HAW HAMBURG

BACHELOR THESIS
Jonas Kindel

Entwicklung eines Simulationsmodells mit Python für das Ladeverhalten von Elektromobilität

FAKULTÄT TECHNIK UND INFORMATIK
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Jonas Kindel

Entwicklung eines Simulationsmodells mit Python für das Ladeverhalten von Elektromobilität

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Regenerative Energiesysteme und Energiemanagement*
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Kolja Eger
Zweitgutachter: Prof. Dr. Wolfgang Renz

Eingereicht am: 25.01.2024

Jonas Kindel

Thema der Arbeit

Entwicklung eines Simulationsmodells mit Python für das Ladeverhalten von Elektromobilität

Stichworte

Simulation, Python, Elektromobilität, Ladeverhalten, Ladeinfrastruktur, Ladesäulen, Supermarkt, Kühlräume

Kurzzusammenfassung

Die Betreiber von Supermärkten bieten ihren Kunden heutzutage immer öfter die Möglichkeit, ihre Elektroautos während des Einkaufens mittels Ladesäulen auf dem eigenen Kundenparkplatz zu laden. Bei der Planung dieser Ladesäulen muss die maximale Leistungsaufnahme bestimmt werden. Je höher diese ist, desto mehr Grundgebühren zahlt der Supermarkt an den Netzbetreiber, unabhängig davon, wie oft diese maximale Leistung tatsächlich angefordert wird.
Durch das Betreiben von Schnellladepunkten auf dem Parkplatz entstehen hohe Lastspitzen im Gesamtlastverlauf des Supermarktes. Um Kosten zu reduzieren, wird in dem Forschungsprojekt 'EcoCharge' nach Möglichkeiten gesucht, diese Lastspitzen abzusenken. Es wird mittels Deap-Learning-Verfahren eine Vorhersage für den Lastverlauf berechnet, um so die Kühlung des Supermarktes in Abhängigkeit der Last an den Ladestationen zu steuern. So kann beispielsweise vorgekühlt werden, um die Kühlung bei einer Lastspitze zeitweise zu reduzieren.
Damit das Programm funktionieren kann, müssen Daten für den Lastverlauf der Ladestationen vorhanden sein. Reale Daten von Schnellladesäulen liegen jedoch nicht vor. In dieser Arbeit wird eine Simulation erstellt, um die fehlenden Daten für das Projekt zu generieren. Dabei werden unterschiedliche Parameter berücksichtigt, welche von dem Benutzer eingestellt werden können. Anschließend werden die Auswirkungen dieser Parameter auf den Lastverlauf untersucht.

Jonas Kindel

Title of Thesis

Development of a simulation model with Python for the charging behavior of electric mobility

Keywords

simulation, Python, electromobility, charging infrastructure, charging columns, supermarket, cold rooms

Abstract

Nowadays, supermarket operators increasingly provide their customers with the opportunity to charge their electric cars while shopping, using charging stations in their own customer parking lots. When planning these charging points, the maximum power consumption must be determined. The higher it is, the more basic fees has the supermarket to pay to the grid operator, regardless of how often this maximum power is actually requested.

Operating fast-charging points in the parking lot results in high load peaks in the supermarket's overall load profile. In order to reduce costs, the research project 'EcoCharge' is exploring ways to reduce these load peaks. Deap-learning methods are used to calculate a prediction for the load profile, allowing control of the supermarkets cooling system based on the load at the charging stations. For example, pre-cooling can be used to temporarily reduce the cooling during a peak load.

For the program to function, data for the load profile of the charging stations must be available. However, real data from fast-charging stations are not available. In this work, a simulation is developed to generate the missing data for the project. Various parameters are taken into account, which can be adjusted by the user. The effects of these parameters on the load curve are then examined.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Glossar

DataFrame Zweidimensionale Datenstruktur, die Zeilen und Spalten organisiert. In dieser Arbeit wird die Python-Bibliothek 'pandas' genutzt, um mit DataFrames zu arbeiten [23].

Deep-Learning Deep-Learning ist ein Teilbereich des maschinellen Lernens, der sich auf die Entwicklung und das Training künstlicher neuronaler Netzwerke konzentriert, um Aufgaben zu lösen, die in der Regel menschliche Intelligenz erfordern [20].

EcoCharge Innovationsprojekt zur Senkung von Energiekosten durch Nutzung der Ladevorgänge von Elektrofahrzeugen zur Lastverschiebung auf der Basis von Deep-Learning-Verfahren.

Enividatec GmbH Firma zur Erbringung von Dienstleistungen im Bereich der Energietechnik, insbesondere im Zusammenhang mit Techniken, die den Energieverbrauch regeln oder messen sowie das Übertragen dieser Daten ermöglichen. Ferner das Entwickeln und Vermarkten dieser Techniken [10].

JSON (JavaScript Object Notation) ist ein schlankes Datenaustauschformat, das für Menschen einfach zu lesen und zu schreiben und für Maschinen einfach zu parsen (Analysieren von Datenstrukturen) und zu generieren ist [1].

Jupyter-Notebook Software zum erstellen und teilen interaktiver digitaler Notizbücher. Es können sowohl Notizen verfasst, als auch Code geschrieben und ausgeführt werden.

Python Objektorientierte, interpretierte, höhere Programmiersprache. Zielsetzung von Python ist es, einen gut lesbaren und knappen Programmierstil zu fördern [22].

Repository Digitaler Speicher für Quellcode. Wird genutzt, um Änderungen am Quellcode vorzunehmen, zu verwalten und zwischen mehreren Entwicklern auszutauschen. So können mehrere Entwickler an einem Projekt arbeiten.

Simulations-Modell Ziel eines Simulationsmodells ist die Darstellung wesentlicher Charakteristiken eines realen Systems oder realen Prozesses. In dem Modell können beispielsweise verschiedene Szenarien mit unterschiedlichen Bedingungen vorausberechnet werden.

Abkürzungen

AC alternating current.

Akku Akkumulator.

Akkus Akkumulatoren.

Benutzer Benutzer der Software.

BEV battery electric vehicle.

BMS Batterie-Management-System.

CCS combined charging system.

Chademo charge de move.

DC/HPC direct current / high-power-charging.

EV electric vehicle.

Fahrer Fahrer des EV.

HAW Hochschule für Angewandte Wissenschaften Hamburg.

ICCB in-cable control-box.

IDE Integrierte Entwicklungsumgebung.

kW Kilowatt.

PHEV plug-in hybrid electric vehicle.

REEV range extended electric vehicle.

SoC state of charge.

SoH state of health.

1 Einleitung

Die voranschreitende Klimaveränderung und die damit einhergehende Notwendigkeit, den CO2-Ausstoß zu reduzieren, haben die Elektromobilität als zentralen Bestandteil einer nachhaltigen Verkehrswende in den Fokus gerückt. In diesem Kontext spielen die Lademöglichkeiten von electric vehicle (EV) eine wichtige Rolle. Fahrer des EV (Fahrer) sollen die Möglichkeit haben, das Laden ihrer Fahrzeuge sinnvoll in ihren Alltag zu integrieren, ohne zusätzliche Zeit für diesen Vorgang einplanen zu müssen. Aus diesem Grund ist die Nachfrage nach Schnellladepunkten in den letzten Jahren deutlich angestiegen. [6] Ein sinnvoller Ansatz ist das Schnellladen von EV während des Einkaufens. Immer mehr Supermarktketten wollen ihren Kunden die Möglichkeit bieten, ihre EV auf dem Supermarktparkplatz zu laden. Der Gesamtenergieverbrauch des Supermarktes nimmt jedoch durch die hinzugekommenen Schnellladepunkte rapide zu. Das Lastprofil der Schnellladepunkte ist über einen gesamten Tag betrachtet nichtlinear und es entstehen hohe Lastspitzen. Der Supermarkt-Betreiber zahlt bei dem zuständigen Verteilnetzbetreiber hohe Gebühren für die maximale Anschlussleistung. Die Kosten entstehen unabhängig davon, wie oft diese maximale Leistung tatsächlich abgerufen wird. Es liegt daher im Interesse des Supermarkt-Betreibers, die Lastspitzen in seinem Gesamtlast-Profil abzuflachen. Zur Optimierung des Energieverbrauchs von Supermärkten wurde in Kooperation mit der Energie-Beraterfirma Enividatec GmbH an der Hochschule für Angewandte Wissenschaften Hamburg (HAW) das Innovationsprojekt 'EcoCharge' initiiert. Dieses Projekt verfolgt das Ziel, die Energiekosten durch Lastspitzen-Verschiebung zu senken. Dabei kommen verschiedene Methoden zum Einsatz. Unter anderem werden mittels verschiedener Deep-Learning Verfahren neurale Netze trainiert, um den Energieverbrauch des Supermarktes vorherzusagen. Mithilfe dieser Vorhersage kann die Steuerung der Kühlung an die Lastkurven der Ladepunkte auf dem Parkplatz angepasst werden. Zum Beispiel wäre es möglich, eine Vorkühlung durchzuführen, um die Kühllast vorübergehend zu reduzieren, wenn eine Lastspitze auftritt. Ein spezieller Fokus des Projektes 'EcoCharge' liegt auf der Analyse des Ladeverhaltens von EV auf Supermarktparkplätzen. Die Erfassung von minutengenauen Ladedaten von Kundenfahrzeugen bildet die Grundlage für

das Trainieren eines neuronalen Netzwerks. Aufgrund der begrenzten Verfügbarkeit von realen Ladevorgängen durch Schnellladesäulen wird in dieser Forschungsarbeit vorerst auf eine Simulation des Ladeverhaltens zurückgegriffen.

Die zentrale Zielsetzung dieser Arbeit besteht darin, ein adäquates Simulationsmodell für das Ladeverhalten von EV zu erstellen und dieses mithilfe der Programmiersprache Python umzusetzen. Der Input für die Simulation besteht primär aus aufgezeichneten Ladekurven von EV. Zudem kommen unterschiedliche stochastische Methoden zum Einsatz, um zum Beispiel das Verhalten der Kunden in der Simulation abzubilden. Wesentliche Parameter sind manuell einstellbar, um eine flexible Anpassung der Simulation an unterschiedliche Szenarien zu ermöglichen. Der Output der Simulation bildet einen Lastgang der Ladesäulen ab, der minutengenau über einen gesamten Tag generiert wird. In der Arbeit werden verschiedene Szenarien durchgespielt, um den Einfluss einzelner Parameter auf den Lastgang zu untersuchen. Die Auswertung erfolgt durch die Erstellung entsprechender Grafiken und die Betrachtung der auftretenden Lastspitzen. Diese differenzierte Analyse soll dazu beitragen, Erkenntnisse über das Ladeverhalten von EV zu gewinnen und Optimierungspotenziale im Kontext von Lastverschiebungen zu identifizieren.

Um diese Zielsetzung erreichen zu können, gliedert sich die vorliegende Arbeit in mehrere Abschnitte. Dies gewährleistet einen systematischen Einblick in den Forschungsprozess. In Kapitel 2 werden die Grundlagen erläutert und es wird auf vergleichbare Forschungsarbeiten eingegangen. In Kapitel 3 werden die spezifischen Anforderungen an das Simulationsmodell im Detail beleuchtet. Dieser Abschnitt legt den Grundstein für die präzise Abbildung des Ladeverhaltens von EV auf einem Parkplatz mit mehreren Ladepunkten, um eine realitätsnahe Simulation zu ermöglichen. Im Abschnitt 3.2 wird auf die essenziellen Eingangsparameter des Modells eingegangen. Hierzu zählen der Ankunftsprozess der Fahrzeuge, die durchschnittliche Parkdauer, der Startwert des state of charge (SoC), die Anzahl der verfügbaren Ladesäulen und die maximale Leistung pro Ladesäule. Eine detaillierte Betrachtung der Ladekurven der gewählten EV erfolgt im Abschnitt 3.3. In Unterpunkt 3.3.1 wird auch die Auswahl der EV näher erläutert.

Das Kapitel 3.4 thematisiert das Klassen-Modell im Kontext des Software Designs. Hier liegt der Fokus auf der Strukturierung des Simulationsmodells in Klassen, um eine übersichtliche Implementierung sicherzustellen.

Das Kapitel 4 befasst sich mit der konkreten Implementierung des Simulationsmodells. Der Abschnitt 4.2 gibt einen Überblick über den initialen Entwurf des Modells, auch als

'Sketch' bezeichnet, und skizziert die wesentlichen Designentscheidungen und Strukturen, die als Grundlage für die Implementierung dienen.

Die Simulationsergebnisse und deren Auswertung werden in Kapitel 5 behandelt. Hier erfolgt eine umfassende Analyse der Umsetzung der gestellten Anforderungen sowie eine eingehende Validierung des Simulationsmodells. Der Abschnitt 5.3 vertieft die Betrachtung der Simulationsergebnisse im Hinblick auf die gestellten Anforderungen.

Abschließend fasst das Kapitel 6 die gewonnenen Erkenntnisse zusammen und reflektiert diese im Fazit. Ein Ausblick gibt Hinweise auf potenzielle Weiterentwicklungen und Anwendungsgebiete des entwickelten Simulationsmodells.

2 Grundlagen

In diesem Kapitel werden Grundlagen bezüglich der Elektromobilität vermittelt. Es wird insbesondere auf die unterschiedlichen Ladetechniken eingegangen, da diese eine besondere Relevanz für die vorliegende Arbeit haben. Dabei werden wichtige Begriffe erklärt und es wird auf Faktoren eingegangen, welche die Ladegeschwindigkeit von EV beeinflussen. Dieses Grundwissen trägt dazu bei, die folgenden Abschnitte der Arbeit besser verstehen und einordnen zu können.

2.1 Definition Elektromobilität

Der Begriff 'Elektromobilität' wird von der aktuell amtierenden Bundesregierung wie folgt definiert:

> Elektromobilität im Sinne der Bundesregierung umfasst all jene Fahrzeuge, die von einem Elektromotor angetrieben werden und ihre Energie überwiegend aus dem Stromnetz beziehen, also extern aufladbar sind. Dazu gehören rein elektrisch betriebene Fahrzeuge (battery electric vehicle (BEV)), eine Kombination von E-Motor und kleinem Verbrennungsmotor (Range Extender, range extended electric vehicle (REEV)) und am Stromnetz aufladbare Hybridfahrzeuge (plug-in hybrid electric vehicle (PHEV)) [5].

