FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Fachhochschule Aachen
Campus Jülich
Fachbereich 9 Medizintechnik und
Technomathematik

Studiengang Angewandte Mathematik und Informatik, B. Sc.

Konzeption und Umsetzung einer mobilen Anwendung zur Bereitstellung von KI-Modellen für Android-Apps am Beispiel einer Schlafphasenanalyse

Bachelorarbeit von Steffen Wolf
Matrikelnummer: 3077834

Diese Arbeit wurde betreut von:
Prof. Dr. rer. nat. Alexander Voss
Philipp Kohl, M. Sc.

Jülich, 20. Januar 2025

Eigenständigkeitserklärung

Diese Arbeit ist von mir selbstständig angefertigt und verfasst. Es sind keine anderen als die angegebenen Quellen und Hilfsmittel benutzt worden.

Jülich, 20. Januar 2025

.....

Steffen Wolf

Zusammenfassung

Die Nutzung von KI-Modellen zur Auswertung von großen Datenmengen hat in der Vergangenheit nicht nur in der Wissenschaft an großer Beliebtheit gewonnen. Auch im Alltag werden immer mehr Daten durch KI-Modelle ausgewertet um den Lebensstandard der Menschen zu erhöhen. Neben den großen Modellen, wie ChatGPT, die zur Generierung von Texten dienen gibt es auch Modelle, die Sensordaten aus Smartwatches analysieren um zum Beispiel das Schlafverhalten des Trägers zu bestimmen. Diese Modelle basieren meist auf serverseitiger Berechnungen. Es müssen also persönliche Daten auf fremde Server hochgeladen werden. Eine alternative Herangehensweise wäre es, die Berechnungen auf dem Smartphone durchführen zu lassen. In der folgenden Arbeit wird dieser Ansatz in einer Android-App konzipiert und umgesetzt. Die App ermöglicht es Daten lokal zu speichern, mittels lokaler TFLite-Modelle zu verarbeiten und diese Daten anderen Apps zur Verfügung zu stellen. Es wird dabei auf eine saubere Programmstruktur geachtet, um zukünftige Erweiterungen und Anpassungen zu erleichtern. Neben der Anwendung wird ein KI-Modell zur Schlafphasenanalyse erstellt um die Nutzung der App zu demonstrieren. Für die Erstellung, das Training, die Konvertierung und das Konfigurieren der Modelldatei wurden Python-Skripte angelegt, die für andere Anwendungsdomänen oder neue Trainingsdaten leicht anzupassen sind. Der Quellcode der Anwendung sowie die Python-Skripte sind öffentlich unter https://github.com/levithas/Bachelorprojekt zugänglich.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einführung

Künstliche Intelligenz (KI) ist heute aus vielen Lebensbereichen nicht mehr wegzudenken. Sie hilft, große Datenmengen, die zu komplex für herkömmliche Algorithmen sind zu verarbeiten und die Informationen verständlicher zu machen [19]. Viele Unternehmen, die KI-Modelle anbieten, gewähren jedoch keinen Einblick in ihre Systeme. Die Modelle laufen auf Servern und müssen daher die Daten, über das Internet empfangen. Für sensible Daten, wie Geschäftsprozesse, persönliche Gesundtheitsinformationen oder andere Daten privater Natur, setzt dies ein Vertrauen an den Serverbetreiber und die eingesetzte Übertragungstechnologie voraus. In der Regel bieten diese Unternehmen keine Möglichkeit zur Weiterentwicklung und keinen Zugang für unabhängige Entwickler. Es gibt zwar kostenpflichtige Schnittstellen für Modelle, wie ChatGPT [**OpenAIPlatform**], jedoch können die Kosten für den Zugriff hier schnell sehr groß werden. Für die Auswertung von kleineren Datenmengen, wie zum Beispiel bei der Schlafphasenanalyse wäre ein solches Modell außerdem überdimensioniert. Eine Alternative wäre hier die Nutzung von Cloud-Anbietern, wie AWS [**CloudComputingDienste**] oder Google [**GoogleScholarSearch**]. Jedoch muss auch hier eine entsprechende Schnittstelle vordefiniert werden und die Daten über das Internet an den Betreiber gesendet werden. Die Entwicklung und Integration dieser Schnittstellen fällt so auf den Entwickler zurück, welcher sich folglich weniger auf wichtige Features seiner Anwendung konzentrieren kann. Endnutzer müssen weiterhin darauf vertrauen, dass ihre Daten sicher und korrekt verarbeitet werden und das keine Informationen gespeichert oder missbraucht werden.

