältnis zur typischen Grundlast eines Haushalts nicht vernachlässigbar ist. Auf Basis dieser Ergebnisse sind generische Prognose-APIs derzeit noch nicht präzise genug, um Ladeentscheidungen für kleine, verteilte Ladestationen zuverlässig zu steuern.

Trotzdem wurde ein API-unabhängiger Prototyp für Backend und Frontend umgesetzt, der aus den Prognosedaten Ladeprofile nach dem OCPP 1.6-Standard erzeugt. Da die zugrunde liegende Datenquelle vollständig austauschbar ist, bildet das System eine solide Grundlage für zukünftige Arbeiten, etwa eigens trainierte Modelle oder neu verfügbare Anbieter, sobald eine ausreichend genaue und kommerziell tragfähige Datenquelle zur Verfügung steht.

Contents

Abstract	ii
Kurzfassung	iii
1 Introduction	1
1.1 Objectives and Approach	1
1.2 Outline	1
2 Background	3
2.1 The KEBA Wallbox	3
2.2 The OCPP Standard	3
3 Finding and Evaluating Forecasting APIs	5
3.1 Finding API Providers	5
3.2 Data Collection	6
3.2.1 Reference PV Systems	6
3.2.2 Process and Architecture	7
3.2.3 Problems	8
3.3 Data Evaluation	9
3.3.1 How Far Do Forecasts Deviate From Actual Production? . . .	12
3.3.2 Does Forecast Lead Time Affect Accuracy?	13
3.3.3 Does Accuracy Depend on the Weather?	14
3.3.4 What Characterizes the Best and Worst Days?	15
3.3.5 Summary and Discussion of Findings	26
4 System Architecture	27
5 Frontend	29
6 Backend	32
6.1 Architecture	32
6.2 Data models	33
6.3 Challenges	34
7 Challenges and Future Work	35
7.1 Challenges and Limitations	35
7.2 Future work	36
8 Related Work	37
8.1 A review and evaluation of the state-of-the-art in PV solar power forecasting: Techniques and optimization	37
8.2 Charge Scheduling Optimization of Plug-In Electric Vehicle in a PV Powered Grid-Connected Charging Station Based on Day- Ahead Solar Energy Forecasting in Australia	38
8.3 Forecast-based Optimal Operation of EV Charging Station with PV Considering Charging Demand and Distributed System	39

9 Reflection and Conclusion	40
9.1 Reflection	40
9.2 Conclusion	40
Bibliography	42

Chapter 1

Introduction

With the rise in EV (electric vehicle) ownership and PV (photovoltaic) systems for people at home, maximizing the potential of self-produced power is getting more important. Prices for selling power and buying power are drifting further apart, making it uneconomical to sell the power when one does not need it and buy it at a later date. While home batteries are now the de facto standard when installing a new PV system, they are either not enough to bridge the gap for EV use, or they are not installed for already existing systems.

Current wallboxes already have the feature to charge an EV with the power not used in the household, but they need a connection to the inverter to be able to do this. In some cases, however, this connection is not possible. To use the self-produced power to the fullest, even in these cases, this thesis aims to bridge this gap and enable smart EV charging by generating a charging schedule ahead of time that varies the charging power throughout the session to approximate this behavior without a direct inverter connection. This thesis is done together with KEBA Energy Automation [8] and is intended to be used with their series of internet-connected wallboxes.

1.1 Objectives and Approach

The goal of this thesis is to find and evaluate different API providers with real-world production data. Since the reference production data used for comparison is only available at an hourly resolution, only forecasting APIs that could provide comparably granular, hour-to-hour data were considered as usable candidates. Finding and evaluating the documentation of the APIs was done manually; collection was done with periodically running jobs, while evaluation was done manually with a graphical interface. Also, a frontend and backend prototype were created using Angular for the frontend and Kotlin with Quarkus as the backend.

1.2 Outline

Chapter 2 introduces the background of the external components this concept builds on, namely the physical hardware and the OCPP standard. Chapter 3 describes how forecasting API providers were selected, how their forecasts were collected with real-world production data, and how this data was evaluated, including the difficulties encountered. Chapter 4 presents the architecture of

the resulting system, followed by the frontend in Chapter 5 and the backend in Chapter 6. Chapter 7 discusses the challenges and limitations of this thesis as well as possible future work; Chapter 8 relates the results to existing research; and Chapter 9 concludes the thesis with a reflection.

Chapter 2

Background

KEBA Energy Automation produces multiple types of wallboxes, which are shown to the user in the KEBA eMobility Portal. In this portal, users can configure their wallboxes and view charging statistics. Most importantly for this thesis, you can also configure charging profiles for the wallboxes, all via an open standard named OCPP.

2.1 The KEBA Wallbox

KEBA produces different kinds of wallboxes, including AC and DC wallboxes. In this thesis, generating charging profiles is restricted to AC wallboxes, since the P30 and P40, which are usable with the eMobility Portal, are both AC wallboxes. These wallboxes are connected to the infrastructure of the portal via a WebSocket connection. Users can also communicate with them via a local REST endpoint, but this cannot be used here since external data is also needed.

The backend is a cloud-based system that communicates internally via asynchronous messaging, using queues to pass messages from producers to consumers, with multiple queues existing for different services. Messages sent within the eMobility backend are partly proprietary, while messages sent to the wallboxes conform to the OCPP 1.6 standard, which is discussed in the next section.

At this time, there are two forms of charging profiles available: time-based and spot-price charging profiles. These tell the wallbox whether it should charge and, if so, with how much amperage. There is also the possibility to configure PV excess charging, which requires a physical connection to the solar inverter.

2.2 The OCPP Standard

The OCPP (Open Charge Point Protocol) is used by the wallbox to communicate with the backend. It is an open standard, first introduced in 2010 and still under development; the current version is 2.1, released in 2025, but the wallbox uses version 1.6 [15], which introduced the implementation with WebSockets and the smart-charging capabilities needed for this thesis. Although version 1.6 is already eleven years old, security patches are still released because 2.0 is not backward compatible and is thus not implemented for a lot of charge points.

The feature used in this thesis is *Charging Profiles*, which tell the wallbox when to start charging, how fast it can charge, or whether it can charge at all. This is done via SetChargingProfile messages sent from the central system, in this case the eMobility Portal, to the wallbox. This can happen at different times, depending on what the profile should do. In most situations, it is tightly coupled with the start of a transaction, the formal name for a single charging session. Sending a message can be triggered when a transaction is started either locally or remotely, during a transaction to change the profile that is currently active, or outside a transaction to set a default profile for every subsequent transaction.

The charging profile request itself contains only the basic fields it needs. First, it needs the connectorId to specify which charge point it belongs to. Then it needs the charging profile itself, telling the wallbox at which interval it can draw how much current or power.

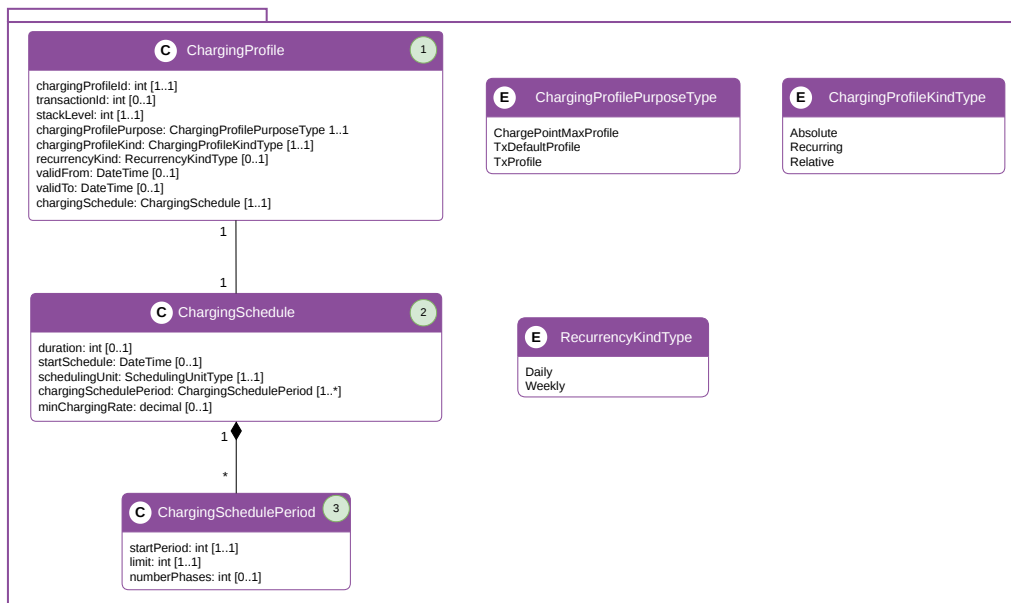


Figure 2.1: Charging profile described in the OCPP 1.6 standard with (1) ChargingProfile (2) ChargingSchedule (3) ChargingSchedulePeriod.

Figure 2.1 shows the three nested objects that make up a charging profile. Every profile (1) needs an ID, a stack level to determine which of multiple current profiles has precedence, a purpose, in this case `TxProfile`, to tell the charge point this profile is only valid for a single transaction, a kind of schedule (whether the times are absolute, relative, or recurring), and the schedule itself. Optionally, one can also pass the ID of the transaction to restrict the profile to it, a recurrence type (not needed here), and a validity interval.

The schedule itself (2) needs the unit it uses, watts or amps, with amps being preferred for AC charging, and one or more periods. It may also include a start, a duration, and a minimum charging rate.

Finally, a schedule period (3) specifies when it starts, how many phases it can use, and what limit applies.

Chapter 3

Finding and Evaluating Forecasting APIs

This chapter describes how suitable forecasting APIs were found and chosen, how real-world production data was collected, and how the resulting data was evaluated. Figure 3.1 provides an overview of the three involved components.

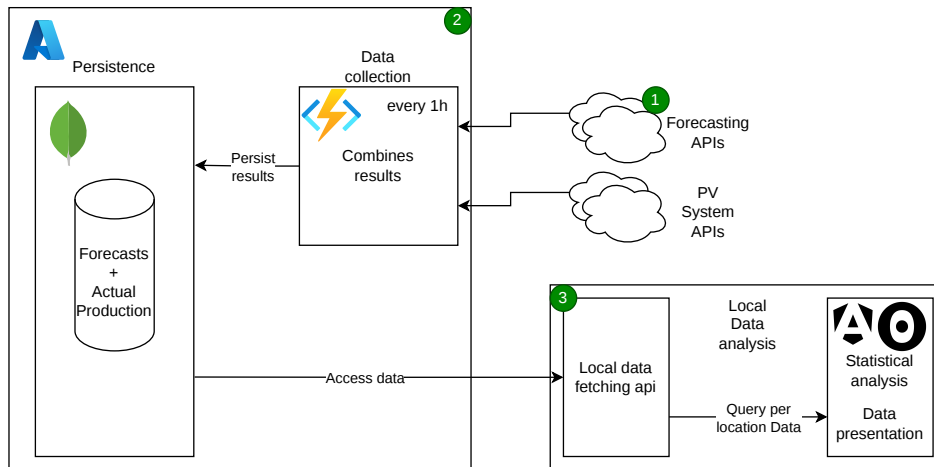


Figure 3.1: Architecture used for evaluating the API providers with its three components: (1) the providers themselves, (2) the data collection runner, and (3) the evaluation frontend.

First, API providers (1) were needed to get the weather or solar radiation data for the locations of the users. Afterward, the whole infrastructure for collecting and persisting the collected data (2) had to be set up. Lastly, the data needed to be analyzed with a dedicated frontend for graphs and statistical analysis (3).

3.1 Finding API Providers

Finding fitting APIs proved to be difficult. Most of the providers' web pages had the documentation hidden deep inside, and while there were no fully free APIs, some could at least be tested on their pages. Of the 12 API providers found through research, three main groups formed, plus two outliers.

First, the outliers. Two governmental APIs were found that are free to use: *PVGIS* [4] from the European Union and *NASA POWER* [13] from the US govern-

ment. Both provide production data, but PVGIS could only be used for planning and not for hour-to-hour data, and NASA POWER seemed to also provide hourly data but was unavailable during the research phase.

The first real category consisted of providers with insufficient public documentation. This group included *MeteoForEnergy* [11], *WorldWeatherOnline* [26], and *Enercast* [3]. These seemed promising at first, but their web pages did not contain enough documentation of the API itself, its pricing, or the values it returns to allow a proper evaluation.

The next category is providers that only return solar radiation, i.e., solar irradiance in W/m^2 , rather than an actual power output in watts. Using this data still requires manually calculating the power produced for each location from the panel's characteristics, unlike the last category below, which returns production estimates. This group included *MeteoMatics* [12], *AccuWeather* [1], and *Stormglass* [25].

The last category returns exactly the needed data at varying resolutions. These were *Solcast* [24], *Meteoblue* [10], *Forecast.Solar* [5], and *OpenWeather* [18]. For all of them, the production forecast per location can be queried and returned in watts or watt-hours, depending on the endpoint used. The evaluation in this thesis uses the watt-based endpoint throughout, matching the units of the actual production data it is compared against. While *Solcast* and *OpenWeather* seemed promising, their pricing models were not feasible for this thesis or for later use in a commercial product. *Forecast.Solar* and *Meteoblue* both had pricing viable for the theoretical work and a later commercial product, with *Forecast.Solar* even having a genuinely usable free tier. Of the 12 providers considered, *Forecast.Solar* and *Meteoblue* were selected for the rest of this thesis, as they were the only ones offering directly usable, hourly production forecasts at a viable price.

3.2 Data Collection

The next step was to collect real-world data alongside the forecasts. For this, a colleague from KEBA volunteered to provide the actual production data from their PV system, together with another PV system from the author. These two systems are located in northern and central Upper Austria.

3.2.1 Reference PV Systems

Both systems are monitored via a SolarEdge inverter, which also provides the actual production data used for evaluation through the SolarEdge API [23].