Somit umfasst die Definition für EV neben BEV auch REEV und PHEV. Der Anteil der BEV liegt mit 2,08 % aller in Deutschland zugelassenen Fahrzeuge in derselben Größenordnung wie PHEV mit 1,77 % im Jahr 2023 [19]. Der Bestand sowie Neuzulassungen von REEV hingegen ist in Deutschland verschwindend gering. [13] Diese speziellen Modelle werden daher in dieser Arbeit nicht weiter berücksichtigt.

PHEV können ledliglich mit niedrigen Leistungen von maximal 11 Kilowatt (kW) geladen werden. Da das Projekt 'EcoCharge' auf die Verwendung von Schnellladepunkten (2.2) abzielt, können sie für die Simulation ebenfalls vernachlässigt werden.

2.2 Ladetechnik von EV

Als Ladetechniken für EV haben sich verschiedene Systeme als Standard etabliert. In Europa wird dabei unterschieden zwischen drei verschiedenen Modi. Mode 2 stellt die einfachste Variante des Ladens dar. Mittels einer in-cable control-box (ICCB) oder einer Wallbox wird das EV mit maximal 22 kW über alternating current (AC) geladen [16]. Mode 3 beschreibt das dreiphasige Laden an öffentlichen Ladesäulen und bietet eine Ladeleistung von bis zu 43,5 kW. Bei Mode 2 und Mode 3 kommt ein Typ 2-Stecker zum Einsatz. Dieser ist durch die Europäische Kommission seit 2013 zur Norm erklärt worden:

> „Ein einheitlicher EU-Ladestecker ist für die Markteinführung dieses Kraftstoffs (Strom) entscheidend. Um die auf dem Markt herrschende Unsicherheit zu beenden, hat die Kommission heute die Verwendung des Steckers vom '**Typ 2**' zur gemeinsamen **Norm für ganz Europa** erklärt."[16]

Mode 4 sieht das Laden an direct current / high-power-charging (DC/HPC) Ladestationen vor. Diese bieten mit bis zu 350 kW eine viel höhere Ladeleistung. Daher wird bei diesen Ladestaionen auch von Schnellladestationen gesprochen [2]. Bei einer DC/HPC Ladung kommen verschiedene Steckverbindungen zum Einsatz. Während der charge de move (Chademo) Stecker lediglich eine Leistung von bis zu 100 kW zulässt, kann über das combined charging system (CCS) oder auch Combo-System mit bis zu 350 kW geladen werden. Der Trend bei den Autoherstellern geht zum CCS-Stecker [8].

2.2.1 Ladekurve von EV

Gewöhnlich werden bei BEV Lithium-Ionen-Akkus als Energiespeicher eingesetzt. Diese bieten verglichen mit anderen Akkumulatoren (Akkus) eine hohe Leistungs- und Energiedichte. Ein weiterer Vorteil ist die geringe Selbstentladung und der hohe Wirkungsgrad

dieser Zellen. In der Praxis bilden viele einzelne Lithium-Ionen-Zellen ein Modul. Diese Module sind wiederum zusammengefasst zu einem Fahrzeug-Akkumulator (Akku).

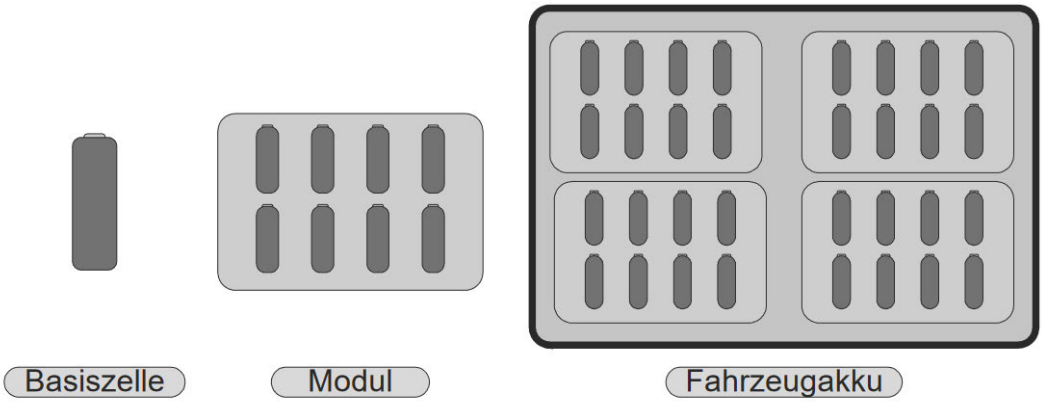

Abbildung 2.1: Modularer Aufbau eines Fahrzeugakkus [16]

Da die Lithium-Ionen-Zellen sehr pfleglich behandelt werden müssen, wird der Akku von einem Batterie-Management-System (BMS) überwacht und gesteuert. Laut [16] übernimmt das BMS dabei folgende Aufgaben:

- Ladekontrolle
- Zellschutz
- Lastmanagement
- Bestimmung des SoC
- Ausbalancieren der Zellen
- Kommunikation, Historie
- Thermomanagement

Eine Aufgabe des BMS ist es, die aktuelle Ladeleistung während eines Ladevorgangs zu steuern. Die daraus resultierende Ladekurve hängt von dem SoC, der Temperatur sowie der Alterung der Batterie ab. Den größten Einfluss hat dabei der SoC. Um die Akkus zu schonen [16], lässt das BMS bei fortgeschrittener Ladung immer geringere Ladeleistungen zu. Spätestens ab 80 % SoC fällt die Ladeleistung signifikant ab [3]. Wie in Abbildung

3.6 zu sehen, variieren die Ladekurven bei unterschiedlichen Modellen stark. Dennoch ist bei allen EV der Trend zu erkennen, dass mit fortgeschrittenem SoC die Ladeleistung stark abnimmt. Daher wird in dieser Arbeit besonders der SoC zur Bestimmung des Lastverlaufes berücksichtigt.

2.3 Vergleichbare Arbeiten

Die zunehmende Bedeutung der Elektromobilität spiegelt sich auch in der Forschung wider. Gerade die Thematik des Ladevorgangs ist dabei von großem Interesse. So wurde beispielsweise an der Hochschule Coburg in Zusammenarbeit mit der Siemens AG bereits daran geforscht, diese Ladevorgänge mithilfe eines Software-Modells zu simulieren. Diese Arbeit bezieht sich jedoch auf innerstädtische Ladestationen mit maximal 11 kW Leistung. Mit einer Anzahl von 47 Ladestationen sind diese jedoch in größerer Zahl in der Arbeit betrachtet [14]. Bei dem Projekt 'EcoCharge' hingegen werden Schnellladesäulen verwendet. Diese sind gewöhnlich in geringerer Anzahl pro Standort vorhanden, stellen aber durch die hohen Lastspitzen eine besondere Herausforderung dar. Diese Thematik ist Gegenstand dieser Arbeit.

3 Simulations-Modell

Ein Simulations-Modell wird genutzt, um die Charakteristiken eines realen Systems oder eines realen Prozesses durch Berechnung zu bestimmen. Mithilfe eines solchen Modells können beispielsweise unterschiedliche Bedingungen simuliert und deren Auswirkungen untersucht werden, ohne die Situationen in der realen Welt nachstellen zu müssen. Dies ist vor allem dann hilfreich, wenn der reale Prozess nicht nachgestellt werden kann oder keine entsprechenden Aufzeichnungen vorliegen. [7]

Ziel des Simulationsmodells in dieser Arbeit ist es, den Lastverlauf der Ladesäulen eines Supermarkt-Parkplatzes möglichst realitisch abzubilden. Mit in die Simulation einfließen sollen authentische, aufgezeichnete Daten, soweit dies realisierbar ist. Wenn keine realen Daten zu Verarbeitung vorliegen, werden verschiedene Parameter für die Simulation genutzt. Hierbei wird mit unterschiedlichen Verfahren gearbeitet, um die erforderlichen Daten zu generieren.

3.1 Anforderungen an das Simulations-Modell

Zu Beginn der Arbeit wurde zusammengefasst, welche Anforderungen an das Modell gestellt werden. Diese Anforderungen beinhalten die Berücksichtigung von Eingangsparametern, die Entwicklung des Simulations-Modells selbst sowie die Auswertung der simulierten Daten. In der nachfolgenden Grafik sind die Anforderungen an das Modell noch einmal dargestellt.

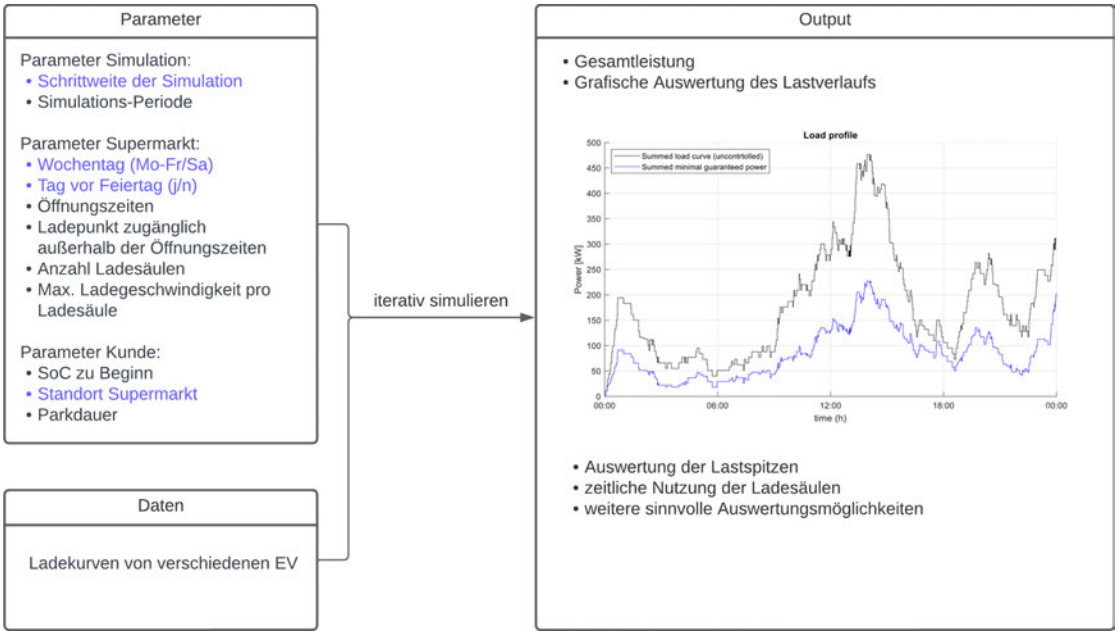

Abbildung 3.1: Modell-Anforderungen. Beipielgrafik für Lastverlauf aus [14]

Unterschiedliche Parameter werden in der vorliegenden Arbeit berücksichtigt. Sie sollen für den Benutzer der Software (Benutzer) manuell, aber zentral einstellbar sein. Die in schwarz dargestellten Parameter werden fest implementiert, andere, optionale Parameter müssen noch auf ihre Zweckmäßigkeit geprüft werden. Diese optionalen Einstellungen sind in der Grafik farblich markiert dargestellt.

Die simulierten Daten dienen in dem Projekt dazu, ein neurales Netz mit den Lastdaten zu trainieren. Ein wichtiger Punkt ist daher die effiziente Nutzung der simulierten Daten in der Zukunft. Bedingung hierfür ist, dass die Daten in einem sinnvollen Datenformat vorliegen müssen. Ziel der Simulation ist es nicht, lediglich einmal Daten zu generieren, welche dann im weiteren Projektverlauf verwendet werden. Stattdessen soll die gesamte Simulation implementiert werden. So lassen sich auch später noch Parameter vom Benutzer anpassen und gegebenfalls Lastverläufe für verschiedene Szenarien direkt beim Trainieren des neuralen Netzwerks generieren. Eine detaillierte Dokumentation über das Repository und die Simulation ist daher von großer Wichtigkeit.

Ein bedeutender Fokus liegt auf der Entwicklung und Auswertung von Szenarien. Dafür müssen passende Methoden zur Auswertung des Simulationsergebnisses herausgearbeitet und implementiert werden. Neben dem Lastverlauf liegt ein Hauptaugenmerk auf der

Auswertung der Lastspitzen. Diese sollen sowohl grafisch als auch numerisch analysierbar sein. Wichtig ist dabei die Vergleichbarkeit verschiedener Lastverläufe miteinander.

Es sollen verschiedene Szenarien simuliert und miteinander verglichen werden. Um den Überblick über diese Szenarien und deren Ergebnisse zu behalten, soll nicht nur das isolierte Simulationsergebnis gespeichert werden. Stattdessen wird in Jupyter-Notebook-Files beschrieben, wie die Parameter konfiguriert wurden und das entsprechende Ergebnis anschließend ausgewertet. Die Arbeitsschritte sind übersichtlich dokumentiert und gegebenenfalls reproduzierbar. So lässt sich sicherstellen, dass die Simulation auch im weiteren Projektverlauf genutzt werden kann.

3.2 Eingangs-Parameter

Um die praktikable Bedienbarkeit der Simulation zu gewährleisten, sollen sämtliche einstellbaren Parameter zentral für den Benutzer erreichbar sein. Hierzu wurde eine Settings-Datei angelegt, in welcher alle möglichen Parameter zu finden sind. Um in Bezug auf die weitere Verwendbarkeit der Simulations-Software möglichst gut aufgestellt zu sein, wird das Datenaustauschformat JSON genutzt. Dieses Format hat sich mittlerweile soweit für diesen Einsatzzweck etabliert, dass es von vielen modernen Programmiersprachen unterstützt wird [1]. Somit ist die weitere Verwendung der Simulation nicht auf die Nutzung von Python beschränkt.