Das Ziel dieser Arbeit ist die Konzeption und Umsetzung von **AiXDroid**. Diese Anwendung ermöglicht es KI-Modelle direkt auf mobilen Geräten zu nutzen, ohne auf Internetverbindung und Cloud-Dienste angewiesen zu sein. Der

Endnutzer kann ein eigenes oder bereitgestelltes Modell auswählen, es auf sein Gerät laden und Prognosen oder Kategorisierungen für die eigenen Daten direkt auf dem Gerät erstellen lassen. Für Entwickler bietet die AiXDroid eine Schnittstelle, um Modelle zu integrieren, ohne sich um die Verwaltung von Daten und Modellen kümmern zu müssen. Die App ermöglicht es, Daten nahtlos zwischen verschiedenen Anwendungen auszutauschen. Der Nutzer behält dabei die volle Kontrolle, welche App auf seine Daten zugreifen darf. Nach der Freigabe einer App übernimmt AiXDroid automatisch die Verarbeitung und Bereitstellung der Daten, ohne dass weitere manuelle Eingriffe erforderlich sind.

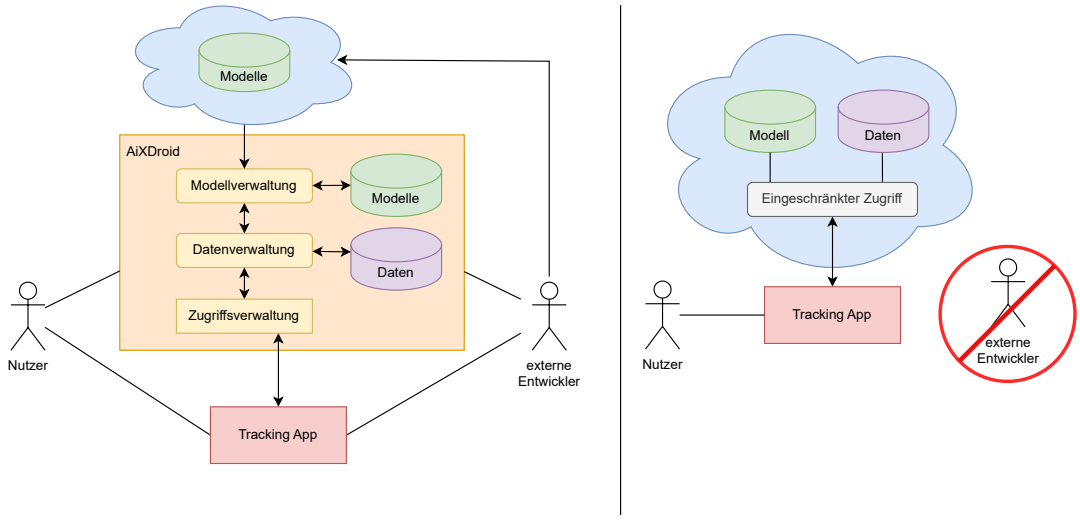

Abbildung 1.1.: Vergleich von AiXDroid und cloudbasierten Anwendungen

Durch die Eingeschränktheit der Daten und Modelle in der Cloud ist der Anwender auf die Zugriffsfreigaben des Anbieters angewiesen. Externen Entwicklern ist der Zugriff auf die Modelle in der Regel untersagt. Durch die lokale Verwendung der Modelle hat der Nutzer bei AiXDroid jederzeit vollen Zugriff auf seine Daten und was mit diesen geschieht. Es können neue Modelle von unabhängigen Entwicklern gefertigt und angeboten werden, sowie eine Zugriffsberechtigung vom Nutzer von AiXDroid zu anderen Apps eingerichtet werden.