The first system, referred to as the *Mühlviertel* location, belongs to the author and is located north of the Danube in Upper Austria, 30 minutes north of Linz. It has a combined peak power of about 18.2 kWp, split across three roof-mounted sub-arrays facing east, south, and west at a 45° tilt.

The second system, referred to as the *Traunviertel* location, belongs to a KEBA colleague and is situated south of the Danube in Upper Austria. It has a combined peak power of about 13.9 kWp, split across three roof-mounted sub-arrays facing east, south, and west at a shallower 22° tilt.

Neither system reported any relevant shading issues during the observation period.

3.2.2 Process and Architecture

Data collection was run once an hour by an Azure Function written in TypeScript. The data collector called the SolarEdge API for both systems, where it received the actual production from the previous hour. At the same time, the two chosen APIs were called to query the production data for the locations using latitude, longitude, peak power, and azimuth. The data is received as-is and saved to a CosmosDB instance in Azure, including location data, the system's actual production, and the forecast.

This collection was run once every hour because the received forecasts differed every hour, and the goal was to find a good middle ground for how often the APIs need to be queried. Another problem was that, for Meteoblue, every request cost tokens, and during the night, when no production can happen, these requests were paused. Because Forecast.Solar has a flat-rate subscription model, we omitted the optimization for it to better track forecast changes during the night. The pause from 19:00 to 07:00 was chosen when the data collection was set up in December, as this roughly matched the hours without daylight in the winter months, and it was kept for the rest of the collection period. This schedule refers to the data collection setup only. The evaluation in Section 3.3 uses data from February through April 2026, for which Meteoblue forecasts are available only from 26 February to 24 April 2026, since the purchased Meteoblue tokens were used up at that point. Since the days get longer over this period, Meteoblue forecasts are also missing for some early-morning and evening hours with low but non-zero production.

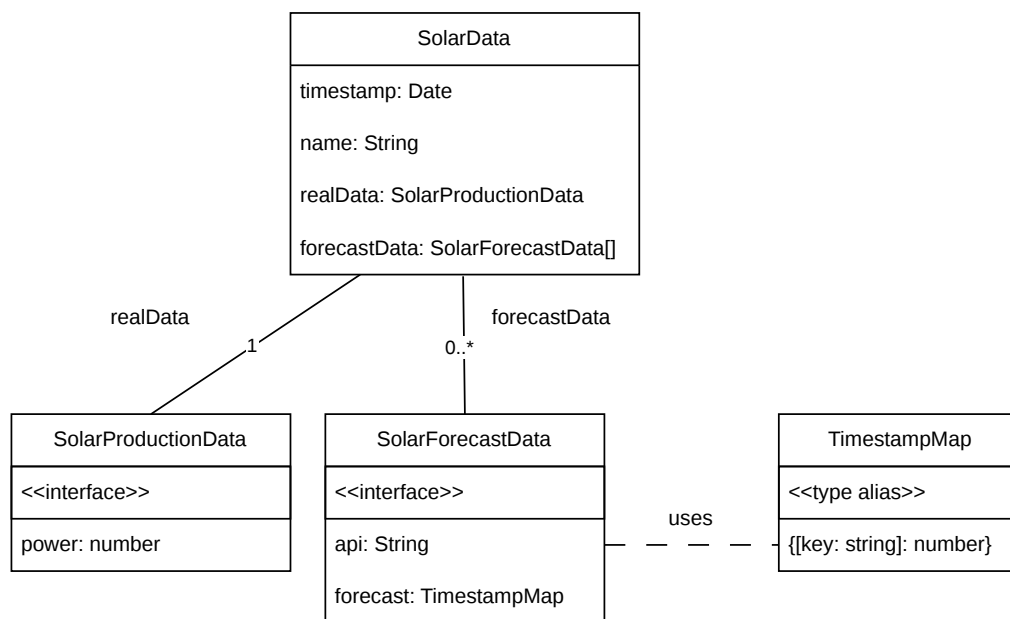


Figure 3.2: The data model for saving the collected data.

Data Models The collected data is persisted as shown in Figure 3.2. Since the collected forecasts have different timestamp formats, the `TimestampMap` needs to be more general. The first field saved is the name of the location, in this case just the name of the person the PV system belongs to. The next field is the timestamp of the recorded production data, an ISO date string, and can thus be saved as a `Date` object. After that, an array of API results is saved, making it open to expansion, should a new API become available to plug into the data collection.

Every object in this array contains the ID of the API, in this case always the provider name in lowercase, and the provided forecast. For `Meteoblue`, this is a map with ISO date strings as keys and the forecasted production as decimal numbers. `Forecast.Solar` does not use ISO date strings, but only times in the format `hh:mm`. Because the post-processing happens in a later step, this data is saved as-is into the database.

3.2.3 Problems

We encountered several problems during API data collection. First, the documentation of some APIs was hard to find, even though the actual content, once found, was usually of high quality. For `Forecast.Solar`, the documentation was straightforward to find, whereas for `Meteoblue` it was nested several levels deep. The same was true for other APIs while looking for the right endpoints. This could not even be mitigated by the use of AI assistance, since the APIs changed often or the model hallucinated parameters that did not exist, even in older versions.

The next problem was pricing. The APIs had different, sometimes intransparent, forms of payment. While `Forecast.Solar` has a simple yearly fee for different use cases, the data collection used the 17 EUR/year tier, with data quality not differing much, just the resolution and commercial usability; `Meteoblue` was less transparent in its use of credits. Each request used a different amount of these credits, making it more flexible but, in the end, more expensive. The reason is probably that, while `Forecast.Solar` only offers solar forecasting, `Meteoblue` offers different weather-related services, with some costing fewer credits per request than others. There is also the option of other subscription tiers for possible future commercial use. We also considered including a third API, `Solcast`, in the evaluation. Yet it was the hardest of the three to access: there was no easy way to get a free trial, and the only way to get the paid version was through a sales conversation. While the pricing was more competitive when rolled out across many locations, the price for only a few locations was too high to try, since it exceeded 2000 EUR per month.

Another problem I encountered, more on the technical side, was time handling in JavaScript. A bug that was only caught after some time had passed involved the method `Date().getMonth()`, since this method returns the number of the month from zero through eleven, not one through twelve. This caused an off-by-one error and old data being queried from the actual PV systems because the timestamp for the query had to be built manually. This data could be corrected afterward, since the production data can be queried far into the past.

3.3 Data Evaluation

Data evaluation was done manually with a self-coded Angular application written in TypeScript and a small Node.js backend for querying the data. The main part of the page consists of input fields to choose location, timespan, and API provider. Below that, the main plot is shown together with the error statistics, as shown in Figure 3.3. This plot shows the systems' actual production, the forecast, and the difference between them.

Chosen Forecast

MSE: 9,386,244.045 Watt²

RMSE: 3,063.698 Watt

MAE: 2,475.806 Watt

MedAE: 1,981.333 Watt

MAPE: 62.615 %

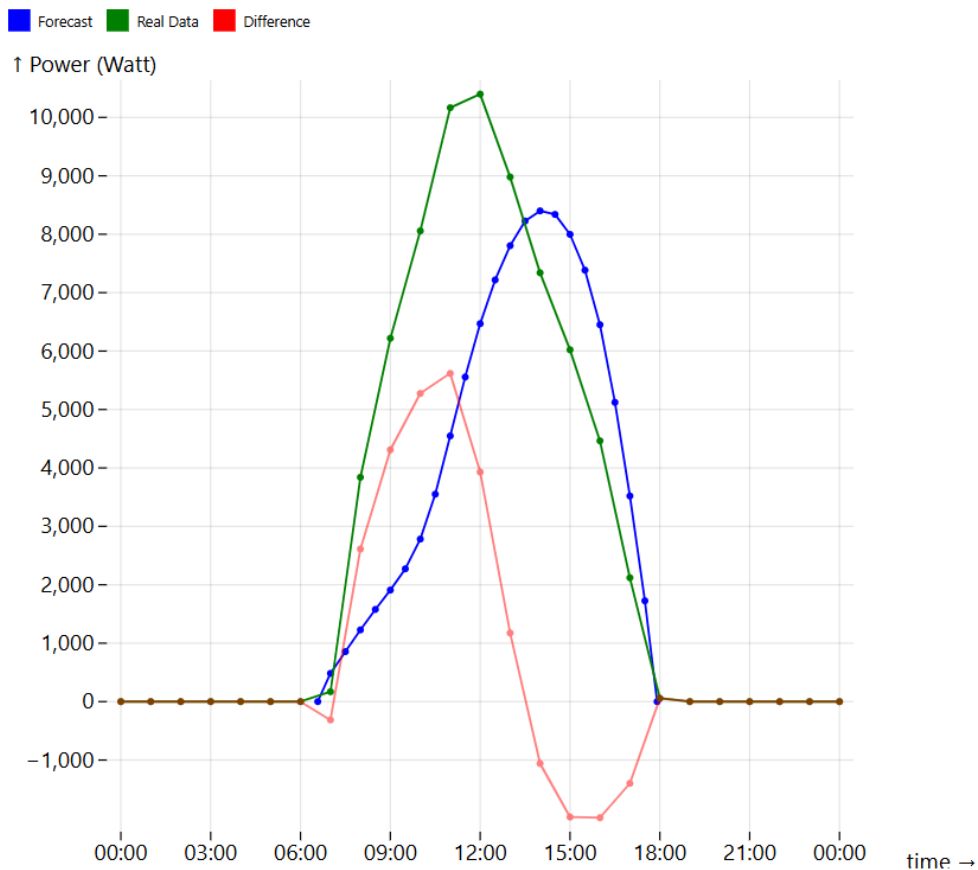


Figure 3.3: Main part of the analysis frontend, showing the forecast, the actual production, and their difference for a selected location, time span, and API provider.

The error metrics used across all plots are the same and are explained in the list below. For the calculation of the error, a toggle was added to exclude non-production hours, meaning hours where production is zero are not included in the calculation.

- **MSE**: Mean Squared Error takes all hourly differences between the forecast and the actual production, squares them, and averages them. Squaring

gives larger errors disproportionately more weight, making MSE particularly sensitive to outliers.

- **RMSE:** Root Mean Squared Error takes the square root of the MSE. It retains the stronger weighting of large errors while expressing the result in the same unit as the original values (watts instead of square watts).
- **MAE:** Mean Absolute Error takes the average of the hourly absolute differences between the forecast and the actual production. Unlike MSE and RMSE, errors are weighted linearly, so large errors have less influence on the result.
- **MedAE:** Median Absolute Error takes the median of the hourly absolute differences between the forecast and the actual production. Because the median is largely unaffected by a small number of extreme errors, MedAE is particularly robust to outliers.
- **MAPE:** Mean Absolute Percentage Error expresses each hourly absolute error relative to that hour's actual production and averages these relative errors, making the result easy to interpret as a percentage difference. However, MAPE is undefined for an actual production of zero and can become very large for values close to zero, which is one of the reasons non-production hours can be excluded from the calculation.

All graphs were made with Observable Plot [14], a JavaScript library focused on plotting data. All plots are line plots, and for better analysis, all parts of the plot can be toggled, which proved to be more difficult than expected: we had to add listeners to the labels manually via DOM manipulation. Especially for comparison plots, such as the one shown in Figure 3.4, this reduced visual noise while still allowing the whole picture to be seen when needed. All plots and metrics also show tooltips explaining each statistic's purpose and show the numerical value of each data point.

Figure 3.5 shows another variant of this comparison, contrasting forecasts fetched 1, 3, or 6 hours before the forecasted hour, together with their error statistics.

With these plots, it is easy to see the differences both visually and textually, making it easier to decide which API to use.

In this frontend, not only the plotting and calculation take place, but also the post-processing of the data, bringing all data into the same time zone. As already visible in the screenshots, the APIs differ significantly from the system's actual production, especially on cloudy days, when solar radiation can vary widely and change quickly.

The rest of this section answers four questions: how far the forecasts typically differ from actual production (Section 3.3.1), whether forecast lead time affects that accuracy (Section 3.3.2), whether the accuracy depends on the weather (Section 3.3.3), and what characterizes the best and worst performing days (Section 3.3.4). The numbers used to answer the first three questions are based on the data collected from February through May 2026. January was excluded since it was affected by the off-by-one bug described in Section 3.2.3, and the corrected data for that month was not considered reliable enough for analysis. Over these four months, 2737 hourly records were collected for each location, of

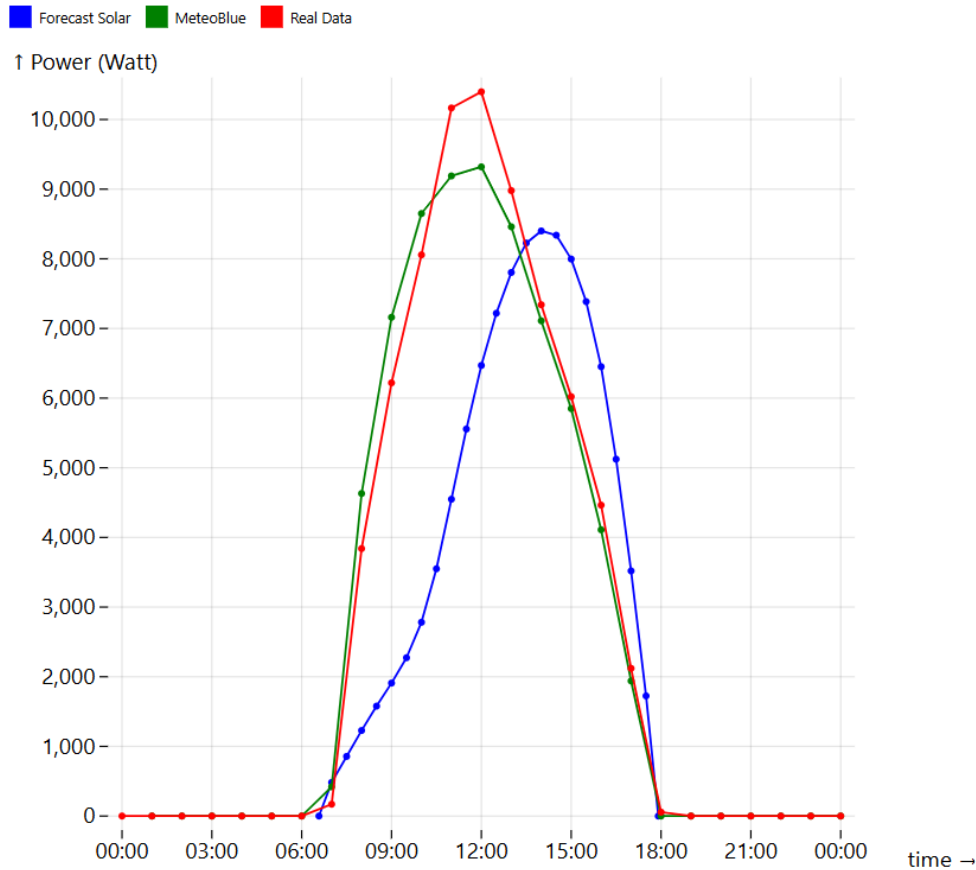
Forecast.SolarMSE: 9,386,244.045 Watt²RMSE: 3,063.698 WattMAE: 2,475.806 WattMedAE: 1,981.333 WattMAPE: 62.615 %**MeteoBlue**MSE: 379,156.608 Watt²RMSE: 615.757 WattMAE: 511.528 WattMedAE: 436.333 WattMAPE: 28.228 %