3.2.1 Modellierung Parkdauer

Die Parkdauer der Kunden stellt einen essenziellen Parameter in der Simulation dar, da sie maßgeblichen Einfluss auf das Ladeverhalten der EV ausübt. Wenn ein EV den Parkplatz vor Abschluss des vollständigen Ladevorgangs verlässt, resultiert dies, wie in Abbildung 3.6 ersichtlich, in einem größeren Anteil des Ladeverlaufs im höheren Lastbereich.

In dem ersten Implementierungsschritt wurde hier nach dem Zufallsprinzip gearbeitet. In den Einstellungen kann der Benutzer zwei Werte in Minuten vorgeben, innerhalb derer sich die Parkzeit der EV befinden soll. Für jedes neu ankommende Fahrzeug wird

dann gleichverteilt zufällig ein Minutenwert innerhalb dieser Zeitspanne gewählt. Diese Methode wird zunächst genutzt, um die Simulation und deren Implementierung zu verifizieren.

Im weiteren Verlauf der Arbeit wird mit real gemessenen Daten bezüglich der Ladezeit gearbeitet. In dem Whitepaper 'Elektromobilität im Handel 2023' [18] werden auf Seite 21 die durchschnittliche Ladezeit ausgewählter Formate dargestellt. Da sich diese Arbeit auf den Parkplatz eines Suptermarktes bezieht, sind speziell die durchschnittlichen Ladezeiten eines Lebensmittelhandels interessant. Folgende Werte können dem Whitepaper entnommen werden:

Tabelle 3.1: Durchschnittliche Ladezeit im Lebensmittelhandel [18]

	Max. 30 Minuten	0,5-1 Stunde	1-2 Stunden
Anteil in %	14	64	22

Als Zwischenschritt wurde in der Simulation eine Funktion geschaffen, welche mit einer gewichteten Wahrscheinlichkeit zufällig aus einer dieser drei Optionen auswählt. Dabei wird jeweils von dem Mittelpunkt in dem Zeitintervall ausgegangen. Es gibt somit die drei Optionen 15 Minuten, 45 Minuten und 90 Minuten, welche einem EV zugeordnet werden. Diese Abstufung ist jedoch noch grob und spiegelt das tatsächliche Verhalten der Fahrer nur verzerrt wieder. Daher wird eine Funktion geschaffen, welche bei jedem Aufruf einen generierten Minutenwert zurückgibt. Der Ansatz sowie die Implementierung für die Nutzung der unterschiedlichen Parkdauern aus 3.1 mittels einer Weibull-Verteilung wurde von Prof. Eger in dem Jupyter-Notebook 'Distribution4Charging_Duration' zur Verfügung gestellt.

In dem ersten Schritt werden Samples unter Berücksichtigung von 3.1 erzeugt. Auch hierbei wird die Annahme getroffen, dass der Mittelpunkt eines Zeitintervalls als repräsentativ für das jeweilige Intervall angenommen werden kann.

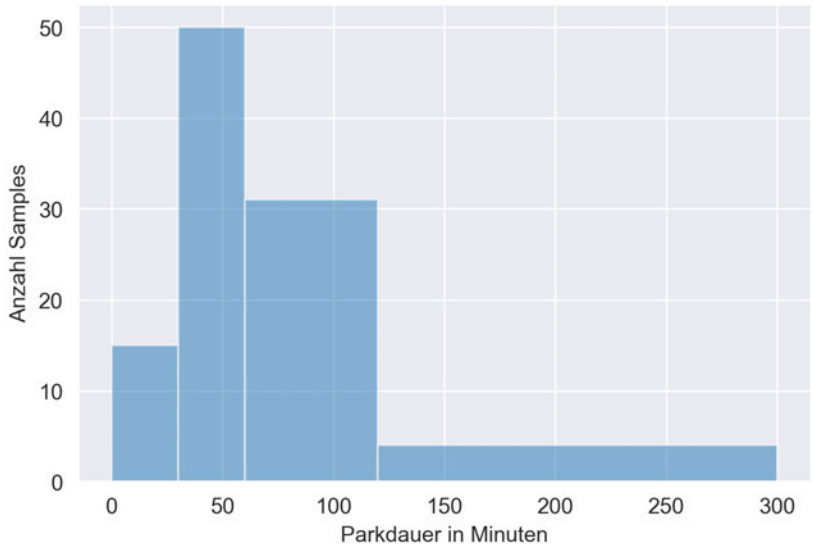

Abbildung 3.2: Samples für Weibull-Verteilung

Mithilfe dieser Samples können nun die benötigten Parameter für verschiedene Verteilungen bestimmt werden. Es sollen folgende drei Verteilungen untersucht werden:

- Normalverteilung
- Logarithmische Normalverteilung
- Weibull-Verteilung

Die Logarithmische Normalverteilung wird unter anderem in [14] genutzt, um die Parkzeit von Fahrzeugen in einem Parkhaus bei einem Shopping-Center zu bestimmen. Jedoch wurden bei dem oben beschriebenen Vorgehen, unter Zuhilfenahme der Samples, immer wieder Ausreißer in der Verteilung festgestellt. Daher wurde die Parkdauer mit der Weibull-Verteilung umgesetzt. Mittels folgender Parameter kann so für jedes Fahrzeug die Parkdauer bestimmt werden:

- $\alpha = 39.858$
- $c = 0.651$
- loc = -20.724
- $scale = 8.472$

In der folgenden Grafik werden die untersuchten Verteilungen miteinander verglichen. Es lässt sich erkennen, dass die Weibull-Verteilung mit den zuvor bestimmten Parametern die Samples der Parkdauer präziser abbildet als lediglich eine Normalverteilung.

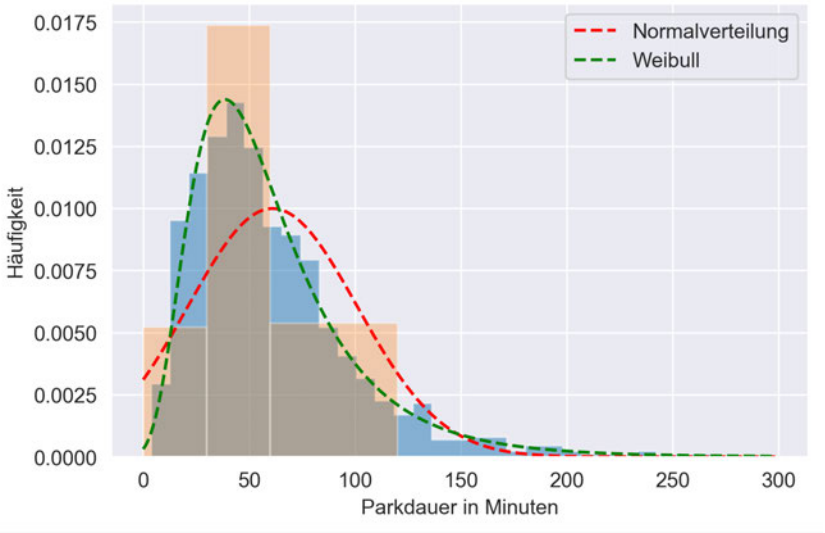

Abbildung 3.3: Unterschiedliche Verteilungen über den Samples

Als Standard-Parameter wird in der Simulation fortan die Weibull-Verteilung für die Parkdauer der neu ankommenden EV genutzt.

3.2.2 Modellierung Öffnungszeiten

Aus [18] geht hervor, dass der Zugang zu den Ladepunkten außerhalb der Öffnungszeiten in 50 % aller Fälle möglich ist. Es ist jedoch für die Simulation nicht sinnvoll, diesen Parameter lediglich zu mitteln. Vielmehr gibt es in dem JSON die Möglichkeit, die Öffnungszeiten des Supermarktes festzulegen. Soll die Lademöglichkeit rund um die Uhr bestehen, kann der Parameter 'accessible_all_day' auf 'true' gesetzt werden. Diese Einstellungen müssen für jeden Standort individuell festgelegt werden.

3.2.3 Modellierung initialer SoC

Zu Beginn eines Ladevorgangs muss bestimmt werden, welchen SoC das ankommende EV hat. Der initiale SoC bestimmt nicht nur die geladene Energiemenge, sondern hat

auch wesentlichen Einfluss auf den Lastverlauf eines Ladevorgangs. Wie in Abbildung 3.6 zu sehen ist, weisen die Ladekurven bei niedrigeren SoC-Werten einen signifikant höheren Lastgang auf als bei SoC-Werten > 50 %.

Für den ersten Implementierungsschritt und zur Validierung des Modells wird ein SoC von 0 % für neu ankommende EV angenommen. Dies ist keinesfalls realistisch und dient lediglich als Referenz für den ungünstigsten Fall (5.3) und zur Validierung des Modells. (5.2)

Im nächsten Schritt wird der initiale SoC für jedes neu ankommende EV mittels Zufallsverfahren bestimmt. Dabei werden verschiedene Zufallsverfahren gewählt. Bei dem ersten Verfahren handelt es sich um eine zufällige Gleichverteilung innerhalb eines bestimmten Bereichs in %, den der Benutzer einstellen kann.

Um das Simulationsergebnis weiter zu verbessern, wird sich an dem Vorgehen in [14] für den intialen SoC orientiert. Dort wird zur Bestimmung des SoC zu Beginn eines Ladevorgangs folgende Beta-Verteilung genutzt:

$$f(x|a,b) = \frac{1}{B(a,b)} x^{a-1}(1-x)^{b-1}$$

Die Parameter für die Beta-Verteilung sind abhängig von der Fahrergruppe der Ladepunkte und folgender Tabelle zu entnehmen:

Tabelle 3.2: Parameter für die Beta-Verteilung nach [14]

Zugang zu Lade-Infrastruktur	a	b
Privat	6	4
Privat und auf Arbeit	8	4
auf Arbeit	5	4
weder Privat noch auf Arbeit	4	4

Da keine genaue Zielgruppe bekannt ist, muss sich für die Option entschieden werden, welche am naheliegendsten erscheint. Die Ladepunkte in 'EcoCharge' werden auf einem Supermarkt-Parkplatz installiert. Es wird die Annahme getroffen, dass dort vorzugsweise

Fahrer laden, welche sonst keinen privaten Zugang zu Ladeinfrastruktur für ihr EV besitzen. Daher wird die Beta-Verteilung mit $a = 4$ und $b = 4$ parametrisiert. Die resultierende Verteilung ist folgendem Histogramm entnehmbar:

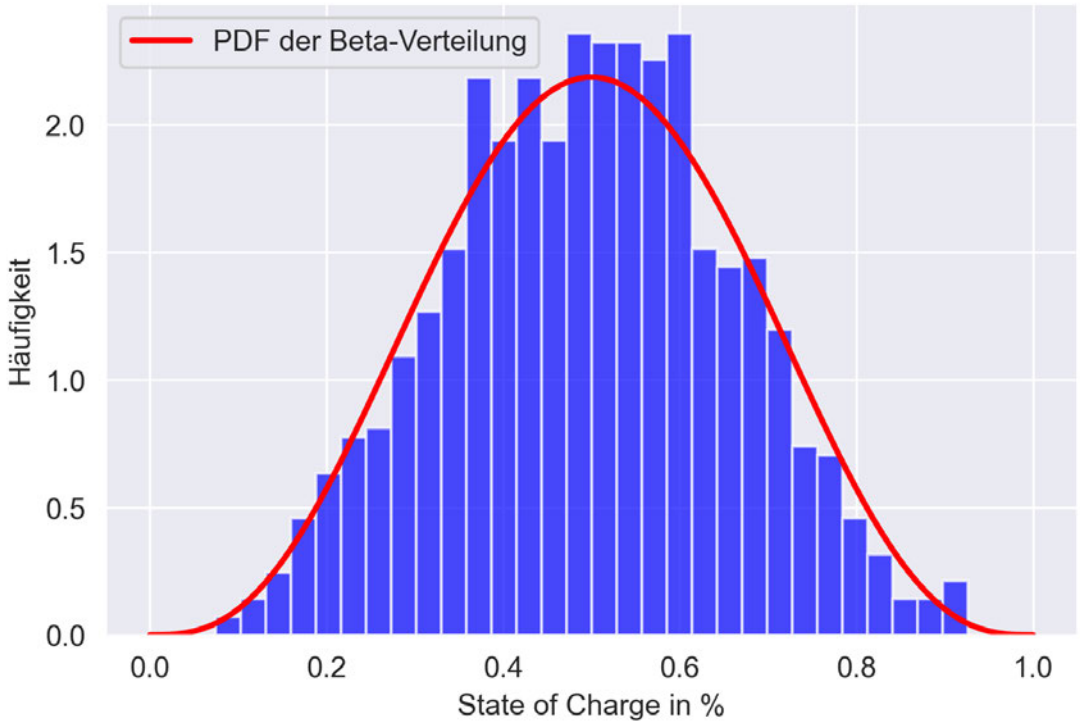

Abbildung 3.4: Beta-Verteilung für den SoC zu Beginn, a=4, b=4

Es ist erkennbar, dass die Verteilung einer einfachen Normalverteilung entspricht. Daher wird sich für die Modellierung des SoC zu Beginn im letzten Implementierungsschritt für eine Normalverteilung entschieden. Der Vorteil hierbei gegenüber der Beta-Verteilung ist, dass der Benutzer in den Einstellungen Grenzen festlegen kann, innerhalb welcher Prozentwerte sich der SoC zu Beginn eines Ladevorgangs befinden soll.

3.2.4 Modellierung Ankunftsprozess

In diesem Abschnitt wird der Ankunftsprozess der EV behandelt. Auch hier werden unterschiedliche Methoden zur Modellierung verwendet und miteinander verglichen.

Um den Lastverlauf der Ladepunkte simulieren zu können muss ein Kriterium gefunden werden welches bestimmt, zu welchem Zeitpunkt ein neues EV den Ladevorgang beginnt. In dem ersten Implementierungsschritt der Simulation wird hierzu ein einfacher Zufallsgenerator mit Gleichverteilung gewählt. Über einen Parameter in den Einstellungen kann der Benutzer festlegen, wie viele EV pro Stunde im Durchschnitt an den Ladepunkten erscheinen. Dieser Parameter kann im weiteren Projektverlauf an die jeweiligen Gegebenheiten angepasst werden.