1.1. Beispiel anhand der Schlafphasenanalyse

Für die Schlafphasenanalyse werden in der Medizin sogenannte Polysomnographen verwendet [10]. Diese Geräte besitzen eine Vielzahl von Sensoren, die über den gesamten Körper des Patienten verteilt werden. Es werden unter anderem Herzfrequenz, Atemfrequenz, Augenbewegungen und Hirnströme aufgezeichnet. Aus dem Verlauf dieser Werte kann nun eine geschulte Person die momentane Schlafphase bestimmen. Für die Messung der Atemfrequenz werden Schläuche mit Sensoren um die Brust gelegt. Diese messen die Ausdehnung des Brustkorbs und so jeden einzelnen Atemzug. Die Augenbewegungen werden über die elektrischen Impulse der Muskelbewegungen auf der Haut gemessen. Dazu werden Elektroden mit Kabeln an den Schläfen angelegt, die Spannungsänderungen aufzeichnen. Hirnströme werden ebenso mit Elektroden auf der Kopfhaut an spezifischen Positionen aufgenommen. Aufgrund der Komplexität und des Aufwandes dieser Messmethodik ist die Nutzung für den alltäglichen Gebrauch nicht geeignet. Die Aufzeichnung der Schlafphasen bietet im Alltag jedoch Vorteile. So kann eine Person besser nachvollziehen wie lang sie geschlafen hat, welche Schlafphasen sie die Nacht für wie lange hatte oder ob eine allgemeine Unruhe vorhanden war. Sie kann auch davon profitieren, dass zum Beispiel der Wecker erst in einer leichteren Schlafphase aktiviert wird. Die Wecker-App würde anhand der Schlafdaten prüfen, ob die person sich z.B. 30 Minuten vor der eigentlichen Weckzeit in einer leichten Schlafphase befindet und die Weckzeit dementsprechend nach vorne verlegen. Mittels Smartwatches lassen sich Daten, wie die Herzfrequenz, Bewegungsintensität oder auch Hauttemperatur während des Schlafes aufzeichnen. Diese Parameter werden auch bei Polysomnographen für die Schlafphasenanalyse verwendet. Über maschinelles Lernen kann ein Modell erstellt werden, dass anhand dieser Messwerte die momentane Schlafphase ermittelt [34]. Allerdings bringt die Aufzeichnung von Schlafdaten und das Verarbeiten dieser in KI-Modellen auch Herausforderungen mit sich, insbesondere im Hinblick auf den Schutz der Privatsphäre. Schlafgewohnheiten sind sehr persönliche Informationen, weshalb viele Nutzer zögern, entsprechende Apps zu verwenden. Durch die Verwendung eines Modells auf dem

lokalen Gerät entfällt die Datenübertragung und somit das Risiko des Datenmissbrauchs für den Anwender. In der Beispielanwendung wird ein Modell auf gelabelten Daten trainiert. Dieses Modell dient als Basis, um zu demonstrieren, wie KI-Modelle in AiXDroid eingebunden und genutzt werden können.

2. Grundlagen

In der Anwendung werden KI-Modelle zur Auswertung von Daten verwendet. Diese Modelle basieren auf dem Begriff des *maschinellen Lernen*. Die Beispielanwendung wird ein solches Modell zur Schlafphasenanalyse nutzen. Diese Analyse erfordert ebenfalls einige Begriffserklärungen. Die Grundlagen zum allgemeinen Verständnis werden im Folgenden näher erläutert.

2.1. Einführung in das maschinelle Lernen

Im Folgenden wird dargestellt, wie nach [22] das maschinelle Lernen verstanden wird. Unter dem Begriff des maschinellen Lernens (ML) versteht man einen Teilbereich der künstlichen Intelligenz, welcher Algorithmen beschreibt die durch Beispieldaten Regeln *erlernen* können. Diese erlernten Regeln können dann auf neuen, nicht bekannten Daten angewendet werden, um Vorhersagen, Empfehlungen und Entscheidungen zu generieren. Das System, dass aus dem Lernprozess hervorgeht nennt man auch *KI-Modell*. Das maschinelle Lernen ist besonders dann geeignet, wenn es zu komplex oder zeitaufwendig wäre, die zugrunde liegenden Regeln in den Daten manuell zu definieren. Stattdessen können Algorithmen diese Regeln eigenständig nähern. Eine vollständige Angleichung an die wahren Regeln ist mit diesen Verfahren jedoch nicht möglich. Es kann nur eine Annäherung gefunden werden.

2.1.1. Arten des maschinellen Lernens

Das maschinelle Lernen lässt sich in 3 Kategorien unterteilen. In Abbildung 2.1 werden die Unterschiede der verschiedenen Kategorien visuell dargestellt.

- **Überwachtes Lernen** (supervised learning) nutzt Daten, die bereits mit *Labeln* versehen sind, als Trainingsgrundlage. Ein Label stellt hierbei den gewünschten Ausgabewert bei den entsprechenden Eingabewerten dar. Ein Label, welches von einem Experten gesetzt wurde, wird als *Gold-Label* bezeichnet. Ein Modell lernt so, Muster zu erkennen und neue, unbekannte Daten auf Basis dieser Muster zu klassifizieren oder zu quantifizieren. Ein Beispiel hierfür wäre die Zuweisung von Schlafphasen zu Sensorwerten. Es kann aus der aktuellen Herzfrequenz, Hauttemperatur und Bewegungsintensität bestimmt werden, ob man sich im REM-Schlaf, Tiefschlaf, leichter Schlaf oder Wachzustand befindet. Was diese Labels genau aussagen wird in Unterabschnitt 2.2.1 genauer erklärt.