Figure 3.4: Comparison plot between the different APIs and their error statistics for the same location and time span.

which around 1470 had non-zero actual production. Meteoblue forecasts, however, are only available from 26 February to 24 April 2026, and only for queries made during the day. To compare both APIs on equal terms, the error metrics below are therefore only calculated for target hours where both APIs provided a forecast at the respective lead time, and the system's actual production was non-zero, applying the non-production-hour exclusion described above. The *lead time* (or forecast horizon) of a forecast value is the time between the moment the forecast was queried from the API and the hour it predicts production for. A production value queried at 10:00 for the hour starting at 13:00 therefore has a lead time of 3 hours. For a given hourly query, the 1h, 3h, and 6h forecasts used below refer to the value the query predicted for 1, 3, and 6 hours after the query was made and not to three separately fetched forecasts. A target hour for which not both APIs provided a forecast is excluded from the calculation, rather than treating the missing value as a zero prediction.

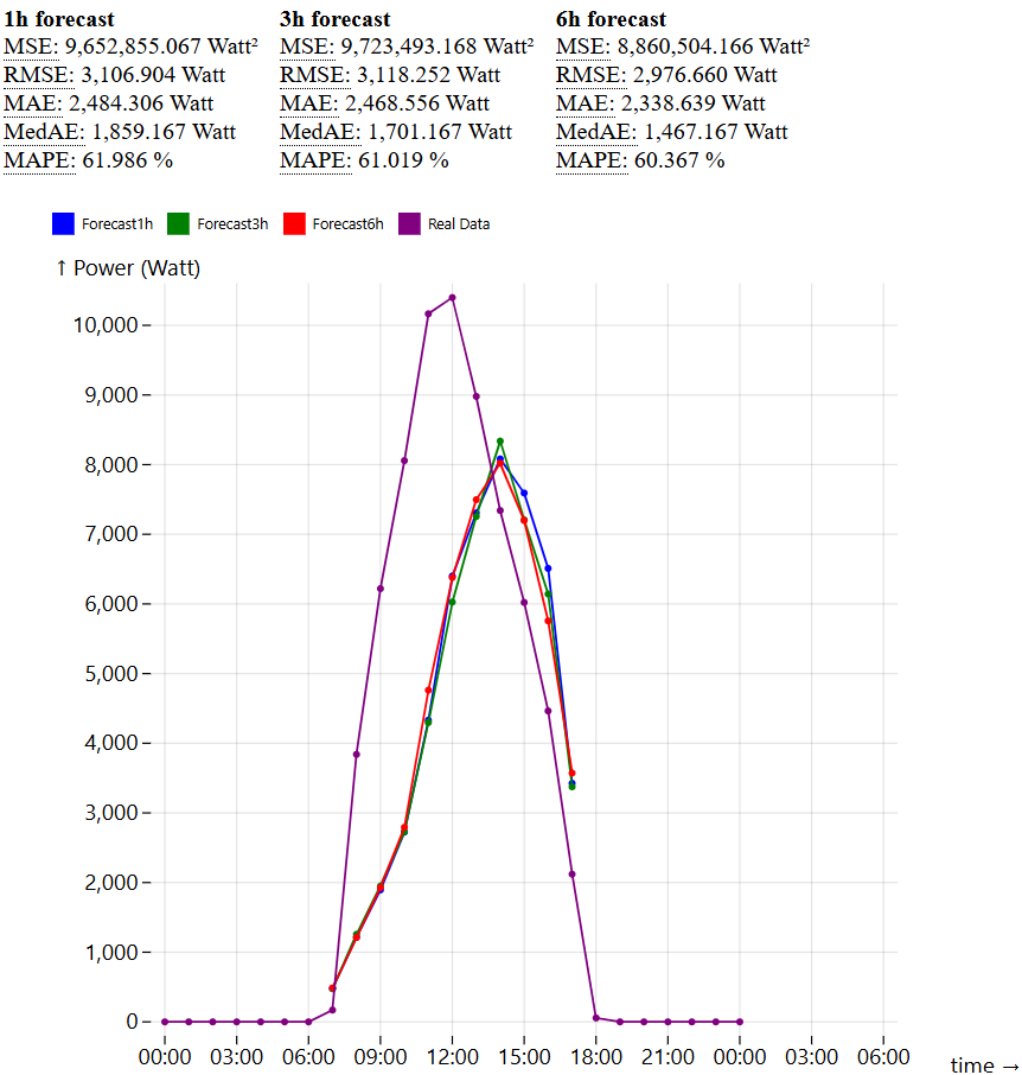


Figure 3.5: Comparison plot showing the charts for data fetched 1, 3, or 6 hours in advance, along with the corresponding error statistics.

3.3.1 How Far Do Forecasts Deviate From Actual Production?

Table 3.1 shows the error metrics for the 1h forecasts for both APIs and both locations; for each target hour, the value from the query made one hour before is used. Since only target hours for which both APIs provided such a forecast are included, both APIs of a location are evaluated on exactly the same n hours. Even on these hours, both APIs deviate substantially from actual production: a MAPE between 63% and 102% means the typical error is of the same order of magnitude as the value being predicted. Meteoblue is more accurate than Forecast.Solar on every metric at both locations, with an MAE of roughly half the size (1121 W compared to 2305 W for the Mühlviertel location, 914 W compared to 1548 W for the Traunviertel location). Restricting the comparison to shared hours matters: evaluated on all of its own 1h forecasts from February through May, which include more early-morning and evening hours with low production as well as the high-production month of May, Forecast.Solar reaches a MAPE of 152% for the Mühlviertel location and 132% for the Traunviertel loca-

Location	API	RMSE (W)	MAE (W)	MedAE (W)	MAPE (%)	n
Mühlviertel	Forecast.Solar	3176	2305	1501	102.1	690
Mühlviertel	Meteoblue	1509	1121	870	85.1	690
Traunviertel	Forecast.Solar	2116	1548	1021	97.0	695
Traunviertel	Meteoblue	1230	914	726	63.2	695

Table 3.1: Error metrics of the 1h forecasts per location and API, evaluated on the target hours with non-zero production for which both APIs provided a forecast (26 February to 24 April 2026). **n** is the number of hourly records included in that row's calculation.

tion. Both APIs are also consistently more accurate for the Traunviertel location than for the Mühlviertel location, even though the two locations are only about 47 km apart and monitored with hardware by the same company, indicating that forecast quality can differ noticeably even between two locations within a small area.

3.3.2 Does Forecast Lead Time Affect Accuracy?

Table 3.2 breaks down the comparison by forecast lead time, again only using target hours for which both APIs provided a forecast at the respective lead time. The 1h rows are therefore identical to Table 3.1.

Location	API	Lead time	MAE (W)	MedAE (W)	MAPE (%)	n
Mühlviertel	Forecast.Solar	1h	2305	1501	102.1	690
Mühlviertel	Forecast.Solar	3h	2630	1831	80.9	590
Mühlviertel	Forecast.Solar	6h	2117	1456	79.4	416
Mühlviertel	Meteoblue	1h	1121	870	85.1	690
Mühlviertel	Meteoblue	3h	1297	923	64.5	590
Mühlviertel	Meteoblue	6h	1106	670	47.5	416
Traunviertel	Forecast.Solar	1h	1548	1021	97.0	695
Traunviertel	Forecast.Solar	3h	1740	1220	69.2	588
Traunviertel	Forecast.Solar	6h	1552	943	68.2	415
Traunviertel	Meteoblue	1h	914	726	63.2	695
Traunviertel	Meteoblue	3h	1183	975	57.9	588
Traunviertel	Meteoblue	6h	1072	845	42.1	415

Table 3.2: Error metrics per location, API, and forecast lead time (1h/3h/6h), evaluated on the target hours with non-zero production for which both APIs provided a forecast at that lead time (26 February to 24 April 2026). **n** is the number of hourly records included in that row's calculation.

For both APIs and locations, the percentage error (MAPE) decreases from 1h to 6h lead time, while the absolute error (MAE) stays roughly flat, with its highest value at 3h. This does not mean that forecasts made further ahead are more accurate, however. Since Meteoblue was only queried during the day, the set of

target hours differs between the lead times: the 1h forecasts cover target hours starting in the morning, whereas the 6h forecasts only reach target hours from around noon onwards and increasingly fall into evening hours without production, which is also why n shrinks from about 690 to about 415. The 6h set thus contains no low-production morning hours, which strongly inflate MAPE, so the lead times are not compared on equivalent hours. Within these limits, the data does not show that querying closer to the target hour substantially improves accuracy for either API.

3.3.3 Does Accuracy Depend on the Weather?

To check whether forecast accuracy depends on the weather, instead of relying on individual example days, every day of the comparison period was assigned to a category based on the daily mean cloud cover reported by the Open-Meteo historical weather API [16] for the coordinates of the respective location: *clear* (below 40%), *partly cloudy* (40% to 80%), and *overcast* (above 80%). These categories align with the measured production: the average daily yield of the Mühlviertel location drops from 73.1 kWh on clear days to 54.4 kWh on partly cloudy days and 33.1 kWh on overcast days (Traunviertel location: 60.9, 47.3, and 24.9 kWh). The error metrics were then calculated in the same way as in Table 3.1: for 1h forecasts on shared target hours with non-zero production, but separately for each category. Since the MAE grows with the amount of power produced, while the MAPE grows strongly at low production, Table 3.3 additionally lists the normalized MAE (**nMAE**), which is the MAE divided by the mean actual production of the same hours, and the **bias**, the mean signed difference between forecast and actual production. A negative bias means that the API underestimates production on average.

Location	Weather	API	Days	n	MAE (W)	MAPE (%)	nMAE (%)	Bias (W)
Mühlviertel	clear	Forecast.Solar	20	236	2790	66.7	45.3	-2135
Mühlviertel	clear	Meteoblue	20	236	1015	36.9	16.5	62
Mühlviertel	partly cloudy	Forecast.Solar	23	275	2327	101.3	51.4	-1597
Mühlviertel	partly cloudy	Meteoblue	23	275	1181	78.6	26.1	387
Mühlviertel	overcast	Forecast.Solar	15	179	1632	150.1	58.9	-267
Mühlviertel	overcast	Meteoblue	15	179	1171	158.4	42.2	767
Traunviertel	clear	Forecast.Solar	19	222	1760	55.0	33.9	-1465
Traunviertel	clear	Meteoblue	19	222	871	26.9	16.8	-437
Traunviertel	partly cloudy	Forecast.Solar	23	278	1633	88.1	42.1	-1177
Traunviertel	partly cloudy	Meteoblue	23	278	958	56.1	24.7	96
Traunviertel	overcast	Forecast.Solar	16	195	1185	157.3	58.1	140
Traunviertel	overcast	Meteoblue	16	195	900	114.7	44.1	586

Table 3.3: Error metrics of the 1h forecasts per location, weather category, and API, evaluated on the same target hours as Table 3.1 (26 February to 24 April 2026). Days were classified by their daily mean cloud cover from Open-Meteo [16]: clear below 40%, partly cloudy 40% to 80%, overcast above 80%. **Days** is the number of days and **n** the number of hourly records in each category.

For both APIs and both locations, the relative error increases steadily from clear to overcast days. The MAPE of Forecast.Solar rises from 66.7% to 150.1% for

the Mühlviertel location and from 55.0% to 157.3% for the Traunviertel location; the MAPE of Meteoblue rises from 36.9% to 158.4% and from 26.9% to 114.7%, respectively. The nMAE, which is not inflated by single hours with very low production, shows the same ordering; for example, from 16.5% to 42.2% for Meteoblue at the Mühlviertel location, so this increase is not only an artifact of the MAPE. The share of hours with a percentage error above 200% rises from at most 5% on clear days to between 14% and 20% on overcast days.

The absolute error does not follow this pattern. The MAE of Meteoblue stays between about 870 and 1180 W in all categories, and Forecast.Solar even has its highest MAE on clear days (2790 W and 1760 W). The bias explains this: on clear days, Forecast.Solar underestimates production substantially, by 2135 W and 1465 W on average, while this underestimation largely disappears on overcast days. Meteoblue, in contrast, overestimates production on overcast days by about 590 to 770 W on average. Meteoblue is more accurate than Forecast.Solar in every category in terms of MAE and nMAE, but the difference is largest on clear days and smaller on overcast days. For the Mühlviertel location, Meteoblue's MAPE on overcast days is even slightly higher than that of Forecast.Solar.