Um die Modellierung der Ankünfte auf dem Parkplatz weiter zu verfeinern, gibt es in den Einstellungen eine weitere Möglichkeit, den Ankunftsprozess zu simulieren. Hierbei wird auf den Poisson-Prozess zurückgegriffen. Dabei handelt es sich um einen stochastischen Prozess, der in der Wahrscheinlichkeitstheorie und Statistik verwendet wird. Der Poisson-Prozess beschreibt das Auftreten von Ereignissen in diskreten Zeitpunkten, wobei die Zeit zwischen aufeinanderfolgenden Ereignissen exponentiell verteilt ist. Genutzt wird der Poisson-Prozess unter anderem in der Warteschlangentheorie [25]. Mithilfe der Warteschlangentheorie kann die Ankunftsrate von Fahrern realistisch modelliert werden. Hierzu wird in den Einstellungen mit dem Lamda-Wert des Poisson-Prozesses festgelegt, wie viele Fahrer pro Stunde durchschnittlich an den Ladepunkten erscheinen.

3.2.5 Modellierung Anzahl der Ladestationen

Eine weitere Einstellungsmöglichkeit in der Simulation ist die Anzahl der Ladestationen auf dem Parkplatz. Je nach Standort variiert diese Zahl. Solange alle Ladestationen belegt sind, kann kein weiteres EV einen Ladevorgang beginnen. Dieser Parameter kann von dem Benutzer in der JSON festgelegt werden.

3.3 Ladekurven der EV

Das typische Ladeverhalten eines EV ist nichtlinear und von Leistungssprüngen geprägt. Die Ladeleistung zu einem bestimmten Zeitpunkt wird von dem EV vorgegeben und ist abhängig von dem SoC. Ist der Ladezustand noch unter 30%, wird das EV tendenziell mit seiner maximalen Leistung geladen. Zwischen 30-50% SoC wird je nach Hersteller und Modell die Ladegeschwindigkeit weiter reduziert. Je weiter der Ladevorgang fortgeschritten ist, desto weiter wird auch die Ladeleistung reduziert. Dieses Verfahren wird

genutzt, um die Lebensdauer des Akkus zu verlängern, sowie der Hitzeentwicklung in dem Akku während des Ladevorgangs entgegen zu wirken.

Um das Ladeverhalten der EV realitätsgetreu simulieren zu können, mussten zunächst Ladekurven von EV beschafft werden. Nach einiger Recherche wurde schnell ersichtlich, dass zahlreiche real gemessene Ladekurven von EV frei im Internet zugänglich sind. Leider lagen diese jedoch meist in Form von Grafen in Grafiken vor. Entsprechende Rohdaten, welche in die Simulation hätten eingelesen werden können, waren nur sehr vereinzelt verfügbar. Daher wurden mithilfe des Tools [4] aus den entsprechenden Ladekurven die benötigten Datenpunkte extrahiert. (siehe 3.3.2)

3.3.1 Auswahl der Fahrzeuge

Die Auswahl der Fahrzeuge für das Simulations-Modell orientiert sich zunächst an dem Vorgehen von [14]. Hier wurden die EV in die folgenden sechs Fahrzeugklassen eingeteilt: Minis, Kleinwagen, Mittelklasse, Luxusautos, Sportwagen, SUVs. Jede Klasse wird von dem meist verkauften Modell aus dieser Klasse aus dem Jahr 2021 repräsentiert. Bei genauer Recherche wurde jedoch klar, dass sich auch die Fahrzeuge innerhalb einer Fahrzeugklasse stark im Ladeverhalten unterscheiden können [11]. Daher wird in dieser Arbeit eine andere Methodik genutzt. Beim Kraftfahrtbundesamt ist einsehbar, wie viele EV von welchem Typ genau in einem Jahr zugelassen wurden. Es ist der Trend erkennbar, dass neuere EV schneller Laden als vergleichbare EV früherer Generationen. Daher spielt die Aktualität der Daten in der Arbeit eine wesentliche Rolle. Es erfolgte die Auswahl der meistzugelassenen Modelle aus dem Jahr 2022. [6] Zunächst wird in der Simulation aus den gewählten Modellen zufällig ausgewählt. Ein möglicher Ansatz zur Verbesserung der Simulationsgenauigkeit wäre es, die EV nicht gleichverteilt zu wählen, sondern mit einer gewichteten Wahrscheinlichkeit. Der entscheidene Vorteil hierbei wäre, dass die unterschiedlichen Zulassungszahlen der Modelle berücksichtigt werden können.

Als eine gute Datenquelle für die genutzten Modelle stellte sich der Ladesäulenbetreiber 'Fastned' [12] heraus. In dem FAQ Bereich der Website lassen sich nach Hersteller sortiert alle gängigen EV finden. Zu jedem Modell lässt sich in dem Bereich eine Grafik abrufen, auf welchem der Lastgang abhängig vom SoC abzulesen ist.

Um die genutzten Ladekurven von [12] zu verifizieren, wurden besagte Ladekurven für die ausgewählten Modelle ebenfalls über andere Quellen abgerufen. Als eine zweite zuverlässige Quelle sollte an dieser Stelle [11] genannt werden. Diese Website bietet eine

Datenbank mit allerhand nützlichen und aktuellen Informationen über EV. Unter anderem können hier die Ladekurven für spezifische Modelle abgerufen werden. Legt man die Ladekurven von [12] und [11] übereinander fällt auf, dass diese mal mehr, mal weniger miteinander korrelieren. Beispielhaft sind in folgender Abbildung die Ladekurven des Hyundai KONA sowie des VW ID3 aus den beiden Quellen dargestellt:

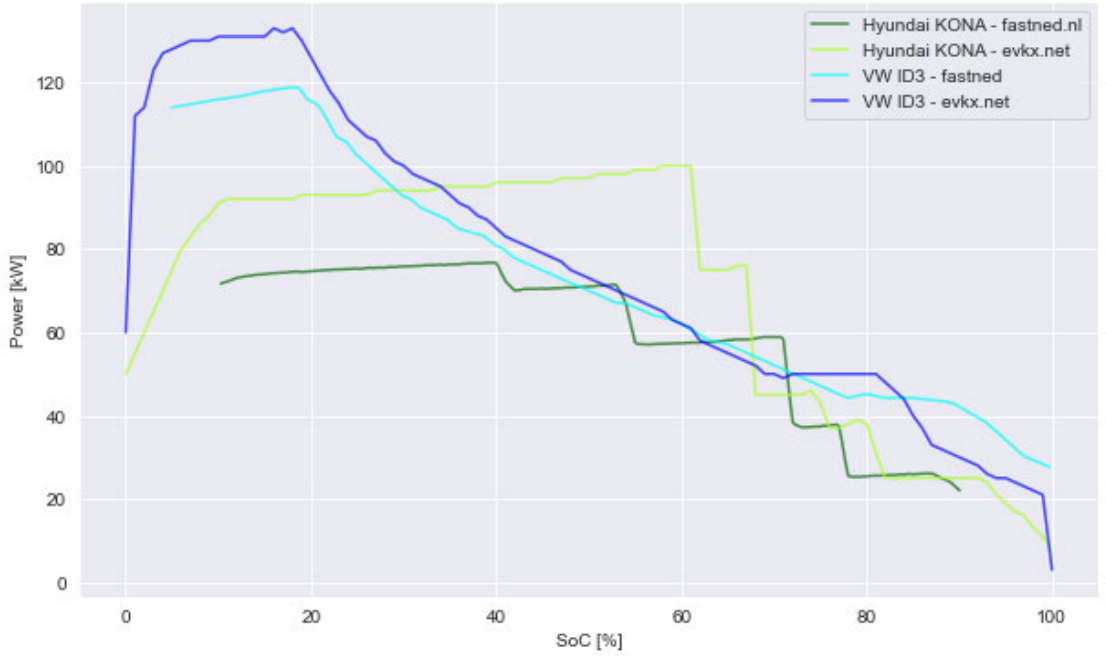

Abbildung 3.5: Vergleich Ladekurven von fastned und EVKX

Es wird ersichtlich, dass die gemessenen Ladekurven nicht exakt übereinstimmen. Je nach Modell variieren die Messwerte aus diesen beiden Datenquellen untereinander mehr oder weniger stark. Untersucht man die Ladegeschwindigeiten von EV wird schnell klar, warum diese Beobachtung gemacht werden kann. Die Laderate hängt von vielen weiteren Faktoren ab als lediglich des SoC, unter anderem die aktuelle Temperatur, der state of health (SoH). Diese Faktoren sind für die vorliegenden Messungen nicht bekannt. Es lassen sich dennoch Trends und Größenordnungen in den Ladekurven der beiden Datenquellen erkennen. Daher können die vorliegenden Daten von [12] für die Simulation genutzt werden.

3.3.2 Extraktion der Datenpunkte

Die Daten bei [12] lagen lediglich als Grafik vor. Um sie in der Simulation verwenden zu können, mussten die Grafiken in csv-Files überführt werden. Nach einiger Recherche stellte sich für diesen Vorgang die Web-Applikation WebPlotDigitizer [4] als besonders zielführend heraus. Mittels verschiedener Filter und Parameter konnten so zuverlässig die Grafen in Tabellen überführt werden. Bei [11] sind die Daten ebenfalls als Grafik auf der Website vorliegend. Zusätzlich findet sich bei jedem Modell eine Tabelle mit den Messwerten, welche einfach kopiert werden kann.

Anschließend wurden die Daten für die Simulation aufbereitet. Da die Ladekurven selten bei 0 % SoC starten, mussten für diese Bereiche Daten generiert werden. Gleiches gilt für den oberen Bereich der Ladekurve, wenn die Aufzeichnungen nicht bis 100 % SoC gehen. Hier wurde der jeweilige Bereich mit dem ersten bzw. letzten Wert für die Ladeleistung aufgefüllt.

In der nachstehenden Grafik sind die extrahierten und aufgearbeiteten Ladekurven abgebildet.

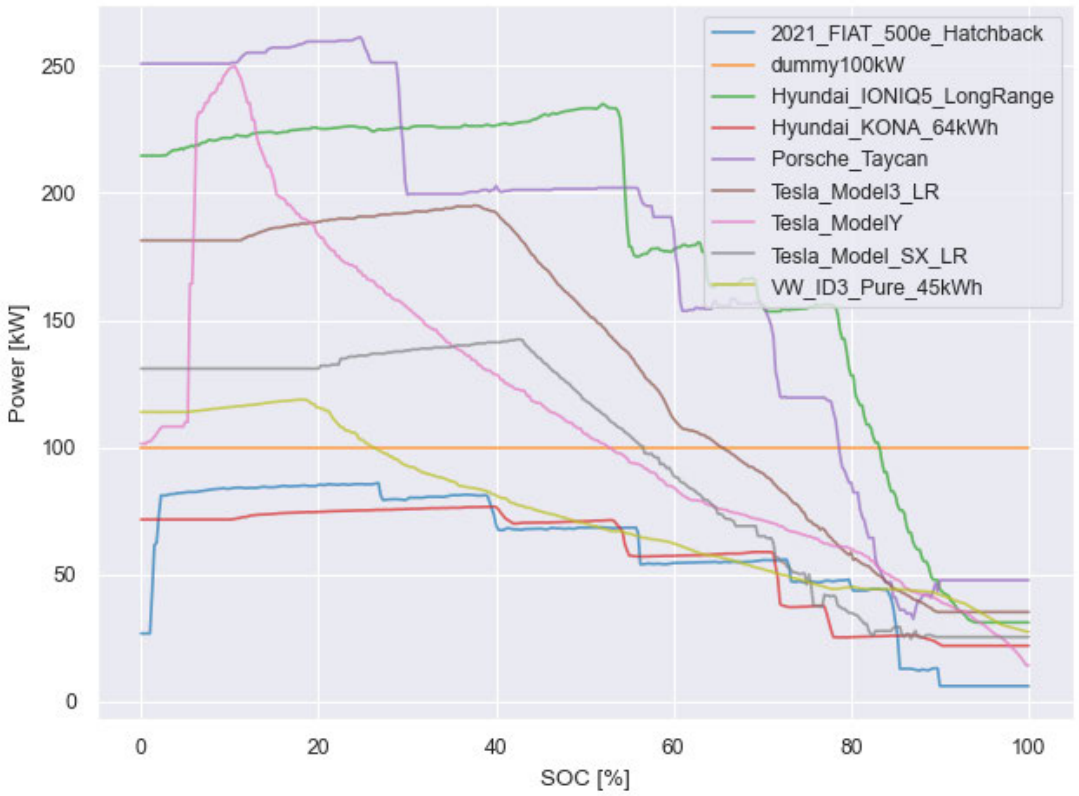

Abbildung 3.6: Aufbereitete Ladekurven

3.4 Klassen-Modell

Die Simulation wurde aus Gründen der Übersichtlichkeit und Wiederverwendbarkeit objektorientiert in verschiedenen Python-Klassen umgesetzt. In diesem Kapitel wird auf den Aufbau der Simulation sowie die einzelnen Klassen eingegangen. Folgende Grafik visualisiert die Klassen und deren Zusammenhänge:

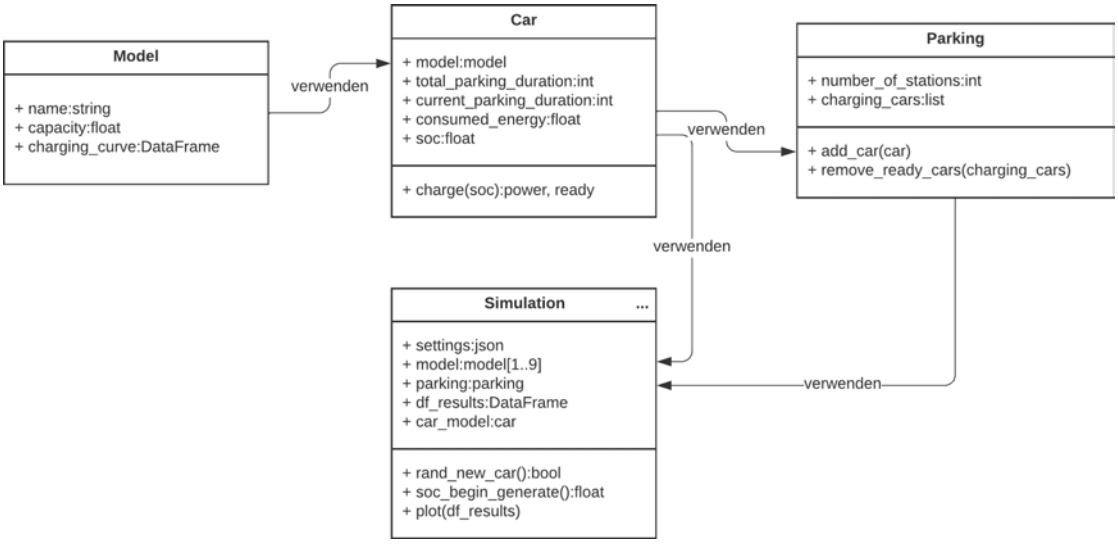

Abbildung 3.7: UML-Klassendiagramm

3.4.1 Klasse 'Model'

Mit der Klasse 'Model' wird zunächst ein Modell eines spezifischen EV erzeugt. Dazu wird der Name des bestimmten EV sowie die Akku-Kapazität in kWh an die Klasse übergeben. Mittels des übergebenen Namens wird nun die zugehörige Ladekurve aus der jeweiligen parquet-Datei geladen. Diese Ladekurve sowie der Name des EV und die Kapazität werden in dem Objekt gespeichert.