- **Unüberwachtes Lernen** (unsupervised learning) wird auf nicht gelabelten Daten angewendet. Das Modell identifiziert hier eigenständig Muster in den Daten, häufig zur Clusterbildung oder Dimensionsreduktion, z.B. bei der Segmentierung von Kundendaten oder zur Anomaliedetektion.

- **Verstärkendes Lernen** (reinforcement learning) ist eine Methode, bei der das Modell durch kontinuierliches Ausprobieren und Anpassen lernt, wie es bestimmte Ziele erreichen kann. Durch ein Belohnungssystem wird hier dem Modell mitgeteilt, wie gut eine Entscheidung gewesen ist. So kann das Modell seine Strategie nach und nach verbessern. Anwendung findet dieses Verfahren beispielsweise in der Steuerung autonomer Systeme oder bei der Spielstrategieentwicklung.

In dieser Arbeit wird ausschließlich das überwachte Lernen angewendet. Innerhalb des überwachten Lernens lassen sich zwei wichtige Arten von Problemen unterscheiden: *Klassifikation* und *Regression*. Diese beiden Problembereiche haben unterschiedliche Ziele und Metriken. Während die Klassifikation darauf abzielt, Daten in verschiedene Klassen oder Kategorien einzuteilen, geht es bei der Regression darum, kontinuierliche Werte vorherzusagen. Ein Beispiel für eine Regression wäre die Analyse von Aktienmärkten, wo die

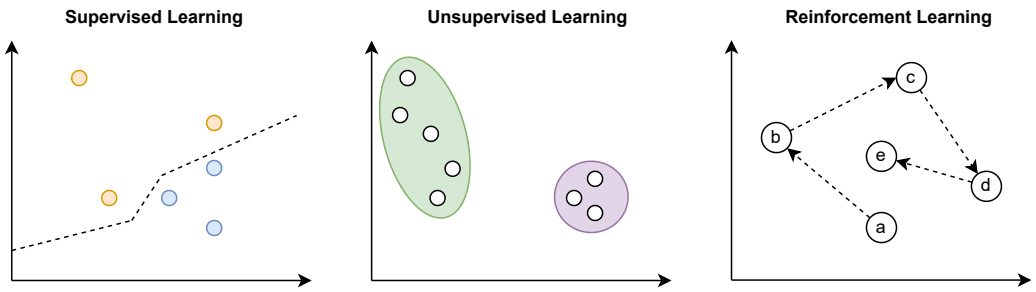

Abbildung 2.1.: Arten des maschinellen Lernens

Maschinelles Lernen ist in 3 Kategorien unterteilbar. Im überwachten Lernen sind gelabelte Daten (blaue und gelbe Punkte) vorgegeben und es wird eine Zielfunktion gesucht (gestrichelte Linie), die die Daten entsprechend separiert. Im unüberwachten Lernen sind die Daten nicht gelabelt. Es wird hier versucht Gruppierungen zu finden, in die möglichst viele Datenpunkte eingeordnet werden können. Beim verstärkenden Lernen wird eine Reihenfolge von Aktionen gesucht, die die gegebene Belohnungsfunktion maximiert.

Stärke des zukünftigen Anstiegs/ Abstiegs einer Aktie vorhergesagt werden soll.

2.1.2. Metriken

Aus [8] geht hervor, dass die Leistung eines Modells immer im Kontext seiner Art und des spezifischen Problems bewertet werden sollte.

Klassifikation

Bei Klassifikationsmodellen spielen drei Maße eine zentrale Rolle: *Genauigkeit* (Accuracy), *Präzision* (Precision) und *Sensitivität* (Recall). Jedes dieser Maße liefert wertvolle Informationen über die Modellleistung, jedoch kann keines allein eine vollständige Beurteilung der Modellqualität bieten. So berücksichtigt die Genauigkeit lediglich den Anteil der richtig klassifizierten Instanzen, ohne zwischen verschiedenen Fehlertypen zu differenzieren. Diese Fehlertypen können in zwei Hauptkategorien unterteilt werden: *Falsch-Positiv* (False Posi-

tive) und *Falsch-Negativ* (False Negative). Ein Falsch-Positiv liegt vor, wenn das Modell eine Instanz fälschlicherweise als positiv klassifiziert, obwohl sie tatsächlich negativ ist. Ein Falsch-Negativ hingegen liegt vor, wenn das Modell eine Instanz fälschlicherweise als negativ klassifiziert, obwohl sie tatsächlich positiv ist. Die Genauigkeit ist das Verhältnis zwischen den korrekt klassifizierten Instanzen (sowohl positiv als auch negativ) und der Gesamtzahl aller Instanzen. Errechnet wird die Genauigkeit durch Gleichung 2.1.

$$Accuracy = \frac{Number\ of\ correct\ Predictions}{Total\ Number\ of\ Predictions} \tag{2.1}$$

Die Präzision bezieht sich auf den Anteil der richtig klassifizierten positiven Instanzen an allen als positiv klassifizierten Instanzen und berücksichtigt somit die Falsch-Positiv-Rate. Eine hohe Präzision bedeutet, dass das Modell wenige Falsch-Positiv-Fehler macht. Sie wird durch Gleichung 2.2 errechnet.