These results carry some uncertainty, since each category contains only 15 to 23 days per location. Resampling the days within each category (bootstrap, 95% interval) gives, for example, a range from 113% to 203% for Meteoblue's MAPE on overcast days at the Mühlviertel location. Classifying days into three equally sized groups by cloud cover, rather than using fixed thresholds, shows the same pattern. Overall, the data supports that the forecasts are relatively less accurate on cloudy and overcast days than on clear days, but not that they become unusable on such days in general: an average percentage error above 200% occurs on individual days, such as the examples in the next section, but not for a weather category as a whole.

3.3.4 What Characterizes the Best and Worst Days?

Forecast.Solar Forecast.Solar is the cheaper alternative, costing only 17 EUR per year for a private person or 70 EUR per year for businesses. Figure 3.6 shows a representative example of a poor forecast in percentage terms: the graph for the 4th of April 2026, a rainy day with heavy cloud cover and almost no measured sunshine. The API overestimates the production of solar power by a lot on this day, following a smooth, unimodal curve while the actual production stays far more irregular and well below the forecast for almost the entire day; the deviation was around 1270 to 1520 watts for Mean, median, and Root Mean Squared error, and 172% between the system's actual and forecasted production. The 1h, 3h, and 6h forecasts for this day, shown in Figure 3.7, are almost identical.

The second example, shown in Figure 3.8 and Figure 3.9, is from the 6th of March, the clearest day of the evaluation period, with 0% average cloud cover at both locations. The forecast underestimates production during the morning and the afternoon, while briefly overestimating it around noon. The Mean, median, and Root Mean Squared error are 2476W, 1981W, and 3064W, respectively. Percentage-wise, the deviation is 63%, still substantial, but a lot better than the 172% on the rainy day above.

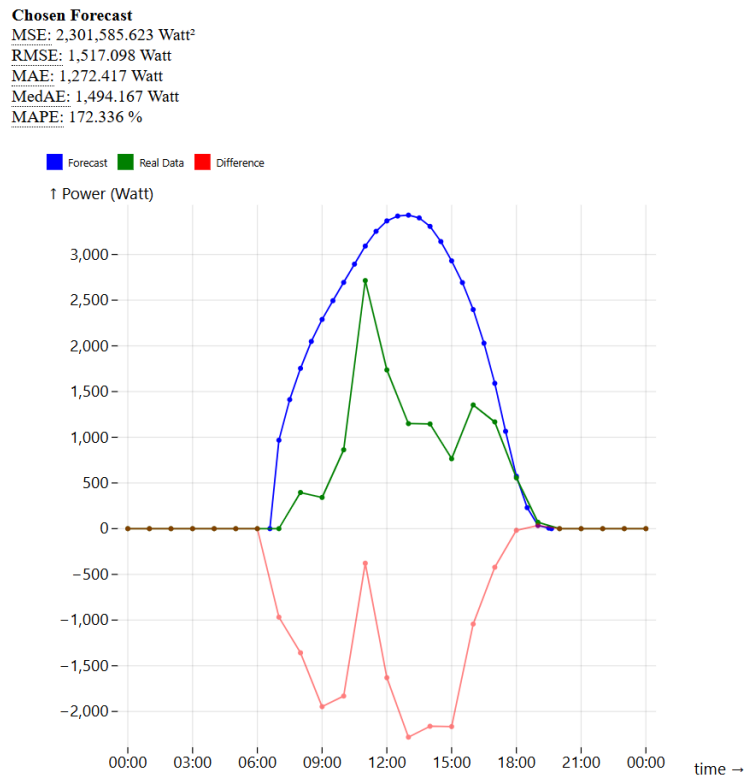


Figure 3.6: Forecast of Forecast.Solar on a rainy day (4 April, Mühlviertel location): forecasted and actual production and their difference.

1h forecast	3h forecast	6h forecast
MSE: 2,352,086.429 Watt ²	MSE: 2,381,796.346 Watt ²	MSE: 2,373,468.540 Watt ²
RMSE: 1,533.651 Watt	RMSE: 1,543.307 Watt	RMSE: 1,540.607 Watt
MAE: 1,298.000 Watt	MAE: 1,296.083 Watt	MAE: 1,294.833 Watt
MedAE: 1,525.667 Watt	MedAE: 1,531.667 Watt	MedAE: 1,526.667 Watt
MAPE: 174.479 %	MAPE: 175.408 %	MAPE: 174.584 %

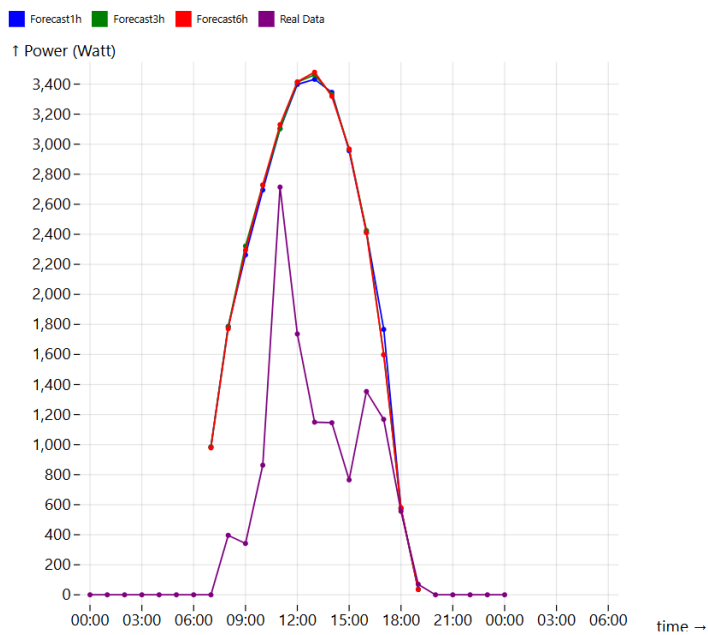


Figure 3.7: 1h, 3h, and 6h forecasts of Forecast.Solar compared to the actual production on the same rainy day (4 April, Mühlviertel location).

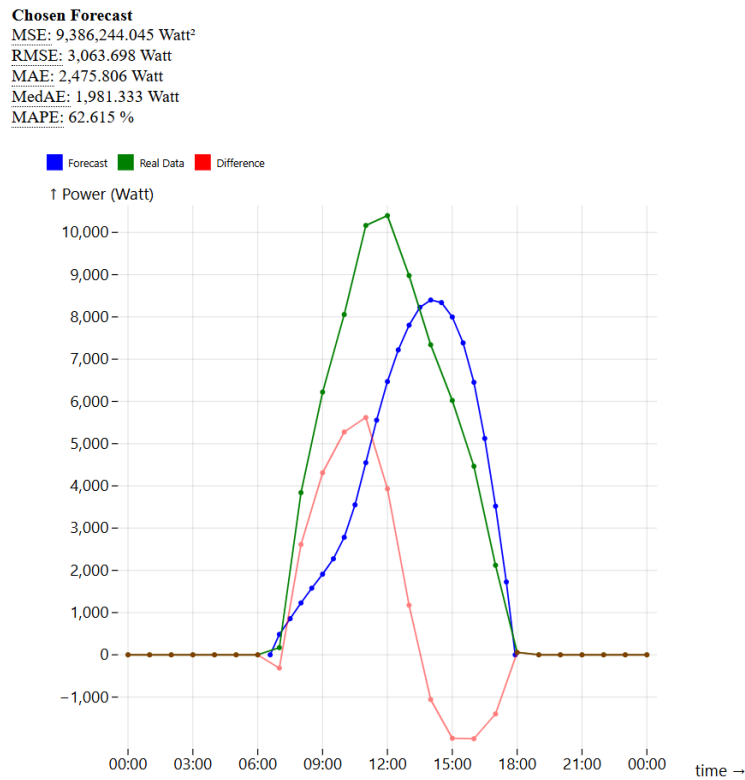


Figure 3.8: Forecast of Forecast.Solar on a sunny day (6 March, Mühlviertel location): forecasted and actual production and their difference.

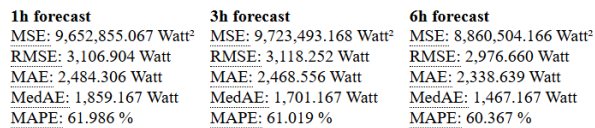
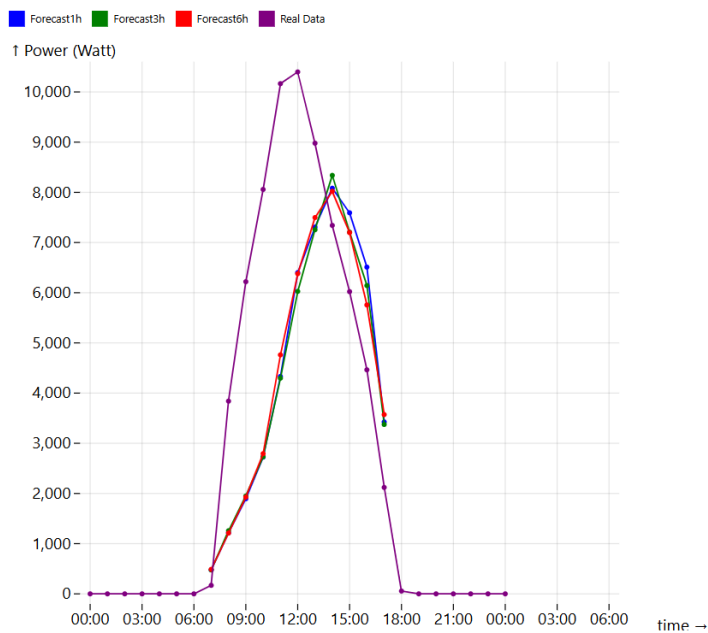


Figure 3.9: 1h, 3h, and 6h forecasts of Forecast.Solar compared to the actual production on the same sunny day (6 March, Mühlviertel location).

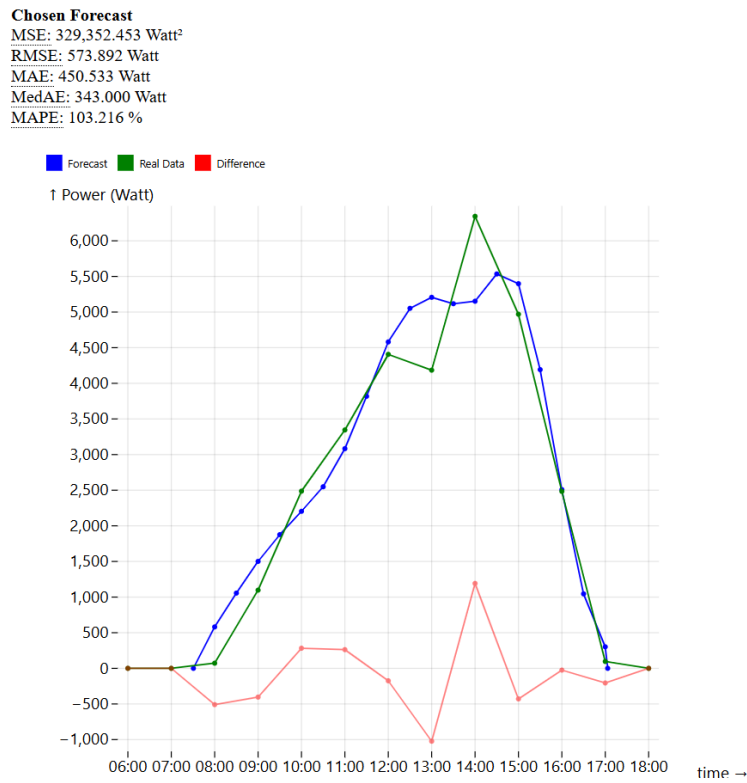


Figure 3.10: Lowest mean-absolute-error result of Forecast.Solar in the evaluation period for the location in the Mühlviertel (2 February).

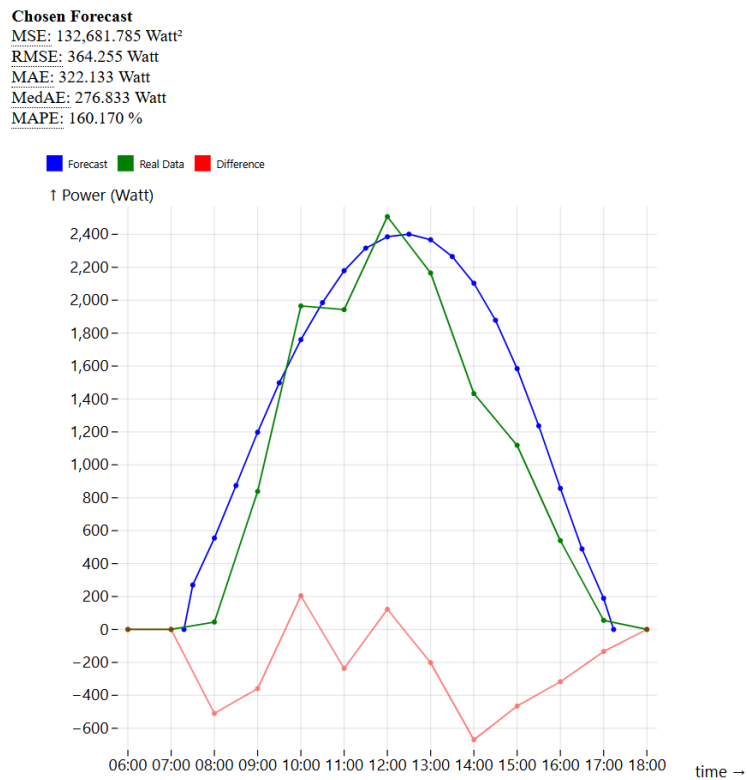


Figure 3.11: Lowest mean-absolute-error result of Forecast.Solar in the evaluation period for the location in the Traunviertel (9 February).