3.4.2 Klasse 'Car'

Immer wenn ein neues EV den Parkplatz erreicht und der Ladevorgang starten soll, wird ein Objekt der Klasse 'Car' erzeugt. Dieses beinhaltet das entsprechende Objekt 'Model' 3.4.1, die Gesamtparkzeit sowie den an die Klasse übergebenen initialen SoC. Weiterhin werden die Instanzen 'Aufgenommene Energie' und 'Aktuelle Parkdauer' erzeugt.

In der Klasse 'Car' wird die Funktion 'Charge' definiert. Hier findet der eigentliche Ladeprozess statt. Zu Beginn eines Aufrufs der Funktion wird der Index bestimmt, an welchem sich der aktuelle SoC in der entsprechenden Ladekurve befindet. Da die Ladekurven in 0,25%-Schritten vorliegen, wird der aktuelle SoC-Wert durch 0,25 geteilt. Wird der so

erhaltene Wert auf eine ganze Zahl gerundet, ergibt sich somit der Index der Ladekurve für den aktuellen SoC. Im nächsten Schritt wird nun die aktuelle Leistung für den Index aus der Ladekurve gelesen. Da die Simulation minutengenau arbeitet, muss der Leistungswert durch 60 geteilt werden. Es ergibt sich die Energiemenge in kWh, die in der aktuellen Minute dazugeladen wird. Der entsprechende Wert wird zur bereits aufgenommenen Energie dazu addiert. Außerdem wird mittels der dazu gekommenen Energie der neue SoC in % errechnet. Als Rückgabewert wird die aktuelle Leistung in kW, sowie das Flag 'ready-loaded' übergeben.

3.4.3 Klasse 'Parking'

In der Klasse 'Parking' wird gespeichert, wie viele Ladesäulen auf dem Parkplatz vorhanden sind. Diese Einstellung wird in der settings.json gesetzt. Weiterhin beinhaltet die Klasse eine Liste der EV, die aktuell laden. Kommen neue EV auf dem Parkplatz an, wird zunächst in der Funktion 'add Car' geprüft, ob eine freie Ladesäule vorhanden ist. Ist dies der Fall, wird das Auto zu der Liste der parkenden EV hinzugefügt. Die Funktion 'remove ready cars' entfernt die entsprechenden Fahrzeuge aus der Liste der geparkten EV, wenn die jeweilige Parkdauer abgelaufen ist.

3.5 Simulation

In diesem Abschnitt wird der Ablauf des Simulationsprozesses erläutert. Zu Beginn der Simulation wird von jedem möglichen EV ein Objekt der Klasse 'Model' 3.4.1 erzeugt und in einer Liste zusammengefasst. Weiter wird ein Objekt der Klasse 'Parking' erzeugt. Die entsprechenden Parameter werden aus der settings.json geladen. Als nächstes wird ein DataFrame 'df_results' erzeugt. Hier werden die Simulationsergebnisse gespeichert. Als Index für das DataFrame wird ein Timecode über einen Tag gewählt, als Frequenz Minuten. Somit ist das DataFrame 1441 Reihen lang, die Anzahl der Minuten eines Tages.

Nach der Initialisierung wird iterativ über einen Tag simuliert. Dabei wird für jede Minute eine Schleife durchlaufen. Zu Beginn eines Schleifendurchlaufs wird mittels der Funktion 'rand_new_car' bestimmt, ob in dieser Minute ein neues EV den Parkplatz erreicht. Dabei wird die jeweilige Benutzereinstellung aus der JSON berücksichtigt. Die Funktion gibt ein Bool zurück. Ist dieses gesetzt, ist ein neues EV bei einem Ladepunkt erschienen.

Als nächstes wird festgelegt, welches spezifische Modell erschienen ist. In der JSON kann der Benutzer in einer Liste Modelle angeben, aus denen, wie in 3.3.1 beschrieben, zufällig eins ausgewählt wird. Anschließend wird die Parkzeit für das neue EV bestimmt. Auch hier werden die Benutzereinstellungen aus der JSON gemäß 3.2.1 verwendet. Anschließend wird ein neues Objekt 'car' erzeugt und zu der Klasse 'parking' hinzugefügt.

Nachfoldend wird für jedes Objekt 'car' in 'parking' die Funktion 'charge' aufgerufen. Wie in 3.4.2 beschrieben wird der neue SoC, sowie die geladene Energiemenge für die aktuelle Minute bestimmt. Das Flag 'ready_loaded' wird gesetzt, wenn ein EV vollständig geladen ist. Außerdem wird die Leistung in kW zurückgegeben, welche der Ladevorgang aktuell in Anspruch nimmt.

Die Leistungswerte aller aktuell ladenden EV werden anschließend addiert und das Dataframe 'df_results' für den aktuellen Zeitstempel geschrieben. So kann über einen Tag der Lastverlauf aller Ladepunkte generiert werden.

4 Implementierung

Bereits in der Frühphase der Forschungsarbeit ergab sich die Notwendigkeit, die Auswahl der Instrumente zur Umsetzung der Simulation zu klären. Dazu zählt als wohl wichtigste Komponente die Wahl der Programmiersprache. Es war zudem erforderlich, eine Möglichkeit zu entwickeln, um den Code auf eine strukturierte und nachvollziehbare Weise an die Projektmitarbeiter zu übermitteln.

4.1 Wahl der Programmiersprache

Bei der Programmiersprache standen aufgrund meines Wissensstands für objektorientierte Programmiersprachen sinnvollerweise Python Java und Matlab zur Auswahl. Nach einer kurzen Recherche überwogen die Vorteile der Programmiersprache Python:

- State of the art
  Python wird gerade im Bereich Data Science und Simulationen in den letzten Jahren immer beliebter und wird für vergleichbare Projekte genutzt [17].

- Pandas und weitere Bibliotheken
  Um einfacher mit Daten umgehen zu können, kann in Python auf die Bibliothek 'Pandas' zurückgegriffen werden [24]. Der Umgang mit Daten, beipielsweise der Im- und Export, wird so erheblich vereinfacht. Weiterhin können Bibliotheken wie 'Matplotlib' genutzt werden für das Erstellen von Grafiken aus den Ergebnissen.

- Weitere Verwendung der Simulation
  Die Simulation soll in dem Projekt 'EcoCharge' weitere Verwendung im Bereich Deep-Learning finden. Wie bei dieser Technik üblich wird auch in diesem Projekt die Programmierung des neuralen Netzes mit Python realisiert [9]. Somit wird die Implementierung der Simulation um ein vielfaches vereinfacht, wenn diese ebenfalls in Python realisiert wird.

- Native Unterstützung in Jupyter-Notebook
  Obgleich Jupyter-Notebook auch andere Programmiersprachen wie Java, R und Matlab unterstützt, ist lediglich Python schon nativ implementiert. Somit muss kein weiterer Kernel installiert und konfiguriert werden, und das Arbeiten mit Jupyter-Notebook wird vereinfacht. [21]

4.1.1 Wahl der IDE

Für die Programmierung der Simulation wurde als Integrierte Entwicklungsumgebung (IDE) PyCharm ausgewählt. Diese IDE bietet sehr gute Debug-Optionen für Python, was bei der Programmierarbeit hilfreich ist. Weiterhin unterstützt Pycharm bereits in der Grundinstallation Jupyter-Notebook. Somit entfällt das manuelle Installieren und Einrichten eines Jupyter-Servers [15].

Das Repository beinhaltet keine Pycharm-spezifischen Daten. Somit kann es in jeder anderen IDE genutzt werden, die Python unterstützt. Bei der weiteren Verwendung der Simulation sind Benutzer also nicht auf eine bestimmte Software beschränkt.

4.2 Repository

Um die weitere Verwendbarkeit der Simulation zu gewährleisten, ist ein sinnvoller Aufbau des Repository wichtig. Hier soll darauf geachtet werden, dass die unterschiedlichen Szenarien einfach reproduzierbar bleiben.

Die nachstehende Grafik gibt einen Überblick über das Repository:

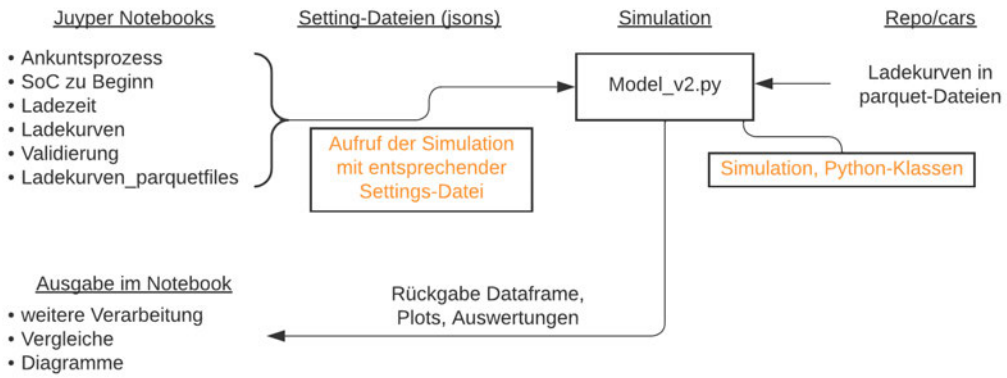

Abbildung 4.1: Aufbau des Repository

Die Simulation selber ist in einer Python-Datei geschrieben. Aufgerufen wird diese über verschiedene Jupyter-Notebooks. Für jede implementierte Funktion existiert ein eigenes Notebook. In diesem werden die jeweiligen Implemetationsschritte erläutert und validiert. Aus den Jupyter-Notebooks heraus wird die Simulation in der Python-Datei aufgerufen. Für jedes Notebook existiert eine eigene JSON-Settings-Datei. So können die unterschiedlichen Notebooks unabhängig ausgeführt werden, ohne die Einstellungen jedes Mal händisch anpassen zu müssen.

Die Ergebnisse der Simulation stehen in dem DataFrame 'df_results'. Für eine Grundauswertung existieren in der Python-Datei die Funktionen 'plot' und 'Auswertung'. Spezifische Auswertungen finden direkt in den jeweiligen Notebooks statt.

Die Ladekurven der EV liegen in dem Repository in dem Ordner 'cars' in parquet-Dateien vor. Sie werden beim Aufruf der Simulation automatisch eingelesen.

Die Aufbereitung der Ladekurven findet in dem Notebook 'Ladekurven_parquetfiles' statt. Somit können auch zu einem späteren Zeitpunkt mithilfe des in 3.3.2 erläuterten Vorgehens weitere EV zur Simulation hinzugefügt werden.

4.2.1 Github

Um das Repository mit den Teammitgliedern des Projektes zu teilen, wurde ein Github-Repo eingerichtet. So können auch schon Zwischenstände der Simulation unkompliziert

verwendet und implementiert werden. Ein gemeinsames Repo mit dem Deep-Learning-Verfahren und der Oberfläche für das 'EcoCharge'-Projekt wurde nach Absprache mit dem zuständigen Mitarbeiter nicht für nötig befunden. Stattdessen wird bei Bedarf der aktuelle Stand der Simulation händisch übernommen. Dennoch ist das Arbeiten mit einem Github-Repository sinnvoll, um die Änderungen nachvollziehen zu können, welche im Laufe der Arbeit getätigt werden.

4.3 Bedienung der Simulation

Um die finale Simulation mit eigenen Einstellungen durchzuführen, befinden sich in dem Repository die Dateien 'model_final.py' sowie die zugehörige JSON 'settings_final.json'. Der Benutzer kann die Simulation direkt in der Python-Datei ausführen. In dem zugehörigen Jupyter-Notebook 'Simulation_final.ipynb' wird dem Benutzer des weiteren exemplarisch das Arbeiten mit der Simulation vermittelt. Dabei werden einige nützliche Funktionen und Auswertungsmethoden erklärt.

Um der Simulation weitere Ladekurven von EV hinzuzufügen, müssen diese zunächst im CSV-Format vorliegen. Eine Möglichkeit für die Datengewinnung wird in 3.3.2 beschrieben. Anschließend müssen die Daten wie in 3.3.2 erklärt aufgearbeitet und in einer parquet-Datei abgespeichert werden. Das entsprechende Skript findet sich in dem Jupyter-Notebook 'Ladekurven_parquetfiles'. Hier wird ausführlich beschrieben, wie der Benutzer dabei vorgehen muss.

5 Szenarien und Auswertung

In diesem Kapitel wird das Vorgehen erläutert, mit dem verschiedene Szenarien simuliert werden. Dabei wird sich an die Reihenfolge gehalten, in der die Szenarien über den Zeitraum der Bachelorarbeit erarbeitet wurden. Es werden nach und nach weitere Funktionalitäten entwickelt und der Einfluss bestimmter Faktoren untersucht.