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive} \tag{2.2}$$

Die Sensitivität hingegen bezieht sich auf den Anteil der richtig klassifizierten positiven Instanzen an allen tatsächlich positiven Instanzen und berücksichtigt somit die Falsch-Negativ-Rate. Eine hohe Sensitivität bedeutet, dass das Modell wenige Falsch-Negativ-Fehler macht. Durch Gleichung 2.3 lässt sich diese berechnen.

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative} \tag{2.3}$$

Der F-β-Score ermöglicht es, die Gewichtung von Präzision und Sensitivität anzupassen, je nachdem, welcher Aspekt wichtiger ist. Wenn $\beta > 1$, wird die Sensitivität stärker gewichtet, d.h. das Modell wird für seine Fähigkeit belohnt, positive Instanzen zu erkennen. Wenn $\beta < 1$, wird die Präzision stärker gewichtet, d.h. das Modell wird für seine Fähigkeit belohnt, korrekte positive Instanzen zu erkennen.

Die Wahl von β hängt von der spezifischen Anwendung ab. Wenn es wichtig ist, alle positive Instanzen zu erkennen (z.B. bei der Krebsdiagnose), kann ein höherer Wert von β gewählt werden (z.B. $\beta = 2$), um die Sensibilität zu

priorisieren. Wenn es hingegen wichtig ist, nur korrekte positive Instanzen zu erkennen (z.B. bei der Spam-Erkennung), kann ein niedrigerer Wert von β gewählt werden (z.B. β = 0,5), um die Präzision zu priorisieren. Mit Gleichung 2.4 lässt sich der F-β-Score berechnen.

$$F\text{-}\beta\text{-}Score = \frac{(1 + \beta^2) \times Precision \times Recall}{\beta^2 \times Precision + Recall} \tag{2.4}$$

Mit β = 1 ergibt sich der F1-Score , welcher den harmonischen Mittelwert von Präzision und Sensibilität angibt.

Regression

Bei der Regression können die Metriken der Klassifikation nicht mehr angewandt werden. Dadurch, dass das Modell immer nur eine Annäherung an die wirkliche Zielfunktion liefert wird der Wert, der aus dem Modell zurückgegeben wird, nahezu immer nicht exakt gleich dem Gold-Label sein. Daher wird hier auf statistische Verfahren zurückgegriffen. Der *mittlere quadratische Fehler* (MSE - Mean Squared Error) misst den durchschnittlichen quadratischen Fehler zwischen den wahren und den vorhergesagten Werten. Er wird durch Gleichung 2.5 bestimmt. Nachteil an dieser Berechnungsmethode ist, dass Ausreißer überproportional gewichtet werden.

$$\begin{aligned} MSE &= \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \\ n &= \text{Anzahl aller Prognosen} \\ y_i &= \text{i-ter Prognosewert} \\ \hat{y}_i &= \text{i-ter wahrer Wert} \end{aligned} \tag{2.5}$$

Über den *mittleren absoluten Fehler* (MAE - Mean Absolute Error) wird der durchschnittliche absolute Fehler zwischen den tatsächlichen und den vorhergesagten Werten bestimmt. Im Gegensatz zum MSE werden hier Ausreißer nicht stärker gewichtet als andere Fehler. Dies ist aber gleichzeitig auch ein Nachteil, da größere Abweichungen bei der Bewertung eben nicht mehr stärker ins Gewicht fallen. Er wird berechet durch Gleichung 2.6.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{2.6}$$

Über das *Bestimmtheitsmaß* (R^2 - R-squared) lässt sich die Güte der Vorhersage der Variation in den tatsächlichen Daten quantifizieren. Sie errechent sich aus Gleichung 2.7.

$$R^2 = 1 - \frac{MSE}{\frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2} \tag{2.7}$$
$$\tilde{y} = \text{Durchschnitt der wahren Werte.}$$

Auch hier gibt es keine beste Metrik für alle Fälle. Es kommt immer auf den Datensatz, die verwendeten Modelle und die zugrunde liegenden Bedingungen an. In der Regression kann die Wahl zwischen Metriken wie MSE, MAE oder R^2 von der Art der Fehlerverteilung und der Bedeutung von Ausreißern abhängen. Daher sollte die Wahl der Metrik immer unter Berücksichtigung der spezifischen Anforderungen und Ziele des Modells sowie der Zielgröße und Datenstruktur erfolgen.