The last example for Forecast.Solar, shown in Figure 3.10 and Figure 3.11, shows the two days with the lowest mean absolute error at each location across the whole evaluation period, which do not fall on the same date: the 2nd of February for the Mühlviertel location and the 9th of February for the Traunviertel location. In absolute terms, these are the closest either API got to the actual production at these locations, with a mean absolute error of 451W and 322W, respectively. In percentage terms, however, both days still show large deviations, at 103% and 160% respectively: both fall early in the year, when overall production is seasonally low, so even a small absolute miss translates into a large percentage error. Because the two locations have different dimensions, the raw numbers cannot be compared directly, only within the same location.

Meteoblue Meteoblue is the more expensive one of the tested APIs, working with a token-based payment method. Figure 3.12 shows data for the same rainy day, the 4th of April, for Meteoblue. As shown in the graph, it generally overestimates production, while often still capturing the general shape. With a mean deviation of 995W and a median of 892W, it is somewhat closer than Forecast.Solar in absolute terms on this day, but its percentage deviation (211%) is higher than Forecast.Solar's 172% for the same day, a first illustration of how MAE and MAPE can disagree about which API looks better, discussed further below. For this API, the forecasts for 6h and 3h, shown in Figure 3.13, are unfortunately not as meaningful, since we had to reduce the amounts of queries because of cost reasons for the data collection period.

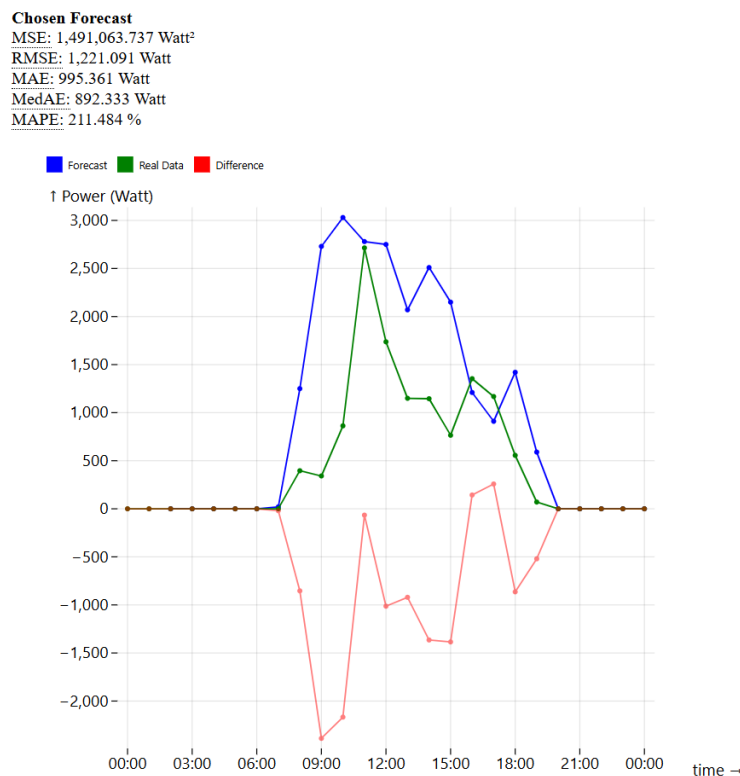


Figure 3.12: Forecast of Meteoblue on a rainy day (4 April, Mühlviertel location): forecasted and actual production and their difference.

Another picture is drawn from the forecast on the same sunny day as for Fore-

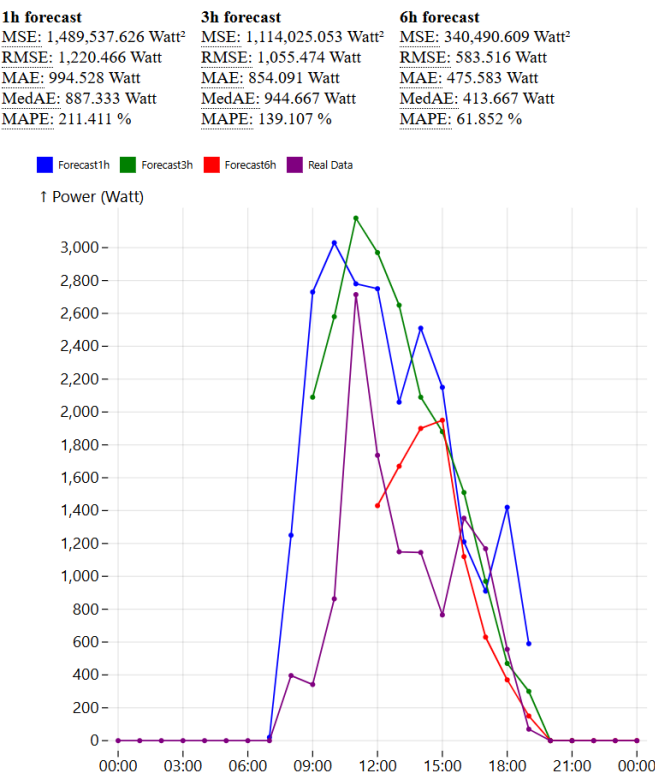


Figure 3.13: 1h, 3h, and 6h forecasts of Meteoblue compared to the actual production on the same rainy day (4 April, Mühlviertel location).

cast.Solar, the 6th of March. With a mean deviation of only 512W and a median of 436W, this is a percentage difference of just 28%, distinctly better than Forecast.Solar’s 63% on the same day. The forecast underestimates production slightly around midday but otherwise closely tracks actual production, as shown in Figure 3.15.

The two best days for Meteoblue by mean absolute error, shown in Figure 3.16 and Figure 3.17, were the 6th of March for the Mühlviertel location, the same day already discussed above, and the 10th of March for the Traunviertel location. The median difference was 436W and 426W, respectively, both under 30% in percentage terms (28% and 19%). To put this in perspective, the household for the Mühlviertel location has an average base consumption of about 500W (around 12kWh per day) on an unremarkable day without EV charging. A deviation of over 400W is therefore still not negligible, comparable in magnitude to the household’s baseline load, or to a large appliance such as a heat pump.

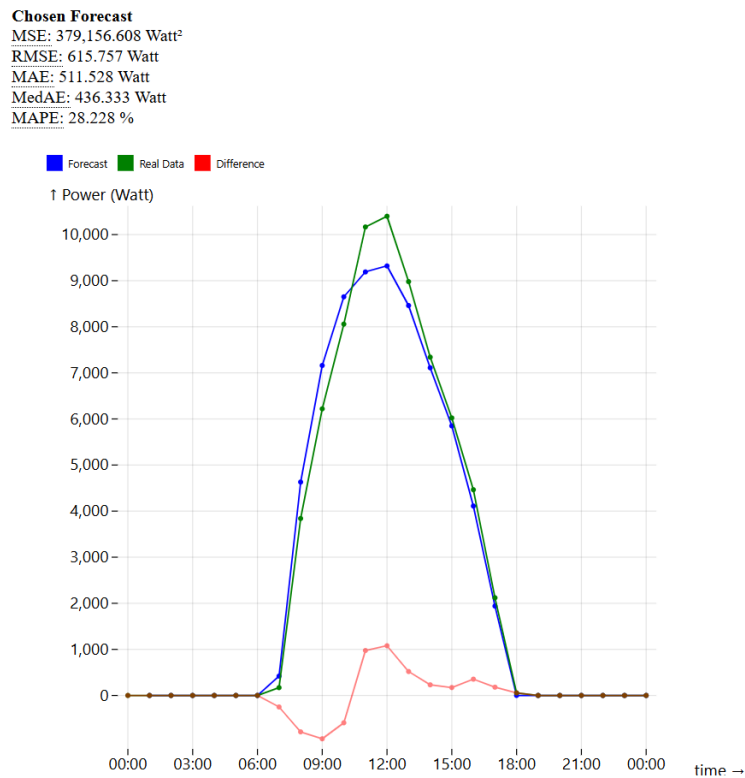


Figure 3.14: Forecast of Meteoblue on a sunny day (6 March, Mühlviertel location): forecasted and actual production and their difference.

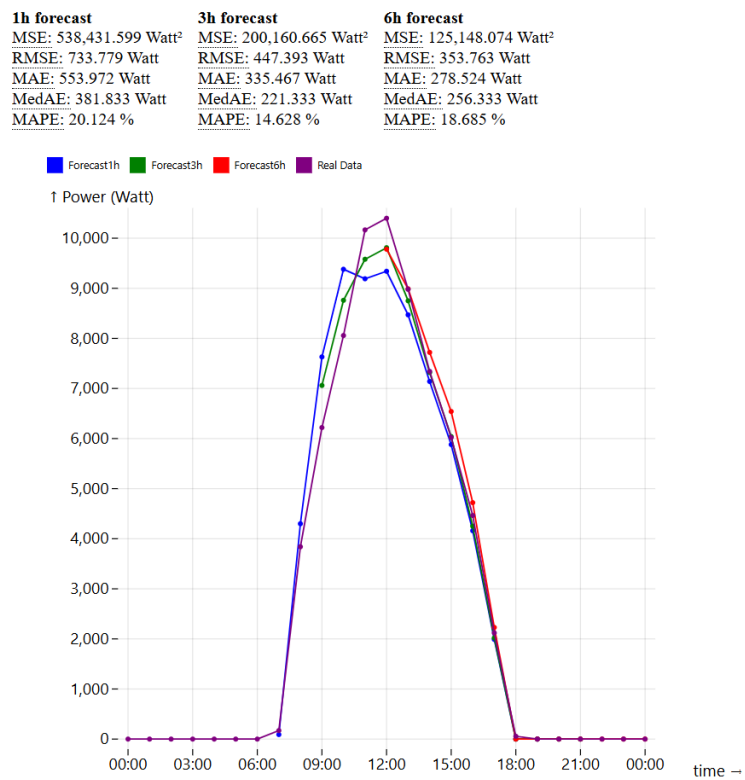


Figure 3.15: 1h, 3h, and 6h forecasts of Meteoblue compared to the actual production on the same sunny day (6 March, Mühlviertel location).

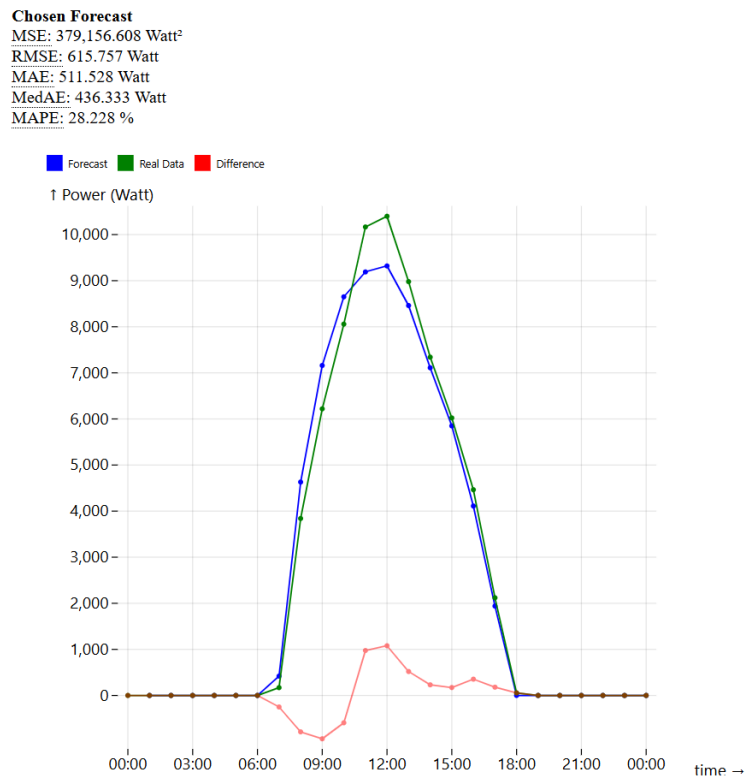


Figure 3.16: Lowest mean-absolute-error result of Meteoblue in the evaluation period for the location in the Mühlviertel (6 March).

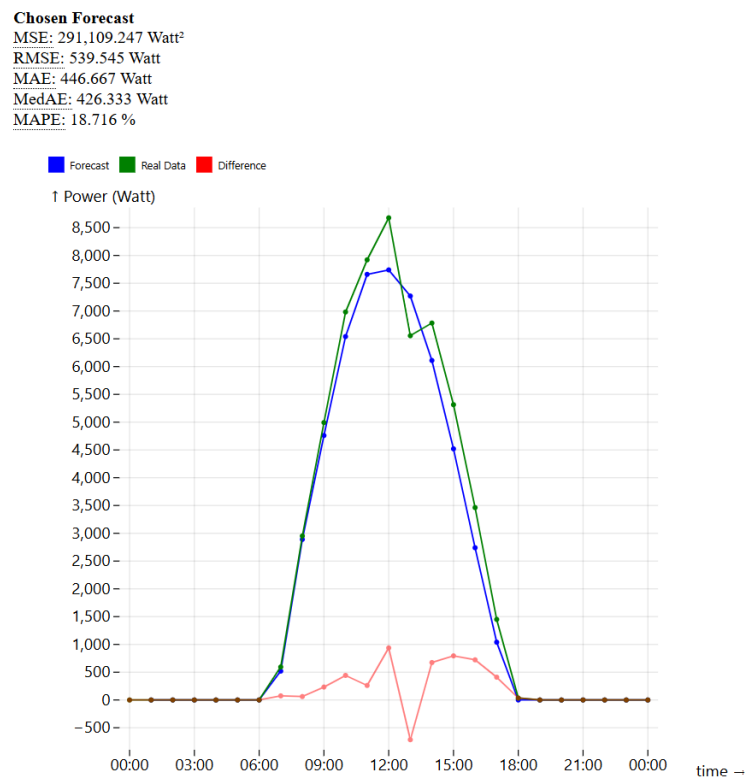


Figure 3.17: Lowest mean-absolute-error result of Meteoblue in the evaluation period for the location in the Traunviertel (10 March).

A systematic look at outliers The individual days above were chosen while writing this thesis because they looked representative; the full February-to-May period was also searched systematically for the days with the highest and lowest mean absolute error (MAE), for each location and API, and the historical weather for those specific days was looked up afterward on Open-Meteo [16] to explain them.