5.1 Umsetzung der Anforderungen

Um die Anforderungen aus 3.1 erfüllen zu können, müssen unterschiedliche Funktionen implementiert werden. Die Grundfunktionen, welche in Abbildung 3.1 markiert sind, wurden als erstes umgesetzt. Anschließend wurde die Simulation wie in 5.2 validiert. So konnte sichergestellt werden, dass die Simulation fehlerfrei funktioniert und mögliche methodische Fehler bei der Umsetzung beseitigt wurden. Anschließend wurden die entsprechenden Funktionen schrittweise ergänzt, die jeweiligen Auswirkungen untersucht sowie Szenarien entwickelt und ausgewertet.

5.2 Validierung des Modells

Um die korrekte Funktionsweise des Simulationsmodells zu validieren und sicherzustellen, dass keine logischen Fehler vorliegen, wird ein Dummy-Modell erstellt. Dieses weist eine Akkukapazität von 100 kWh und eine konstante Ladeleistung von 100 kW auf. Somit müssen die Lastkurve und die Anzahl der ladenden EVs immer deckungsgleich sein. Dies ließ sich in dem resultierenden Lastverlauf bestätigen:

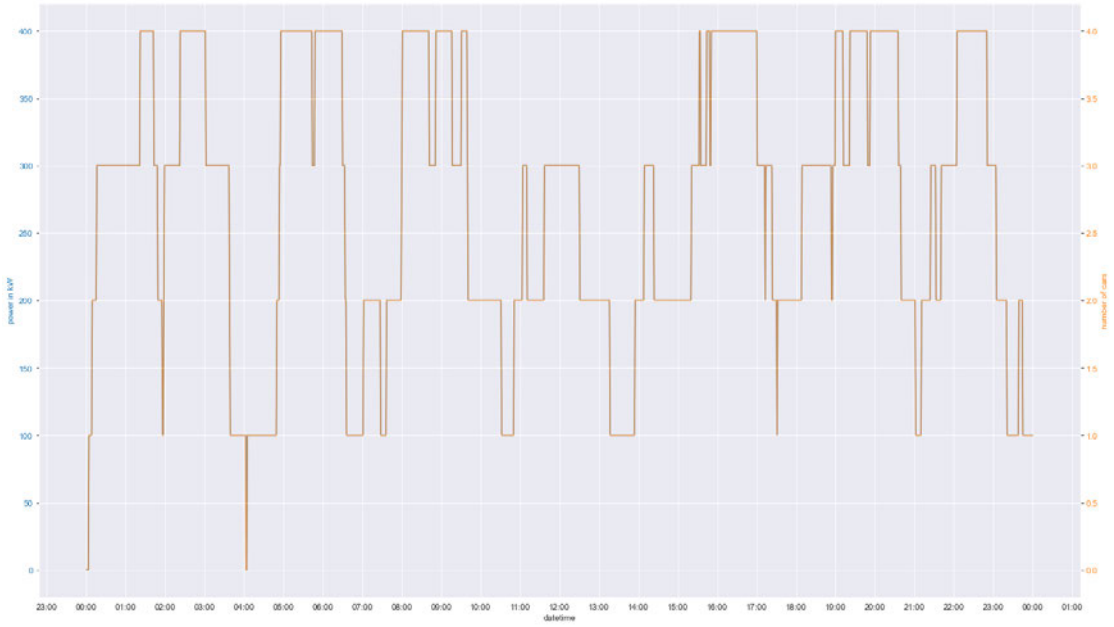

Abbildung 5.1: Simulation mit Dummy-EV: 100kW Ladeleistung

Mit diesem generierten Lastverlauf wurde lediglich die Grundfunktionalität der Simulation sowie einfache Auswertungsmethoden verifiziert. Der Ankunftsprozess sowie die Parkdauer sind in diesem Simulationsdurchlauf mit einfachen Zufallsfunktionen randomisiert.

5.3 Simulations-Ergebnisse

In den folgenden Unterkapiteln werden die Auswirkungen der Parameter unter 3.2 auf das Simulationsergebnis untersucht. Aus Gründen der Vergleichbarkeit werden dabei für die Parameter, die gerade nicht untersucht werden, immer dieselben Einstellungen genutzt. Diese Standard-Einstellungen sind folgender Tabelle zu entnehmen:

Tabelle 5.1: Standard-Parameter für die Simulation

Parameter	**Parameter Wert**
list_of_cars	Liste aller EVs aus 3.3
soc_begin	gauss
soc_gauss_von	0
soc_gauss_bis	65
soc_gauss_sigma	10
arriving_process	poisson
arriving_process_poisson_lambda	1.35
number_of_stations	4
max_power_per_station	300
accessible_all_day	true
ChargingTime	Weibull mit Parametern aus 3.2.1

5.3.1 Auswertung initialer SoC

Im ersten Schritt liegt der Fokus auf dem SoC zu Beginn des Ladevorgangs. Da die Ladekurven bei niedrigeren SoC-Werten einen signifikant höheren Lastgang aufweisen als bei SoC-Werten > 50 %, wird angenommen, dass der SoC zu Beginn des Ladevorgangs einen hohen Einfluss auf den Lastverlauf der Simulation hat.

Zunächst wurde mit dem ungünstigsten Fall simuliert, bei dem die Lastspitzen am größten sind. Dieser Fall tritt ein, wenn alle EV zu Beginn des Ladevorgang einen SoC von 0 % haben. Dies ist natürlich keinesfalls realistisch und dient lediglich als Referenz, um die Auswirkungen des SoC beurteilen zu können.

Das Ergebnis wird in 5.3 mit den anderen Methoden verglichen.

Als nächstes wurde der SoC zu Beginn mit Zufallsverfahren bestimmt. Dabei wurden zwei unterschiedliche Verfahren gewählt: gleichverteilt und eine Gauß-Normalverteilung. In beiden Fällen kann der Benutzer in den Einstellungen wählen, in welchem Bereich der SoC zu Beginn liegen soll. Im Folgenden wurden gemäß [14] für beide Verteilungen Bereiche von 0 % bis 65 % SoC eingestellt.

Für die Gauß-Normalverteilung wurde eine einfache Normalverteilung implementiert. Die Verteilung wird in nachstehender Grafik veranschaulicht:

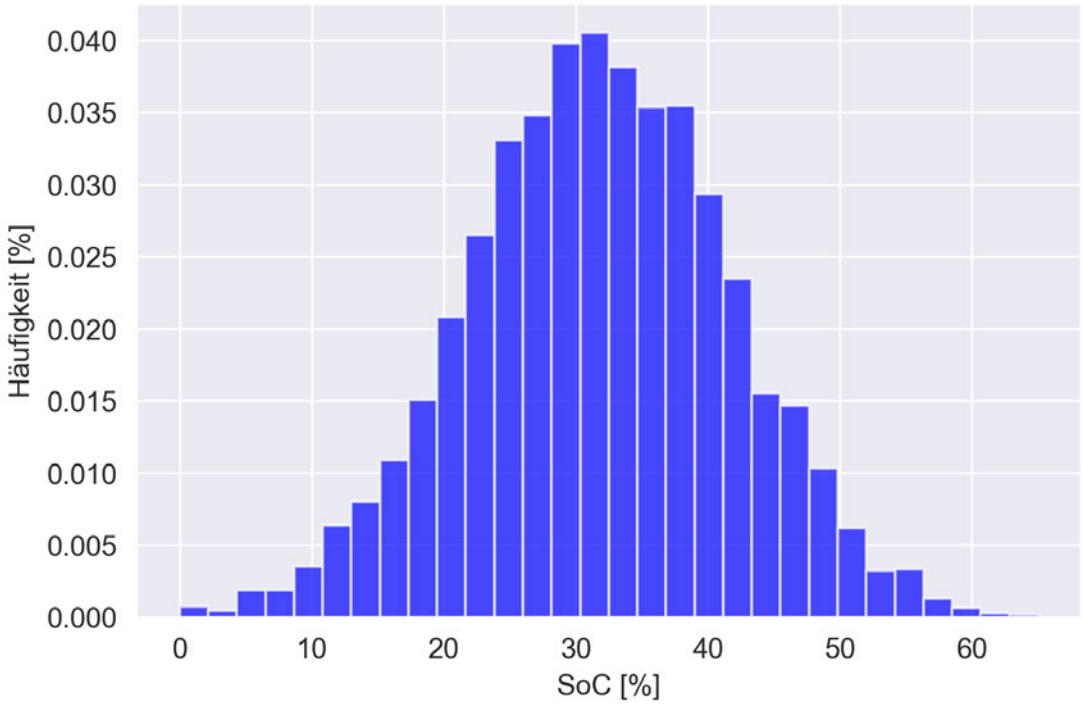

Abbildung 5.2: Normalverteilung für den SoC zu Beginn

Die unterschiedlichen Methoden zur Bestimmung des SoC zu Beginn des Ladevorgangs haben Einfluss auf den Lastverlauf der Ladepunkte. Um diesen Einfluss untersuchen zu können, werden sie in einer empirischen Verteilungsfunktion dargestellt. Diese Funktion bietet eine bessere Vergleichbarkeit als ein normales Histogramm.

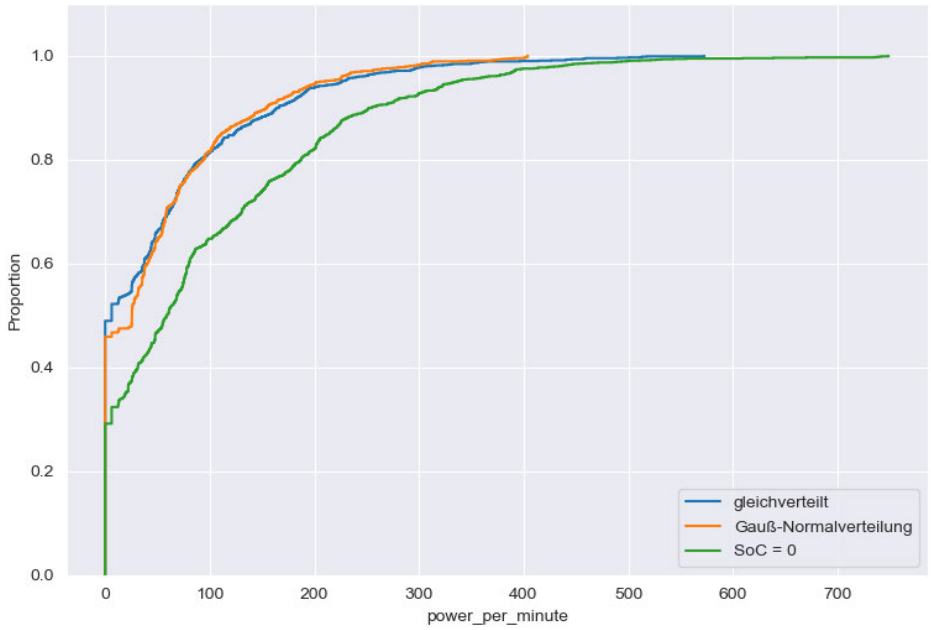

Abbildung 5.3: Empirische Verteilungsfunktion der unterschiedlichen SoC zu Beginn

In den Last-Verteilungen lässt sich gut erkennen, dass sich die Gleichverteilung und Gauß-Normalverteilung des SoC zu Beginn des Ladevorgangs nicht signifikant unterscheiden. Wird hingegen mit einem SoC zu Beginn von 0 % simuliert, sind höhere Lastspitzen zu verzeichnen. Dieses Ergebnis deckt sich mit der zuvor getroffenen Annahme.

Im weiteren Verlauf der Simulation wird mit einer Gauß-Normalverteilung gearbeitet, um den initialen SoC zu bestimmen.

5.3.2 Auswertung Ankunftsprozess

In diesem Abschnitt wird der Ankunftsprozess der EV ausgewertet. Die in 3.2.4 beschriebenen Methoden zur Modellierung werden dafür miteinander verglichen. Es wurde einmal mit einer Gleichverteilung und einmal mit dem Poisson-Prozess simuliert.

Nachfolgend sind als Simulationergebnisse die Lastverläufe mit diesen beiden Einstellungen abgebildet. Die Vergleichbarkeit zwischen diesen Verläufen ist nicht besonders hoch.

Daher wird im weiteren Verlauf der Arbeit mit zusätzlichen Auswertungsmethoden gearbeitet.

Als Ankunftsrate wird zunächst bei beiden Methoden 1.35 EV pro Stunde eingestellt.