2.1.3. Techniken des maschinellen Lernens

Neuronale Netzwerke spielen eine zentrale Rolle in der Welt des maschinellen Lernens. Sie umfassen verschiedene Modelle und Architekturen, die in Abhängigkeit von der Komplexität der Aufgabe und den verfügbaren Daten gewählt werden. Im Folgenden werden die grundlegenden Techniken von einfachen Perzeptrons bis hin zu komplexen Deep-Learning-Architekturen beschrieben. Für eine detailiertere Betrachtung der folgenden Thematiken kann man in [35] näheres finden.

Neuronales Netzwerk

Ein *Neuronales Netzwerk* (NN) besteht aus einer Eingabeschicht (*Input-Layer*), einer oder mehreren versteckten Schichten (*Hidden-Layers*) und einer Ausgabeschicht (*Output-Layer*). Jede dieser Schichten enthält Neuronen, die über gewichtete Verbindungen miteinander verbunden sind. Die Gewichte steuern

die Stärke der Signale zwischen den Neuronen und bestimmen, ob ein Neuron aktiviert wird. Im Trainingsprozess werden diese Gewichte durch die *Backpropagation* (Unterabschnitt 2.1.4) angepasst, um das Netzwerk zu optimieren. Neuronale Netzwerke sind vielseitig einsetzbar und können mit überwachter, unüberwachter oder verstärkender Methode verwendet werden.

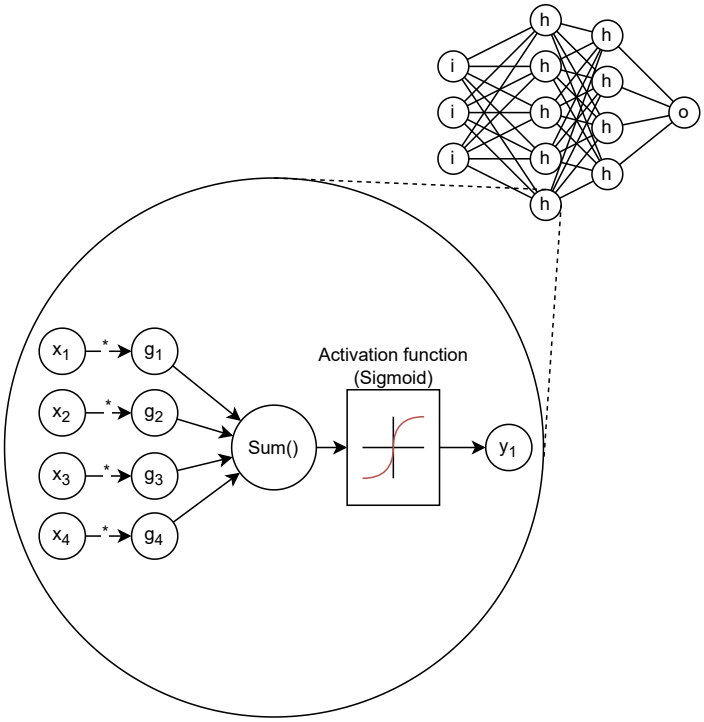

Abbildung 2.2.: Aufbau eines MLP

In der Abbildung ist ein Netzwerk mit 2 versteckten Schichten dargestellt. Ein mehrschichtiges Perzeptron ist, wie der Name es vermuten lässt, eine Anzahl an Schichten von einzelnen Perzeptrons, deren Output wieder als Input der nächsten Schicht dient. In der Abbildung sieht man die Funktionsweise eines einzelnen Neurons innerhalb der Schicht. Die Inputwerte werden mit den entsprechenden Gewichten multipliziert und die Summe aller Produkte wird an die Aktivierungsfunktion weitergegeben. Diese errechnet dann den Outputwert, der an die folgende Schicht weitergegeben wird.

Perzeptron

Das *Perzeptron* ist das einfachste Modell eines neuronalen Netzwerks. Es besteht aus einem einzelnen Neuron, das Eingabedaten auf Basis von gewichteten Summen und einer Aktivierungsfunktion verarbeitet. Es eignet sich jedoch nur für linear separierbare Probleme. Trotz seiner Einfachheit war das Perzeptron ein Meilenstein in der Entwicklung des maschinellen Lernens, da es den Grundstein legte für die Entwicklung der heute verwendeten Modelle.