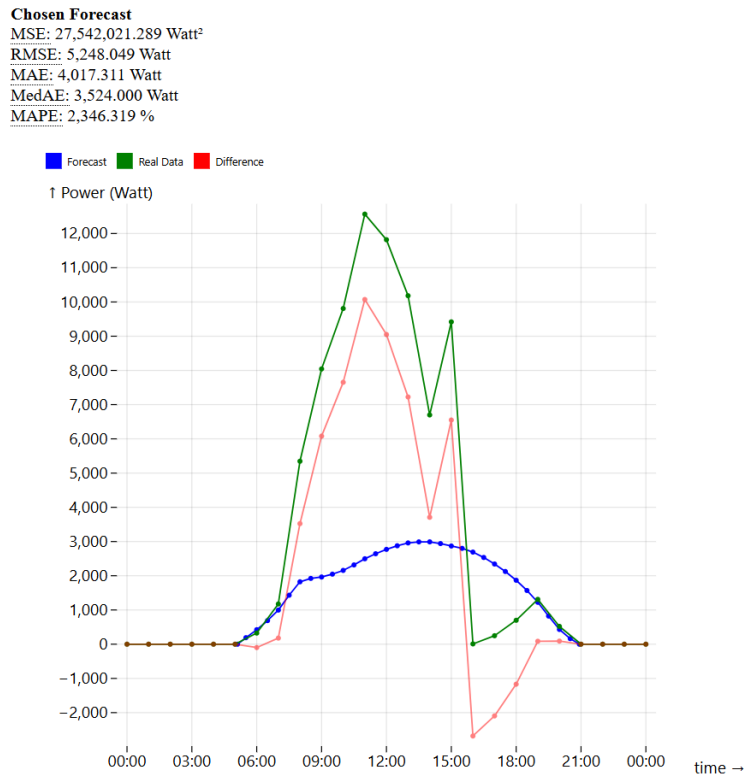


Figure 3.18: Highest mean-absolute-error result of Forecast.Solar in the evaluation period for the location in the Mühlviertel (31 May).

For Forecast.Solar, the 31st of May ranked among the three worst days by MAE at both locations at once, shown in Figure 3.18 and Figure 3.19: a mean absolute error of 4017W for the Mühlviertel location and 2818W for the Traunviertel location, by far the largest absolute errors found anywhere in the collection period. The percentage error for the Mühlviertel location on this day, 2346%, is itself a good illustration of MAPE's instability described earlier in this section: production briefly dropped close to zero in the afternoon while a small forecasted value remained, blowing up the percentage far beyond what the graph itself suggests. The historical weather for 31 May was only moderately cloudy (74–75% average cloud cover, around 10 hours of sunshine, a few millimeters of rain at both locations), not a clear day, but late enough in the year that even partial sun still produced substantial power, over 5000W on average during the hours evaluated at the Mühlviertel location. The best (lowest MAE) days, by contrast, cluster earlier in the year: the 2nd of February, the best day for the Mühlviertel location discussed above, falls in a period when production is seasonally much lower overall; 26 March, another of the best days for the same location, was almost completely overcast (98–100% cloud cover, no measurable sunshine) with a small amount of rain. Both point to the same underlying effect: the absolute error scales with how much power is actually being pro-

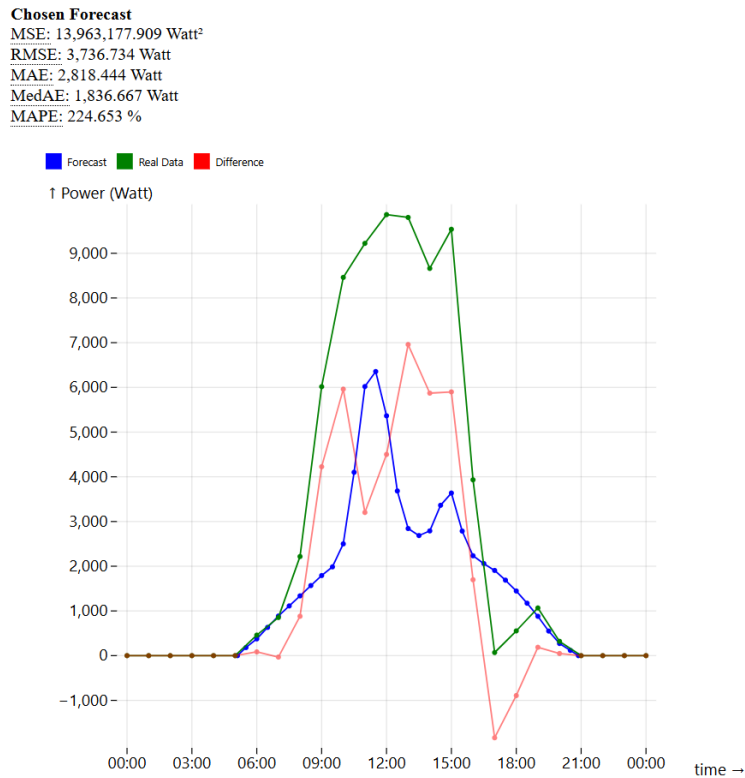


Figure 3.19: Highest mean-absolute-error result of Forecast.Solar in the evaluation period for the location in the Traunviertel (31 May).

duced, so days, or seasons, with high production leave much more room for a large absolute miss in watts, while low-production days or months leave little room for a large absolute error, even when the API's estimate is percentage-wise far off. This is the mirror image of the percentage-based pattern shown by the individual rainy/sunny-day examples above and by the weather categories in Section 3.3.3, where the relative error (MAPE) was largest on rainy or overcast days and smallest on sunny days; both patterns are consequences of the same underlying effect, just measured in absolute watts versus relative percent. It also means that, on a single day, MAE and MAPE can disagree about which API looks better, as already seen on the rainy day above.

Meteoblue's worst day was the 14th of April at both locations, shown in Figure 3.20 and Figure 3.21: a mean absolute error of 2493W for the Mühlviertel location and 2201W for the Traunviertel location, on a day with close to no sunshine (2–2.4 hours) and near-total cloud cover (96–99%). Unlike Forecast.Solar, whose largest absolute misses occur on the highest-production days, Meteoblue's largest absolute miss occurs on a genuinely overcast, low-production day, suggesting the two APIs do not fail in the same way, and that neither a pure weather label nor a single metric is enough to say which API is more reliable.

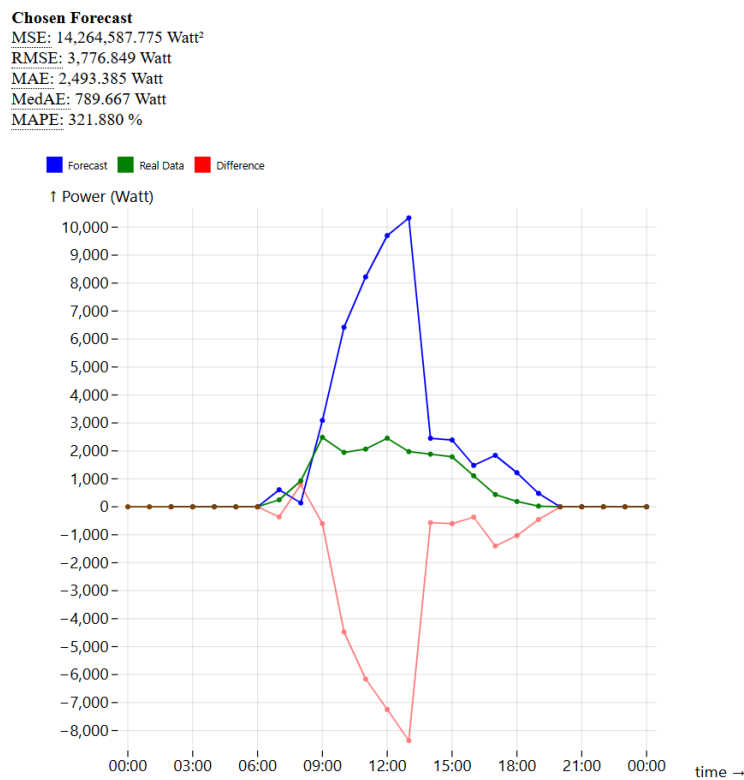


Figure 3.20: Highest mean-absolute-error result of Meteoblue in the evaluation period for the location in the Mühlviertel (14 April).

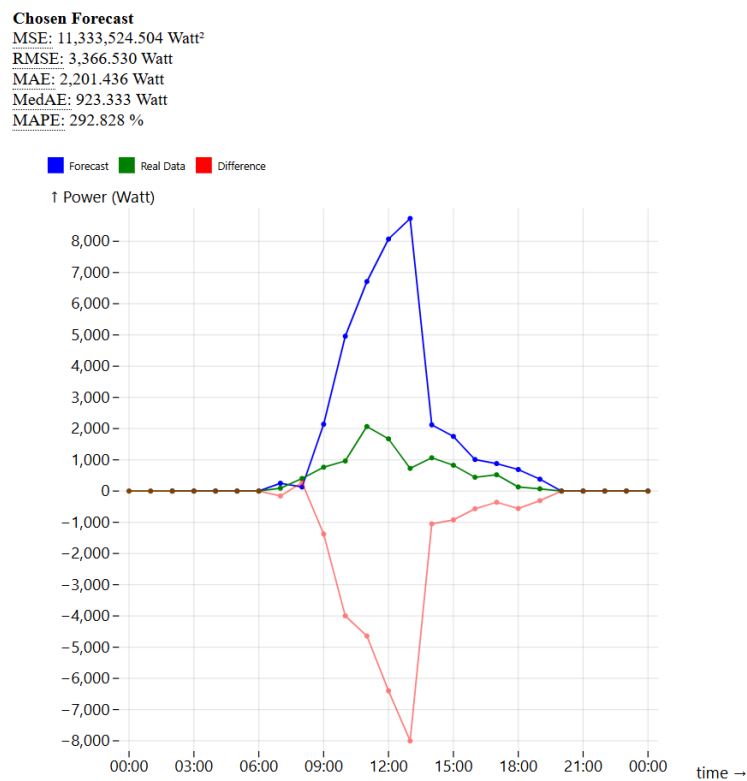


Figure 3.21: Highest mean absolute error result of Meteoblue in the evaluation period for the location in the Traunviertel (14 April).

3.3.5 Summary and Discussion of Findings

Overall, the aggregates in Table 3.1 show that neither API is precise enough on average: a MAPE between 63% and 102% for the 1h forecasts means the typical forecast error is on the same order of magnitude as the actual production it is trying to predict. On the shared hours, Meteoblue is consistently more accurate than Forecast.Solar, but even its MAE of around 900 to 1100 W is well above a typical household base load. Individual days can look much better, particularly high-production days evaluated by absolute error, or low-production days by percentage error, but Section 3.3.4 shows that this depends strongly on the metric used for the day. Grouped by weather (Section 3.3.3), the relative error of both APIs increases from clear to overcast days, from a MAPE of 27% to 67% on clear days to 115% to 158% on overcast days, while the absolute error does not increase. Querying more often (Section 3.3.2) also does not meaningfully help within the lead times tested here. The results also differ between the two locations. This indicates that forecast quality may vary substantially even between geographically close locations. Evaluating additional locations would be necessary to determine how broadly this observation is true.

This means that these APIs are not precise enough to be used in a production environment. This can be mitigated to some extent by using a home battery to charge excess power and use it when the prediction is lower than the real data, but most often, if a home battery is used, the possibility of connecting the wallbox is also given, and this feature is not needed.

Since the data collection period lasted over multiple months, the backend and frontend microservices were still developed independently before the results were available. Because there was no final choice for an API during development, or in this case even afterward, a big focus was put on making it API-agnostic.

Chapter 4

System Architecture

This chapter describes the overall architecture of the prototype built for this thesis, and how it would need to change to become part of the existing KEBA eMobility Portal.

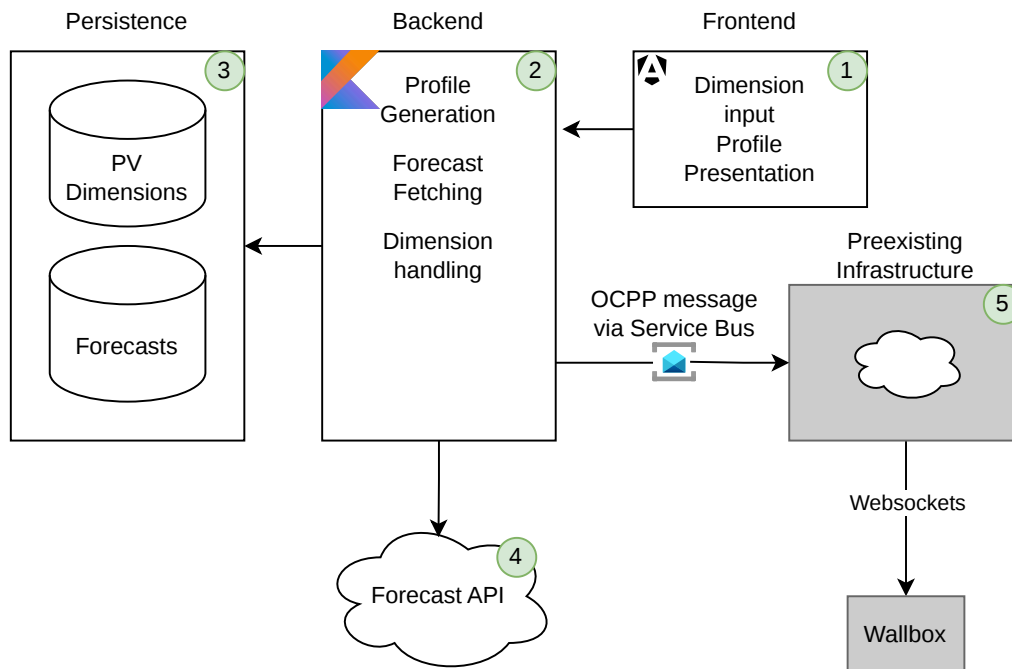


Figure 4.1: Architecture of the resulting module: (1) the frontend, which can be integrated into the existing eMobility Portal, (2) the backend running standalone, (3) the persistence layer, which would need to be added to the existing database, (4) a future API returning usable data, and (5) all the pre-existing infrastructure, which is not touched by this thesis.