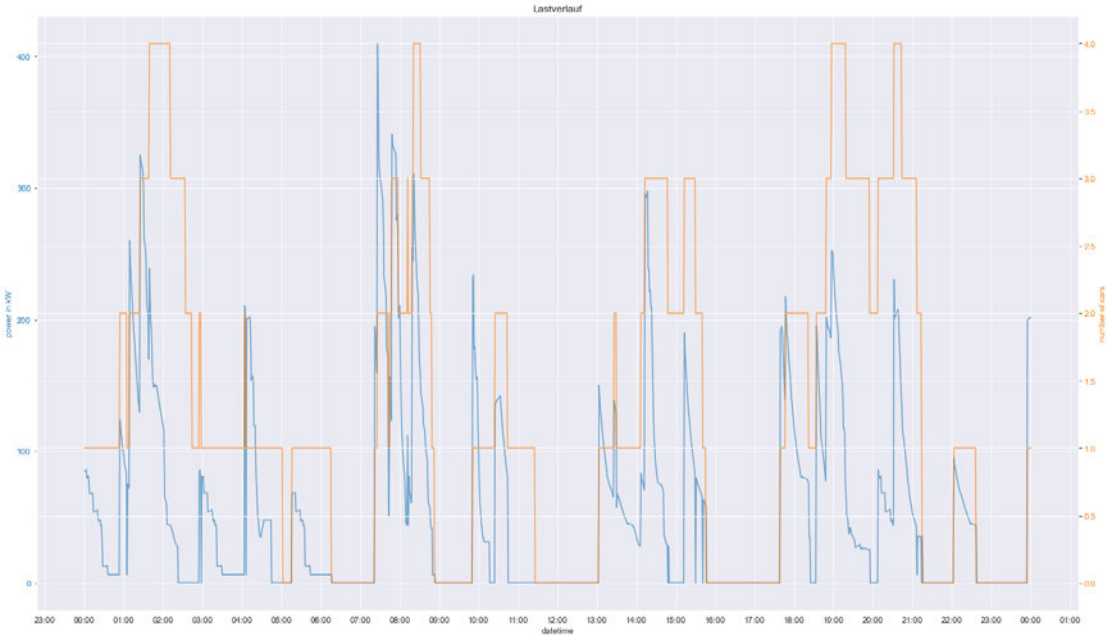

Abbildung 5.4: Lastverlauf über einen Tag, Ankunftsprozess zufällig gleichverteilt

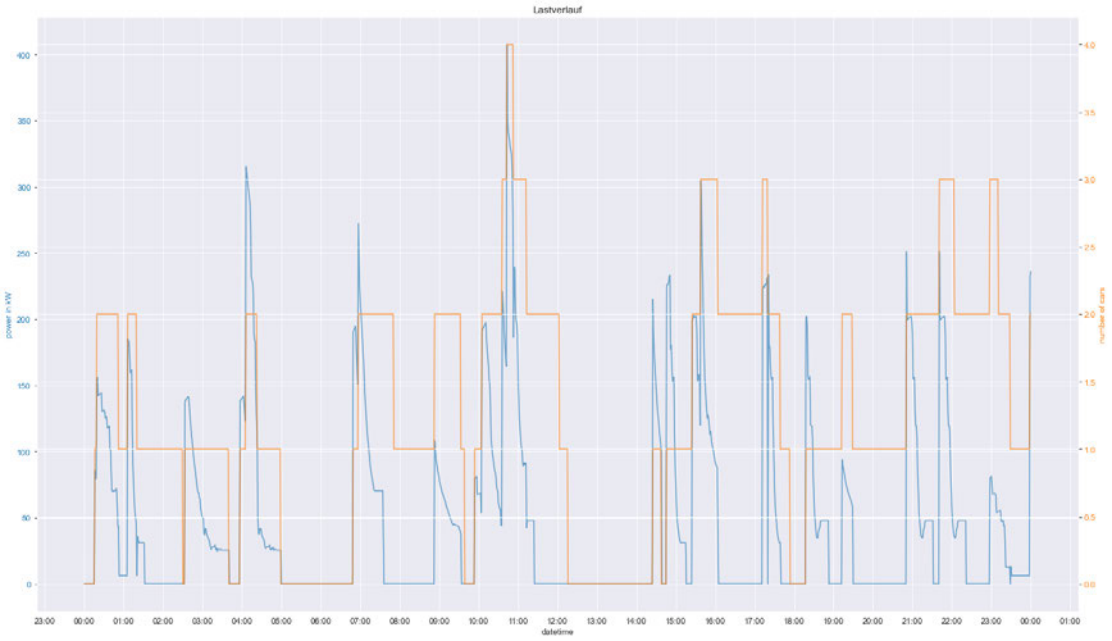

Abbildung 5.5: Lastverlauf über einen Tag, Ankunftsprozess Poisson-Prozess

Um die Simulationsergebnisse und die Auswirkungen der unterschiedlichen Ankunftsprozesse auf diese besser untersuchen zu können, wird die Simulation jeweils über 100 Tage simuliert ausgeführt. Dabei wird besonders die Gesamtanzahl der erschienenen EV betrachtet sowie die aufgrund eines zur Zeit vollständig belegten Parkplatzes abgewiesenen EV.

Tabelle 5.2: Ankunftsprozesse mit Lambda = 1,35 EVs / Stunde

	EVs gesamt / Stunde	abgewiesene EVs / Tag
zufällig gleichverteilt	1,27	1,02
Poisson-Prozess	1,28	1,09

In den folgenden Durchläufen wird die durchschnittliche Ankunftsrate auf 2 EV pro Stunde erhöht:

Tabelle 5.3: Ankunftsprozesse mit Lambda = 2,0 EVs / Stunde

	EVs gesamt / Stunde	abgewiesene EVs / Tag
zufällig gleichverteilt	1,87	3,93
Poisson-Prozess	1,92	4,25

Anschließend wird die Ankunftsrate nochmals auf 3 EV pro Stunde erhöht:

Tabelle 5.4: Ankunftsprozesse mit Lambda = 3,0 EVs / Stunde

	EVs gesamt / Stunde	abgewiesene EVs / Tag
zufällig gleichverteilt	2,78	12,52
Poisson-Prozess	2,91	14,41

Aus den Ergebnissen lässt sich der Trend erkennen, dass unter Verwendung des Poisson-Prozesses bei gleicher durchschnittlicher Ankunftsrate öfter der Fall eintritt, dass mehrere Kunden gleichzeitig in einem aktiven Ladevorgang sind und neu ankommende EV abgewiesen werden müssen. In dem zugehörigen ECDF-Plot lässt sich jedoch erkennen, dass die Verwendung der Warteschlagentheorie und des gleichverteilten Zufallsgenerators für den Ankunftsprozess nur geringe Auswirkungen auf den Lastverlauf der Ladepunkte hat.

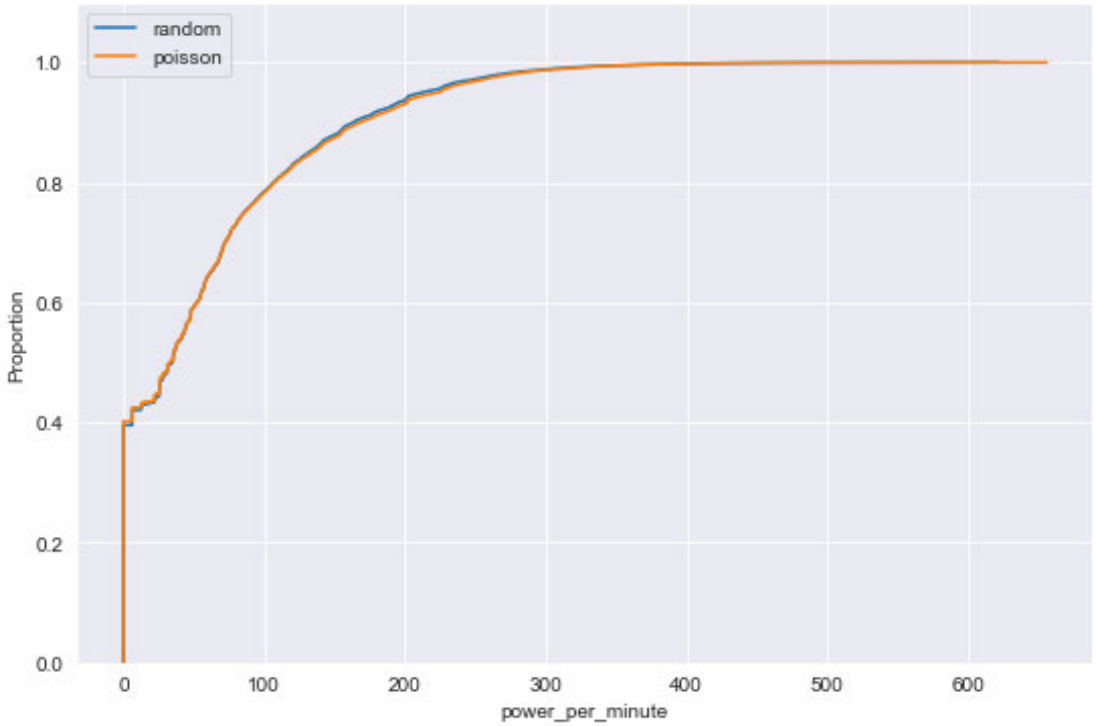

Abbildung 5.6: ECDF-Plot Lastverlauf der unterschiedlichen Ankunftsprozesse. Lamda = 1.35 EVs pro Stunde, 100 Simulationsdurchläufe

Im weiteren Verlauf der Arbeit wird mit der Warteschlangentheorie simuliert, da diese den Ankunftsprozess realisitscher abbildet als eine gleichverteilte Zufallsfunktion.

5.3.3 Auswertung Ladezeit

Wie in 3.2.1 beschrieben, wird die Ladezeit mittels zwei unterschiedlichen Methoden modelliert. In dem ersten Implementierungsschritt wird die Parkdauer und die damit verbundene Ladezeit des Kunden mittels Zufallsverfahren umgesetzt. Der Benutzer kann in den Einstellungen ein Zeitintervall angeben, innerhalb dessen sich die Parkdauer der Fahrzeuge befinden soll. In dem zweiten Implementierungsschritt werden die unterschiedlichen Zeiten der Parkdauer aus [18] genutzt, um die Ladezeiten zu bestimmen. Näheres wird in Kapitel 3.2.1 beschrieben.

Um die Auswirkungen der verschiedenen Modellierungsmethoden auf das Simulationsergebnis zu untersuchen, wird mit beiden Methoden simuliert und die Ergebnisse verglichen. Die restlichen Einstellungen bleiben jeweils unverändert.

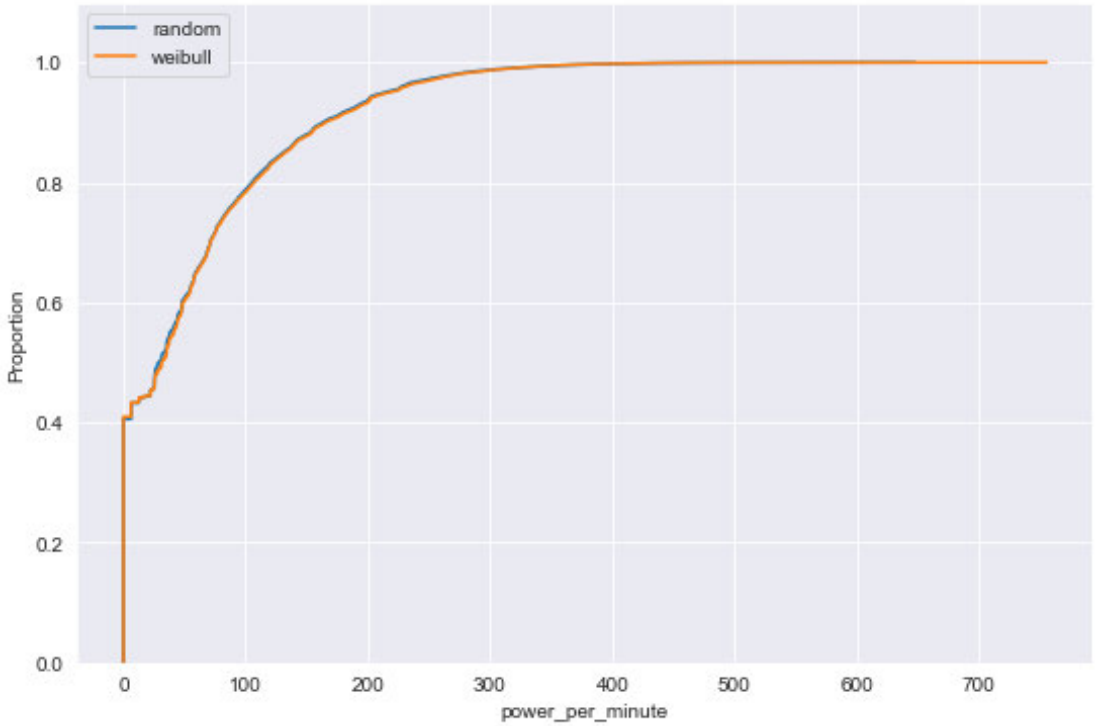

Abbildung 5.7: ECDF-Plot Lastverlauf der unterschiedlichen Modellierungen der Ladezeiten, 100 Simulationsdurchläufe

Wie zu erkennen ist, wirkt sich die Verwendung der unterschiedlichen Modellierungsmethoden nicht signifikant auf den Lastverlauf der Ladepunkte aus. Auch die Anzahl aufgrund belegter Ladepunkte abgewiesener EV ist bei beiden Methoden ähnlich. Da das Fahrerverhalten in der Realität akkurater durch [18] beschrieben wird, wird in der finalen Simulation diese Methode genutzt.

5.3.4 Auswirkung der verschiedenen Ladekurven

In diesem Abschnitt wird die Auswirkung der unterschiedlichen Ladekurven als Input-Parameter auf das Simulationsergebnis untersucht. Wie in 3.3 beschrieben, stehen un-

terschiedliche Ladekurven in der Simulation zur Auswahl. Mittels einer Liste in den Einstellungen kann der Benutzer festlegen, welche dieser Ladekurven genutzt werden sollen.

Um den Einfluss der Modellauswahl auf den Lastverlauf der Ladepunkte zu untersuchen, wird mit drei unterschiedlichen Einstellungen für die Auswahl der Modelle gearbeitet. In dem Worst-Case-Szenario wird ausschließlich mit dem Modell simuliert, welches die höchsten Lastspitzen in der Ladekurve aufweist. Wie unter 3.6 zu erkennen, ist dies bei den ausgewählten EV bei dem Porsche Taycan der Fall. Dieses Modell weißt Lastspitzen von bis zu 270 kW auf. Als Resultat sind hier höhere Lastspitzen in dem Simulationsergebnis zu erwarten. Analog dazu tritt das Best-Case-Szenario mit den geringsten Lastspitzen im Simulationsergbnis dann ein, wenn lediglich mit EV-Modellen simuliert wird, welche eine niedrige Ladegeschwindigkeit aufweisen. Unter den gewählten Modellen in 3.3 ist das EV mit der niedrigsten Lastspitze der Hyundai KONA. (3.6) Als Referenz wird das dritte Szenario mit den Standard-Einstellungen simuliert. Hier wird aus der gesamten Liste aus 3.3 simuliert.