Mehrschichtige Perzeptrons (MLP)

Nach [27] ist das *Mehrschichtige Perzeptron* (MLP) eine Erweitertung des Perzeptron um mehrere Schichten, was die Lösung von nicht-linear separierbaren Problemen ermöglicht. Jede Schicht besteht aus mehreren Neuronen, und die Daten durchlaufen alle Schichten, wobei sie bei jeder Schicht durch Aktivierungsfunktionen wie die Sigmoid- oder ReLU-Funktion transformiert werden (siehe Abbildung 2.2). Das MLP gilt als Grundform moderner neuronaler Netzwerke.

Deep Learning

Aus [21] geht hervor, dass unter *Deep Learning* neuronale Netzwerke mit einer großen Anzahl von Schichten zu verstehen sind, die es ermöglichen, komplexe und hierarchische Merkmale aus Daten zu extrahieren. Durch Architekturen wie *Convolutional Neural Networks* (CNNs) für Bildverarbeitung oder *Recurrent Neural Networks* (RNNs) für sequenzielle Daten können Deep-Learning-Modelle in einer Vielzahl von Domänen, wie der Bilderkennung (CV - Computer Vision) oder der natürlichen Sprachverarbeitung (NLP - Natural Language Processing) eingesetzt werden.

2.1.4. Training

Es müssen für den Lernprozess (Modelltraining) ausreichend Beispieldaten vorhanden sein, die zudem eine repräsentative Stichprobe darstellen. Eine

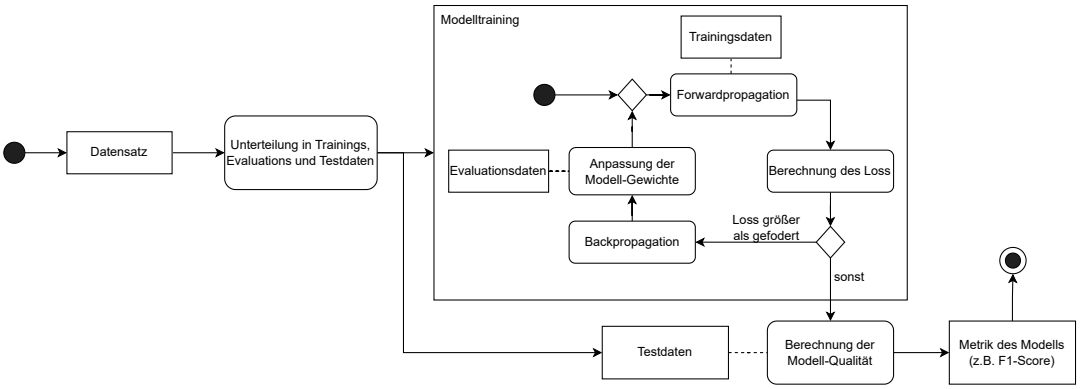

Abbildung 2.3.: Ablauf eines Trainings mit Aufteilung des Datensatzes

In der Abbildung wird eine vereinfachte Darstellung des Ablaufes beim Modelltraining gezeigt. Die Aufspaltung des Datensatzes in Trainings-, Evaluations- und Testdaten erfolgt noch vor dem Beginn des Trainings. Die Trainings- und Evaluationsdaten werden zur Optimierung der Modellparameter verwendet, während über die Testdaten nach dem Training die Modellqualität bestimmt wird.

ungleiche Verteilung innerhalb der Beispieldaten kann zu einer Verzerrung (Bias) führen, die die Ergebnisse des Modells beeinträchtigt. Dies ist auch ein entscheidender Kritikpunkt bei der Verwendung von KI-Modellen [20]. Über Metriken wird die Qualität eines KI-Modells evaluiert. Hierzu wird der Beispieldatensatz in 3 einzelne Datensätze unterteilt. Der Trainingsdatensatz stellt hierbei den größten Teil dar (60-80%). Auf diesen Daten wird das Modell trainiert. Es lernt die Regeln und Strukturen des gegebenen Problems anhand der Trainingsdaten. Um nun eine Aussage über die Qualität nach einem Trainingszyklus geben zu können wird der Evaluationsdatensatz verwendet (10-20%). Es werden die Prognosen des Modells mit den Werten im Evaluationsdatensatz verglichen und so ein Maß errechnet, welches die Qualität des Modells darstellt (siehe Unterabschnitt 2.1.2).