Figure 4.1 shows the architecture of the resulting module. The system is designed to be only loosely coupled with the existing infrastructure via asynchronous messaging. The rest of the components (1 through 3) can be deployed as their own microservice. While this is acceptable for this prototype, deploying it to the eMobility Portal would require two further changes: the data that is currently entered manually would need to come from other existing services, and the frontend would need to be integrated into the main application rather than run as a standalone page.

To make the scope of this thesis explicit: components (1) to (3), the frontend, the backend, and the persistence layer, were implemented as part of this thesis. Component (5) represents existing KEBA infrastructure, such as the Service Bus and the service that forwards charging profiles to the wallbox, which is used as-is and was not modified. Component (4) is currently only an abstraction for a future forecasting provider: the backend accesses forecasts through an exchangeable interface, but none of the APIs evaluated in Chapter 3 was found to be suitable as this provider.

Chapter 5

Frontend

The frontend consists of a single-page Angular [6] application. This technology was chosen because the existing eMobility Portal is written entirely in Angular, allowing this part to integrate easily with and be iterated upon within the existing product. Other than input validation, nothing is calculated in the frontend; data is only presented. The page can be split into two distinct parts: the first loads previously entered dimensions and provides input fields for updating them or creating new ones. The second part visualizes the resulting charging profiles, both graphically and as raw JSON, for evaluation purposes. The frontend communicates with the backend via simple REST endpoints.

Choose existing config

69df2cd688fadd1ba57a4ac New

Input your configuration



Latitude
48.317536 °

Longitude
14.288687 °

Locate Me

Base Consumption
0 W

Minimum PV portion
0 %

Configuration 1

Peak power
0 kW

Azimuth
0 °

Slope
0 °

+

Save

Figure 5.1: Input part of the frontend, showing old configs and a map input (left), and input fields plus a dynamic array of sub-panels configurable for different directions (right).

Dimension input Figure 5.1 shows the dimension input part of the frontend. The list at the top is populated by data fetched from the backend. The more interesting parts are the inputs. The location is chosen via a map built with Leaflet [9], an open-source mapping library, with OpenStreetMap [17] as the

map provider. The user can either click on their home, let the browser locate them, or enter latitude and longitude directly. The map and the input fields are kept in sync. The other data needed for calculating profiles were *Base consumption* and *Minimum PV portion*. These tell the backend later what the household uses as a baseline and how much power can still be bought from the grid to reach the fastest possible charging rate. While this is a workaround, there is no real way of knowing the current power draw in the household from the perspective of the backend, and since the forecast data will never be 100% accurate, this was seen as an acceptable drawback. The amount of power that must come from the PV system is modeled the same way as normal PV charging.

The input data for the forecasts has to be dynamic, since PV systems can have arbitrarily many sub-groups. Each input field shows its unit of measurement, so it is clear that, for example, the peak power is expected in kW, the base consumption in W, and the orientation of a sub-group in degrees.

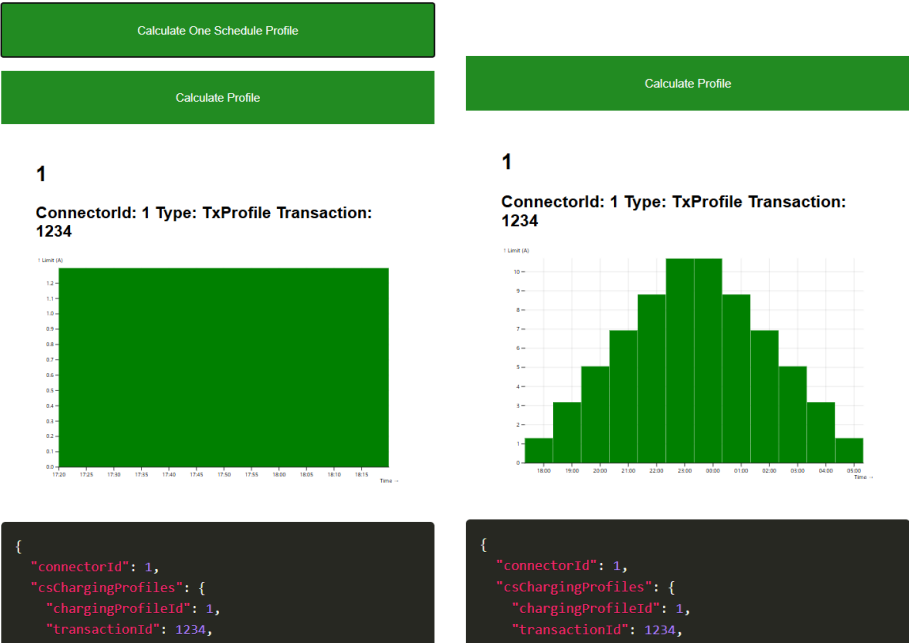


Figure 5.2: Lower part of the frontend with schedules calculated from dummy data.

Schedules Figure 5.2 shows the two calculated schedules. The first schedule needed is a single-hour schedule. This can be generated every hour and sent to the wallbox, or, in this case, shown to the user. The view shows all important information of the profile: the profile ID, the connector ID, the type, and the transaction ID. The graph, a bar chart made with Observable Plot [14], shows one bar per period of the schedule, with its height indicating the charging limit in amperes. Since the tested APIs return hourly or 30-minute data, the single-hour schedule consists of only one or two periods. In the longer schedule, the bars follow the shape of the PV forecast, so the reader can see at a glance during which hours the car will charge the most.

The user inputs directly scale these bars, following the calculation described in Chapter 6. For example, a forecasted PV power of 5000 W in a one-hour slot

and a base consumption of 500 W leave a surplus of 4500 W. With a minimum PV portion of 100%, the car can charge only from this surplus, resulting in a limit of about 6.5 A for a three-phase charge. Lowering the minimum PV portion to 50% allows the same amount to be drawn from the grid, doubling the bar to about 13 A, while raising the base consumption to 1000 W at 50% lowers it to about 11.6 A. Below the graph is the full JSON representation sent to the wallbox via the OCPP backend. This visualization uses Prism [19] for syntax highlighting to make it easier to read at a glance.

The second schedule is a schedule based on the longer forecast. Since every call to a paid forecasting API costs money, the hourly schedule could be more expensive but also more reliable since the data is queried near the start of the transaction. To mitigate this, the forecasting schedule makes fewer calls to the forecast API. But this not only saves money, but it can also be shown to the user to give them a general overview of when the car will charge the most, and they can plan their outings accordingly.

The underlying graph and the JSON output show the data dynamically, so the logic is the same for both.

Chapter 6

Backend

The backend is written in Kotlin [7] and uses Quarkus [20]. The author was already familiar with Kotlin and worked with it professionally, but had not yet combined it with Quarkus, making this pairing a welcome learning opportunity as well as a more modern choice of technology. Although the existing backend uses Java for each component, this does not pose a real problem. If configured correctly, Java and Kotlin code can be used interchangeably. Since this is a separate microservice, the other systems will not access this backend except for REST calls or Service Bus messages, while shared libraries written in Java can be easily used by the new backend.

Quarkus was chosen because it is designed for running Java and Kotlin applications as containerized microservices [20]. In contrast to Spring, which the existing backends use and which resolves dependency injection at runtime, Quarkus still uses core concepts like beans and proxies but performs most of this wiring at build time. According to its documentation, this reduces startup time and memory usage [20], which would allow the backend to be started on demand instead of running permanently. For this thesis, startup time was not a deciding factor; however, the existing services already use different technologies: while most use Spring, some use Azure Functions or no framework at all. All libraries for communicating with Azure resources have Quarkus-optimized libraries ready, and older libraries can also easily be used. Quarkus can additionally be used for serverless deployments and offers a good development experience through live reload and dev services, both of which sped up development of the prototype.

6.1 Architecture

The backend has three functionalities: Fetching forecasts on a schedule, persisting data for the frontend, and calculating charging profiles.

Configuration The configurations are split into two collections, since the dimensions are not needed for calculating the charging profile and the base load and PV percentage are not needed for fetching forecasts. There is not much logic in this part, since it's a simple REST endpoint responsible for CRUD operations on the configurations.

Forecast fetching Forecast fetching runs in the background, but can also be triggered when a new request comes in. This is done so that, depending on how expensive the future chosen API will be, the amount and time of fetching can be adjusted. This service calls only a `forecastService`, which can be exchanged depending on the chosen provider. The forecasting data is then saved to a database in case the transaction is canceled and restarted. In the current implementation, it considers all locations and fetches them periodically.

Charging profile generation This part of the backend generates the profiles. It has two entry points: one for forecasting and one for a single hour, which in turn point to the same logic. Generating the profile is quite straightforward. It takes the forecast for the profile location, queried via the watt-hour endpoint rather than the watt endpoint used for evaluation in Chapter 3, since building a schedule needs the energy per time slot to handle slots of different lengths. For every forecast item, starting at the current time, the charging limit is then calculated in four steps:

$$\begin{aligned} P_{PV} &= \frac{E_{\text{forecast}}}{\Delta t} & P_{\text{surplus}} &= \max(0, P_{PV} - P_{\text{base}}) \\ P_{\text{limit}} &= \frac{P_{\text{surplus}}}{p_{\text{min}}} & I_{\text{limit}} &= \frac{P_{\text{limit}}}{230 \text{ V} \cdot n_{\text{phases}}} \end{aligned}$$

First, divide the forecasted energy E_{forecast} for a slot by the slot length Δt , which depends on the granularity of the source, to get the average PV power P_{PV} . Subtracting the base consumption P_{base} of the household gives the expected surplus. This surplus is then divided by the minimum PV portion p_{min} configured by the user, so that the PV surplus makes up exactly this share of the charging power, while the rest may be drawn from the grid. Lastly, the limit is converted into amperes for the profile using a voltage of 230 V and the number of phases the car charges with. This would also work with watts, but amperes is the recommended unit. As an example, a forecast of 5000 Wh for a one-hour slot gives an average PV power of 5000 W. With a base consumption of 500 W, the surplus is 4500 W. A minimum PV portion of 50% doubles this to a charging limit of 9000 W, of which 4500 W comes from the grid, which corresponds to $9000 \text{ W} / (230 \text{ V} \cdot 3) \approx 13 \text{ A}$ for a three-phase charge. With a minimum PV portion of 100%, the car would only charge the surplus of 4500 W, or about 6.5 A. After this calculation, the profile is built according to the OCPP standard using the official library and sent back. When deployed, it will send the profile to the asynchronous messaging service instead of the REST endpoint. An already existing microservice picks up this message and forwards it to the wallbox over the existing WebSocket connection, where the wallbox then accepts or rejects the profile.

6.2 Data models

Figure 6.1 shows the backend data model. All data for the backend is saved in a document database. This decision was made because every `PVDimensions` object contains an arbitrary number of `PVParts` and a structured location. This is

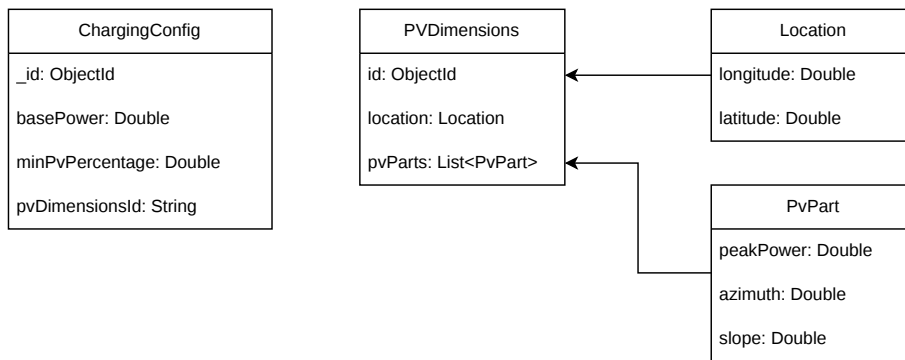


Figure 6.1: Data model of the backend: PVDimensions contains multiple Objects but is saved in the same object. ChargingConfig contains the key of the associated Dimensions to couple them.

saved as a single object to make the table structure easier to understand. CosmosDB is also cheaper to run than an SQL server, saving unnecessary costs.

6.3 Challenges

The first problem encountered with the development of the backend is a well-known issue with using Kotlin with frameworks like Quarkus and Spring. Quarkus needs to create proxies of classes in order to intercept some calls, but Kotlin makes every class non-inheritable by default. As a programmer, you could just make every class open. This overhead can be eliminated by using the Gradle plugin `allopen`. Another challenge was that, while the OCPP library still gets updates, it had some incompatible dependencies, specifically the StAX (Streaming API for XML) dependency. This is an old XML library, included in the JVM since Java 6. This dependency led to issues because the dependency from OCPP shadowed the default implementation and led to `ClassNotFoundException` errors being thrown on startup.

Chapter 7

Challenges and Future Work

7.1 Challenges and Limitations

While the challenges during development of the microservices were easily solvable, the challenges encountered during data collection were harder to overcome. The biggest hurdle was obtaining usable real-world data. Although many volunteers were available, only a few could be used for real-world data collection. The scope and capabilities of the SolarEdge API were already well known from prior work, but other platforms either did not have a usable API for automated gathering or kept it behind a paywall. This is also the reason why fewer real-world locations were available than anticipated. The next hurdle concerned the API providers themselves. Forecast.Solar was a known provider going into the thesis, and while its data was expected to be somewhat inaccurate, the extent of the deviation was still greater than expected. Meteoblue was fairly pleasant to work with, but since a large volume of data was needed around the clock, its pricing proved too steep.