Für die Auswertung der drei Szenarien wird ein ECDF-Plot des Lastverlaufes der Ladepunkte aus jeweils 100 Durchläufen erstellt:

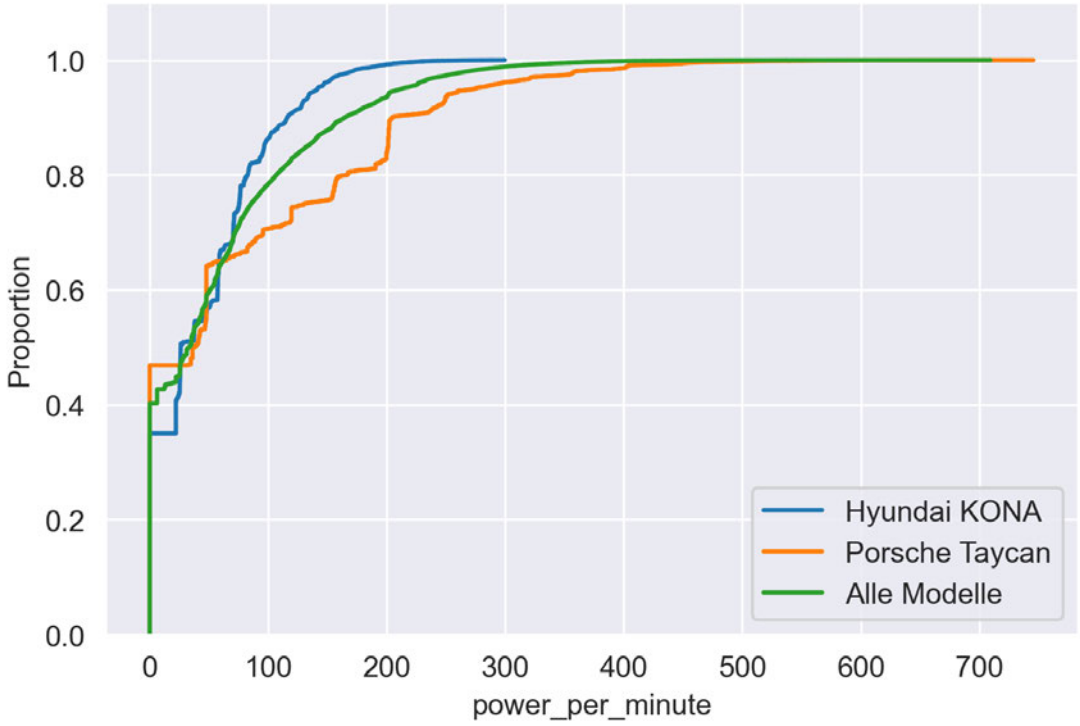

Abbildung 5.8: ECDF-Plot Lastverlauf der unterschiedlichen Ladekurven, jeweils 100 Simulationsdurchläufe

Die Abbildung 5.8 entspricht den Erwartungen. Die höchsten Lastspitzen an den vier Ladepunkten mit bis zu 830 kW treten bei dem Porsche Taycan auf. Wird hingegen ausschließlich mit dem Hyundai KONA simuliert, sind lediglich Lastspitzen bis maximal 300 kW zu verzeichnen. Wird mit allen Modellen simuliert, wurde die höchste Lastspitze bei 633 kW gemessen.

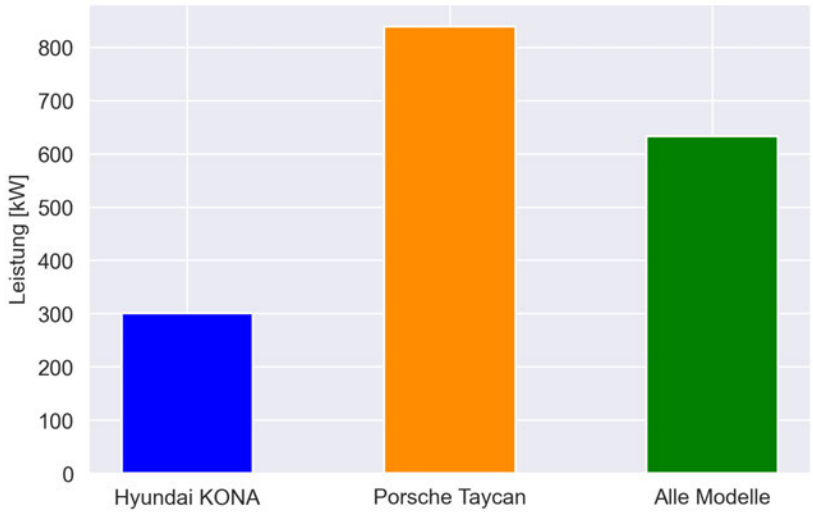

Abbildung 5.9: Lastspitzen abhängig von der Auswahl der Modelle

Diese Werte können abhängig von der Belegung der Ladesäulen und weiteren Parametern, welche von Zufallsverfahren beeinflusst werden, leicht variieren. Jedoch ist gut erkennbar, dass das Simulationsergebnis stark abhängt von der Auswahl der ladenden EV.

6 Fazit

Im Mittelpunkt steht als Ergebnis der Arbeit das Simulationsprogramm. Unter 3.1 sind die Anforderungen an die Simulation aufgeführt. Die wesentlichen Anforderungen sind implementiert und in 5.3 ausgewertet worden. Die Simulationsergebnisse könnten jedoch noch weiter verbessert werden, wenn verschiedene Faktoren zusätzlich berücksichtigt würden. Neben den in 3.1 bereits genannten optionalen Parametern 'Schrittweite der Simulation', 'Wochentag oder Wochenende', 'Tag vor Feiertag' und 'Standort des Supermarktes', sind im Laufe der Arbeit noch weitere Faktoren aufgefallen, welche in der Zukunft in die Simulation mit implementiert werden könnten. Den wohl größten Effekt auf die Simulationsgenauigkeit hat dabei der Einfluss der Temperatur. Laut [3] laden EV bei niedrigen Temperaturen langsamer. Da die Akkus jedoch nach dem Fahren eine höhere Temperatur als lediglich die Umgebungstemperatur aufweisen, wäre die Umsetzung in der Simulation mit hohem Aufwand verbunden. Eine weitere bisher nicht genannte Verbesserung wäre die Berücksichtigung der unterschiedlichen Zulassungszahlen bei der Auswahl der Modelle unter 3.3. Weiterhin ist die Ankunftsrate über einen Tag verteilt konstant und wird lediglich durch die Öffnungszeiten begrenzt. Hier ließe sich zur Verbesserung der Simulation eine variable Ankunftsrate implementieren.

Die Software arbeitet iterativ, um die Simulationsergebnisse zu berechnen. Die Berechnung kann dabei einige Zeit in Anspruch nehmen. So dauert das Simulieren über 100 Tage je nach verwendeter Hardware zwischen ein bis zwei Minuten. Soll im weiteren Projektverlauf ein größerer Datensatz, beispielsweise mit verschiedenen Parametereinstellungen parallel erzeugt werden, könnte eine Performance-Optimierung der Software von Vorteil sein.

Zusammenfassend lässt sich sagen, dass die Arbeit die erfolgreiche Implementierung und Analyse eines Simulationsprogramms hervorhebt. Dennoch werden potenzielle Verbesserungen für zukünftige Forschungen identifiziert. Die Simulation erweist sich als robust, aber eine Performance-Optimierung könnte ihre Effizienz bei der Verarbeitung umfangreicherer Datensätze steigern.

Im Kontext des bereits angelaufenen 'EcoCharge'-Projekts sind mehrere Mitarbeiter bereits aktiv in die anspruchsvolle Programmierung der benötigten Software vertieft. Um den Fortschritt dieser Arbeit nicht zu behindern, wurde dem Team vorübergehend eine exklusive Vorabversion der entwickelten Simulation über Github zur Verfügung gestellt. Dieser strategische Schritt erwies sich als unerlässlich, um beispielsweise die Grundfunktionen der Benutzeroberfläche in einem frühen Stadium gezielt entwickeln zu können.

Nach sorgfältigem Abschluss dieser initialen Phase wird die ausgearbeitete Simulation in einer formalen Übergabe an das Projektteam weitergereicht. Damit wird die Simulation zu einem integralen Bestandteil der umfassenden Software, die im Rahmen des 'EcoCharge'-Projekts entwickelt wird. Auf diese Weise wird die Simulation einen entscheidenden Beitrag zur Verwirklichung der Ziele von 'EcoCharge' leisten.

Literaturverzeichnis

[1] : *JSON Definition.* – URL https://www.json.org/json-de.html. – Zugriffsdatum: 20.12.2023

[2] ADAC: *Elektroauto laden: Das sind die passenden Kabel und Steckertypen.* – URL https://www.adac.de/rund-ums-fahrzeug/elektromobilitaet/laden/elektroauto-ladekabel-steckertypen/. – Zugriffsdatum: 21.01.2024

[3] ADAC: *Schnellladen Elektroauto: Die besten Modelle für die Langstrecke.* – URL https://www.adac.de/rund-ums-fahrzeug/elektromobilitaet/tests/schnellladen-langstrecke-ladekurven/. – Zugriffsdatum: 20.01.2024

[4] Ankit Rohatgi: *WebPlotDigitizer - Extract data from plots, images and maps.* – URL https://automeris.io/WebPlotDigitizer/. – Zugriffsdatum: 25.12.2023

[5] Bundesministerium für Wirtschaft und Klimaschutz: *Elektromobilität (Definition i.S. der Bundesregierung).* – URL https://www.erneuerbar-mobil.de/glossar/elektromobilitaet-definition-der-bundesregierung. – Zugriffsdatum: 20.01.2024

[6] Bundesnetzagentur: *Anzahl der öffentlichen Ladepunkte in Deutschland von Januar 2017 bis Juli 2023.* – URL https://de.statista.com/statistik/daten/studie/1190896/umfrage/ladesaeulen-in-deutschland/. – Zugriffsdatum: 25.09.2023

[7] Campus Verlag: *Definition Simulationsmodell.* – URL https://www.onpulson.de/lexikon/simulationsmodell/. – Zugriffsdatum: 18.12.2023

[8] DA-DIREKT: *Ladestecker fürs Elektroauto: Übersicht der gängigsten Typen.* – URL https://www.da-direkt.de/elektroauto-versicherung/ratgeber/ladestecker-fuer-elektroautos. – Zugriffsdatum: 21.01.2024

[9] DEEPLEARNING.AI: *What is Pandas - Pandas Introduction.* – URL ACompleteGuidetoNaturalLanguageProcessing. – Zugriffsdatum: 23.01.2024

[10] ENVIDATEC GROUP: *Envidatec - Energieberatung im Sinne der Nachhaltigkeit.* – URL https://envidatec.com/. – Zugriffsdatum: 23.01.2024

[11] EVKX.NET: *EVKX Database - Datenbank für fast alle EVs inklusive der zugehörigen Ladekurven.* – URL https://evkx.net/evsearch/. – Zugriffsdatum: 16.12.2023

[12] FASTNED: *Fastned FAQ - Auflistung der Fahrzeuge inklusive zugehöriger Ladekurven.* – URL https://support.fastned.nl/hc/de/sections/4428932764573-Fahrzeuge. – Zugriffsdatum: 15.12.2023

[13] GOINGELECTRIC.DE: *Zulassungszahlen von Elektroautos im Jahr 2019.* – URL https://www.goingelectric.de/zulassungszahlen/2019/. – Zugriffsdatum: 20.01.2024

[14] HERTLEIN, Timo A. ; BLENK, Tobias ; WEINDL, Christian ; OCHS, Joerg: Object-Oriented Charging Model for the Simulation of Grid-Serving Intelligent Charging Infrastructure. In: *ETG Congress 2023* VDE (Veranst.), 2023, S. 1–8

[15] JETBRAINS.COM: *Funktionen von Pycharm.* – URL https://www.jetbrains.com/de-de/pycharm/features/. – Zugriffsdatum: 23.01.2024

[16] KARLE, Anton: *Elektromobilität: Grundlagen und Praxis.* Carl Hanser Verlag GmbH Co KG, 2022

[17] KEVIN D.DAVIS: *Top 11 Programming Languages for Data Science.* – URL https://www.knowledgehut.com/blog/data-science/programming-languages-for-data-science. – Zugriffsdatum: 23.01.2024

[18] KLITZSCH, Cathrin: Elektromobilität im Handel 2023. In: *ETG Congress 2023* EHI Retail Institute e. V (Veranst.), 2023, S. 1–21

[19] Kraftfahrt-Bundesamt: *Anteil der Elektroautos am Bestand der Personenkraftwagen in Deutschland von 2013 bis 2023*. – URL https://de.statista.com/statistik/daten/studie/784986/umfrage/marktanteil-von-elektrofahrzeugen-in-deutschland/. – Zugriffsdatum: 20.01.2024

[20] Laurenz Wuttke: *Deep Learning: Definition, Beispiele & Frameworks*. – URL https://datasolut.com/was-ist-deep-learning/. – Zugriffsdatum: 23.01.2024

[21] Mendez, Kevin M. ; Pritchard, Leighton ; Reinke, Stacey N. ; Broadhurst, David I.: Toward collaborative open data science in metabolomics using Jupyter Notebooks and cloud computing. In: *Metabolomics* 15 (2019), S. 1–16

[22] python.org: *What is Python? Executive Summary*. – URL https://www.python.org/doc/essays/blurb/. – Zugriffsdatum: 23.01.2024

[23] w3schools: *Pandas DataFrames - What is a DataFrame?*. – URL https://www.w3schools.com/python/pandas/pandas_dataframes.asp,urldate={23.01.2024}

[24] w3schools: *What is Pandas - Pandas Introduction*. – URL https://www.w3schools.com/python/pandas/pandas_intro.asp. – Zugriffsdatum: 23.01.2024

[25] Waldmann, Karl-Heinz ; Helm, Werner E.: *Poisson-Prozesse*. S. 247–260. In: *Simulation stochastischer Systeme: Eine anwendungsorientierte Einführung*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2016. – URL https://doi.org/10.1007/978-3-662-49758-6_9. – ISBN 978-3-662-49758-6

A Anhang

Der weitere Anhang zur Arbeit befindet sich auf CD und kann beim Erstgutachter eingesehen werden.

Erklärung zur selbstständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original