These challenges also limit how far the evaluation results in Chapter 3 can be generalized. Only two PV systems were evaluated, both located in Upper Austria and only about 47 km apart, so the results are not necessarily the same for other regions. The data available for the two APIs also differs: Forecast.Solar was queried around the clock, while Meteoblue was only queried during the day and only from 26 February to 24 April 2026, until the purchased tokens were used up. The direct comparison of both APIs is therefore restricted to the roughly 690 hours per location for which both provided a forecast. The evaluation period covers only about four months, from February through May 2026, and for the API comparison only about two months, and therefore does not include the summer months with the highest production. Furthermore, the evaluation by weather category is based on only 15 to 23 days per category and location, and on a daily cloud cover value that does not capture changes in the weather during the day, so its results are subject to some uncertainty. Finally, the evaluation measures the accuracy of the forecasts themselves, which is used as a substitute for the expected quality of the resulting charging schedules. The actual effect of the schedules on grid consumption, PV self-consumption, or charging performance was not evaluated quantitatively, and doing so would be a valuable addition for future work.

7.2 Future work

Training a custom model With the data collected during the thesis duration, an in-house model can be trained for this purpose. As shown in other papers discussed in Chapter 8, specially trained models are not only feasible in theory but have been shown to be effective. While most research has focused on single-site forecasting, this can also be tried for multi-site deployments like with the KEBA wallbox. For historical data, there are also a lot of APIs to fetch historical weather and irradiance data needed for this model.

Evaluation of Future Forecasting APIs The current data collection and calculation backend are designed to be easily extensible, making it easy to retry this experiment in the future: once new APIs get released, already tested ones get expanded or refined, or governmental ones that were previously offline get re-activated. Only a new source needs to be added and mapped to the existing data model. The rest stays the same, and if a better fit is found, only the data fetching needs a new implementation of an interface, with no new data model needed.

Chapter 8

Related Work

8.1 A review and evaluation of the state-of-the-art in PV solar power forecasting: Techniques and optimization

Overview Ahmed et al. [2] provide a comprehensive review of state-of-the-art PV forecasting and conclude that ensembles of artificial neural networks achieve RMSE and MAPE values between 5–20% for short-term forecasts, the same forecast horizon evaluated in this thesis. They also conclude that solar irradiance, an easily attainable metric, has a high correlation for such models. Based on this, one would expect to receive comparatively good results from APIs, at least for the timeframe of our experiments. But the results we got paint a less optimistic picture, with an average relative error between 63% and 102%, rising to more than 100% on overcast days, and absolute errors not negligible for a typical household base load.

Explanation A likely explanation is that the models reviewed by Ahmed et al. were typically trained on data from a single site, captured by research-grade sensors. The APIs used for this thesis, on the other hand, are generic, widely available forecasting services that can be used in arbitrarily chosen locations, which is what a commercial product with customers at many different locations needs. Ahmed et al. also describe this transfer in Section 8 of their review as an open research topic rather than a solved one, so high accuracy reported for specific research settings cannot be expected to carry over directly to generic services like those evaluated in this thesis.

Conclusion Notably, Ahmed et al. also point out in one of their references that pure statistical error metrics are often insufficient and that application-specific evaluation is needed instead, a claim directly reinforcing our conclusion. Even on the best example days, with a relative error below 30%, the absolute error remained large relative to typical household consumption, which undermines confidence in the resulting charging schedule. For the two APIs evaluated here, this suggests that their accuracy and their price are not yet sufficient for this use case, even though the results for site-specific models are considerably more optimistic.

8.2 Charge Scheduling Optimization of Plug-In Electric Vehicle in a PV Powered Grid-Connected Charging Station Based on Day-Ahead Solar Energy Forecasting in Australia

Overview Sheik Mohammed et al. [22] took a forecasting approach directly for solar-powered charging stations, training their own model for day-ahead forecasts in a single location in New South Wales, Australia. Their model reaches a prediction efficiency of 99.61% and a MAPE of only 7.9% for irradiance. With this near-perfect forecast and their scheduling algorithm, the authors report that their charging cost was reduced by 10–20% using grid power alone and by 50–100% when used with a solar PV system. With this, a 65kW charging station is projected to become net-zero and even generate an annual profit.

Explanation This is in stark contrast to the findings in this thesis again, but this stems from the same reason as in [2]. The authors trained their model for a single, fixed location and they used 70% of solar irradiance data for training, leaving the rest for testing, both from a historical dataset. Their evaluation therefore shows how well the model performs on held-out data from the same site and period, and it could behave differently under less predictable weather. The main difference from this thesis is not that one approach has to handle future, unknown weather and the other does not, since a deployed version of their model would face the same requirement. Instead, the difference lies in scope: their model is custom-made for one specific location, while the APIs evaluated in this thesis, and any solution for the KEBA eMobility Portal, need to work across many different locations without site-specific training. The results of Sheik Mohammed et al. show that high accuracy is achievable for a single site, but they do not directly indicate how well such an approach transfers to a geographically distributed set of small installations.

Conclusion The authors paint a picture in which near-perfect forecasts are achievable, highlighting the economic gains and concluding that this optimization cannot only be cost-neutral but even profitable. Their results show this for a single, large charging station with a model trained on historical data from that site. Whether comparable results can be achieved for a distributed network of small home charging stations, as in the KEBA eMobility Portal, is not addressed by their work: training one model per home would require historical data for every location, and whether generalized or regional models could close this gap remains an open question.

8.3 Forecast-based Optimal Operation of EV Charging Station with PV Considering Charging Demand and Distributed System

Overview Rho et al. [21] present a forecast-based energy management system for a PV-connected charging station, forecasting three different values: PV generation, regional load demand, and EV charging demand. These forecasts feed a program that schedules EV charging and discharging to minimize operation cost and maximize PV self-consumption, evaluated on real fast-charging data. The authors report PV self-consumption rising from around 30% to 68% and operating cost falling, ultimately turning a net profit.

Explanation As with the previous sources, the strength lies in training data being from a single location and a well-characterized EV fleet. The historical training data led to learning local weather patterns, a strength systems like the KEBA eMobility Portal cannot easily use. Such amounts of historical PV, load, and EV charging data are usually not available for every location a wallbox is deployed at, and training a model per location would be costly. Models for clusters of locations or transferable models might reduce this effort, but they are not evaluated by Rho et al. The authors also note that effectiveness heavily depends on the willingness of EV owners to report arrival and departure times, as well as state of charge, to participate in this scheduling.

Conclusion Rho et al. reinforce the pattern seen in the other two sources while also extending the discussions to EV charging management systems and distributed systems. They show that such systems can even turn a profit, demonstrated with a model tailored to one site and trained on its historical data. While the accuracy is high in this setting, a commercial system like the KEBA eMobility Portal needs to work out-of-the-box for a wide variety of small sites, for which such site-specific data is usually not available. How well the approach transfers to this setting remains open.

Chapter 9

Reflection and Conclusion

9.1 Reflection

In hindsight, more time should have been spent looking for additional real-world data locations to broaden the evaluation. While it would have brought more data, the locations would still have been located in a relatively small area.

Because data collection took so long and proved less useful than hoped, implementation of all the OCPP logic needed to be API-agnostic, and some assumptions needed to be made early on. In hindsight, this proved positive: because no good API provider was found, there is less work to do to adapt it to a future real provider.

9.2 Conclusion

The goal of this thesis was not only to implement a solar-forecast-powered charging strategy for the KEBA wallbox, but also to determine commercial usability.

The evaluation started with 12 API providers but quickly narrowed down to only two that could be evaluated, based on pricing, availability, and data granularity. Comparing both forecasts to real-world production data showed that, for forecasts made one hour ahead, both APIs deviated from actual production by 63% to 102% on average (MAPE), with Meteoblue being consistently more accurate than Forecast.Solar. An analysis by weather category, based on daily cloud cover, showed that the relative error of both APIs increases from 27% to 67% on clear days to 115% to 158% on overcast days, while the absolute error does not increase and remains between roughly 900 and 2800 W in all categories. Errors above 200% occurred on individual days, but not for a weather category as a whole. Since these errors are not negligible in relation to typical household use, generic forecasting APIs are, based on the forecast accuracy measured here, not precise enough to guide charging decisions for small charging stations. As discussed in Chapter 7, this conclusion uses forecast error as a substitute for the quality of the resulting charging schedules, whose actual effect on grid consumption and PV self-consumption was not quantitatively evaluated.

While the current research reviewed in Related Work (Chapter 8) seemed optimistic, this may be because most research was done for a narrow scope and

highly site-specific, tailored models. Such site-specific models are hard to apply to systems like the KEBA eMobility Portal, since training a model per site would require historical data for every location and be costly; whether generalized or regional models can close this gap was not investigated in the reviewed work or in this thesis.

Despite this, the system developed as part of this thesis provides an API-agnostic proof-of-concept implementation, as shown in Chapter 5 and Chapter 6. This means that as soon as a viable API hits the market or another calculation method for forecasts is introduced, either an in-house model or calculation via other parameters, only the data fetching service needs to be changed, while the rest of the backend handles different granularity as well as different timeframes out of the box.

In this sense, this thesis concludes that, while currently not feasible on a larger scale, solar forecasting can be achieved on a smaller scale, as seen in related work, and all groundwork is done to create systems like these on a larger scale.

Bibliography

- [1] AccuWeather, Inc. 2026. AccuWeather APIs. Retrieved 05/20/2026 from <https://developer.accuweather.com/>.
- [2] R. Ahmed, V. Sreeram, Y. Mishra, and M.D. Arif. 2020. A review and evaluation of the state-of-the-art in PV solar power forecasting: Techniques and optimization. *Renewable and Sustainable Energy Reviews*, 124, 109792. ISSN: 1364-0321. DOI: <https://doi.org/10.1016/j.rser.2020.109792>. <https://www.sciencedirect.com/science/article/pii/S1364032120300885>.
- [3] enercast GmbH. 2026. enercast. Retrieved 05/20/2026 from <https://www.enercast.de/>.
- [4] European Commission, Joint Research Centre. 2026. PVGIS Photovoltaic Geographical Information System. Retrieved 05/20/2026 from https://joint-research-centre.ec.europa.eu/pvgis-online-tool/pvgis-tools_en.
- [5] Forecast.Solar. 2026. Forecast.Solar. Retrieved 05/20/2026 from <https://forecast.solar/>.
- [6] Google. 2026. Angular. Retrieved 05/20/2026 from <https://angular.dev/>.
- [7] JetBrains. 2026. Kotlin Programming Language. Retrieved 05/20/2026 from <https://kotlinlang.org/>.
- [8] KEBA Group AG. 2026. KEBA. Retrieved 05/20/2026 from <https://www.keba.com/>.
- [9] Leaflet. 2026. Leaflet – a JavaScript library for interactive maps. Retrieved 05/20/2026 from <https://leafletjs.com/>.
- [10] meteoblue AG. 2026. meteoblue. Retrieved 05/20/2026 from <https://www.meteoblue.com/>.
- [11] MeteoForEnergy. 2026. MeteoForEnergy. Retrieved 05/20/2026 from <https://www.meteoforenergy.com/en/>.
- [12] Meteomatics AG. 2026. Meteomatics Weather API. Retrieved 05/20/2026 from <https://www.meteomatics.com/en/weather-api/>.
- [13] NASA Langley Research Center. 2026. NASA POWER (Prediction Of Worldwide Energy Resources). Retrieved 05/20/2026 from <https://power.larc.nasa.gov/>.
- [14] Observable, Inc. 2026. Observable Plot. Retrieved 05/20/2026 from <https://observablehq.com/plot/>.
- [15] Open Charge Alliance. 2017. Open Charge Point Protocol 1.6, Edition 2. Technical report. Final release, merges all Edition 1 errata. Licensed under CC BY-ND 4.0. Open Charge Alliance, (September 2017). Retrieved 05/20/2026 from <https://www.openchargealliance.org/protocols/ocpp-16/>.

- [16] Open-Meteo. 2026. Historical Weather API. Retrieved 09/05/2026 from <https://open-meteo.com/en/docs/historical-weather-api>.
- [17] OpenStreetMap contributors. 2026. OpenStreetMap. Retrieved 05/20/2026 from <https://www.openstreetmap.org/>.
- [18] OpenWeather Ltd. 2026. Solar Panels and Energy Prediction API. Retrieved 05/20/2026 from <https://openweathermap.org/api/solar-panels-and-energy-prediction>.
- [19] Prism. 2026. Prism – Lightweight, robust, elegant syntax highlighting. Retrieved 05/20/2026 from <https://prismjs.com/>.
- [20] Red Hat. 2026. Quarkus – Supersonic Subatomic Java. Retrieved 05/20/2026 from <https://quarkus.io/>.
- [21] Seoeun Rho, Myeongseok Chae, and Dongjun Won. 2024. Forecast-based Optimal Operation of EV Charging Station with PV Considering Charging Demand and Distributed System. In *2024 IEEE Power Energy Society Innovative Smart Grid Technologies Conference (ISGT)*, pp. 1–5. DOI: 10.1109/ISGT59692.2024.10454193.
- [22] Sheik Mohammed S., Femin Titus, Sudhakar Babu Thanikanti, Sulaiman S. M., Sanchari Deb, and Nallapaneni Manoj Kumar. 2022. Charge Scheduling Optimization of Plug-In Electric Vehicle in a PV Powered Grid-Connected Charging Station Based on Day-Ahead Solar Energy Forecasting in Australia. *Sustainability*, 14, 6. ISSN: 2071-1050. DOI: 10.3390/su14063498. <https://www.mdpi.com/2071-1050/14/6/3498>.
- [23] SolarEdge Technologies. 2026. SolarEdge API | Energy Data Platform for Developers. Retrieved 05/20/2026 from <https://developer.solaredge.com/>.
- [24] Solcast. 2026. Solcast. Retrieved 05/20/2026 from <https://www.solcast.com/>.
- [25] Stormglass. 2026. Stormglass. Retrieved 05/20/2026 from <https://stormglass.io/>.
- [26] World Weather Online. 2026. World Weather Online. Retrieved 05/20/2026 from <https://www.worldweatheronline.com/>.