Während des Trainingsprozesses verarbeitet das Modell eine große Menge an Daten, um Muster und Zusammenhänge zu erkennen, die es zur Verbesserung seiner Vorhersagekraft benötigt (siehe Abbildung 2.3). Zur Bewertung des Fortschritts des Modells wird eine sogenannte Loss-Funktion ver-

wendet. Diese Funktion berechnet einen Fehlerwert, der angibt, wie gut oder schlecht das Modell im Vergleich zu den tatsächlichen Zielwerten abschneidet. Ein niedrigerer Wert der Loss-Funktion bedeutet, dass das Modell genauer in seinen Vorhersagen ist. Um das Modell zu verbessern, muss es den Fehlerwert „rückwärts“ verarbeiten, um seine internen Parameter anzupassen. Dieser Vorgang, bekannt als *Backpropagation*, ermöglicht es dem Modell, aus den Fehlern zu lernen und sich schrittweise zu verbessern. Backpropagation ist ein zentraler Bestandteil des Trainingsprozesses und wird durch den Gradientenabstieg und die Kettenregel der Differenzialrechnung optimiert. Der genaue Ablauf wird hier nicht dargestellt, kann aber in [24] nachgelesen werden. Das Training wird so lange wiederholt, bis das Modell die geforderte Güte erreicht hat. Um nun eine entgültige Qualitätsabschätzung zu bekommen, die nicht in den Trainingszyklus Einfluss genommen hat, wird der Testdatensatz verwendet. Auf diesen wird das Modell endgültig getestet und die Qualität des Modells bestimmt.

2.1.5. Zeitreihenanalyse

Die Zeitreihenanalyse ist ein spezieller Bereich der Datenanalyse, bei dem jedem Datenpunkt ein eindeutiger Zeitindex zugeordnet wird. Dies ermöglicht es, Veränderungen und Muster im zeitlichen Verlauf der Daten zu erkennen und auszuwerten. Der zeitliche Kontext ist dabei von zentraler Bedeutung, da die Beziehung zwischen aufeinanderfolgenden Datenpunkten essenziell für die Analyse ist. Ein Modell, das Zeitreihen verarbeitet, muss daher in der Lage sein, zeitliche Abhängigkeiten zu erkennen und zu nutzen. Zeitreihen zeichnen sich durch folgende Eigenschaften aus:

- **Zeitlicher Verlauf:** Die zeitliche Reihenfolge der Daten ist entscheidend, da vergangene Werte häufig zukünftige Entwicklungen beeinflussen.
- **Muster und Abhängigkeiten:** Trends (langfristige Entwicklungen), Saisonalitäten (wiederkehrende Muster) und Störungen (zufällige Schwankungen) sind typische Merkmale.

- **Kausalität:** Frühere Ereignisse können spätere Werte bedingen oder beeinflussen.

Die Verarbeitung von Zeitreihen stellt aufgrund ihrer spezifischen Anforderungen einen Spezialfall in der Datenanalyse dar und erfordert Modelle, die explizit auf die zeitliche Dimension abgestimmt sind. Um zeitliche Abhängigkeiten zu modellieren, werden spezialisierte neuronale Netzwerke wie *Long-Short-Term Memory*(LSTM)-Netze verwendet. Aus [18] geht hervor, dass diese besonders gut darin sind, Informationen über längere Zeitspannen hinweg „im Gedächtnis“ zu behalten. Ein weiterer Mechanismus wird in *rückgekoppelten neuronalen Netzen* (RNN - Recurrent Neural Network) verwendet. Hierbei werden Ausgabewerte als Eingabewert in späteren Zeitschritten verwendet, und so dem Modell ermöglicht frühere Zustände in die Berechnung mit einzubeziehen. Ein typisches Anwendungsbeispiel ist die Erkennung von Mustern in Finanzdaten, etwa bei Aktienkursen: Ein starker Anstieg eines Wertes in der Vergangenheit kann die Interpretation eines aktuellen, weniger starken Anstiegs beeinflussen. Dies erlaubt eine kontextabhängige Reaktion, wodurch Vorhersagen oder Entscheidungen verbessert werden können. Die Zielfunktion des Modells wird somit optimiert, da die zeitlichen Muster in den Daten berücksichtigt werden. in der Praxis kommen Zeitreihenanalysen in zahlreichen Bereichen zum Einsatz. Auch in der Medizin werden Zeitreihen von physiologischen Signalen wie Herzfrequenz oder EEG-Daten analysiert (siehe Abschnitt 2.2). In der Beispielanwendung wird ein solcher Ansatz verfolgt, um anhand der Sensorwerte einer Smartwatch die aktuelle Schlafphase des Trägers zu bestimmen (siehe Kapitel 5).

2.1.6. Anpassungen für den mobilen Einsatz

Nach dem Training eines Modells kann dieses eine hohe Leistungsfähigkeit erreichen. Allerdings geht dies häufig mit einer großen Modellgröße einher, da die zahlreichen Parameter als Zahlenwerte gespeichert werden müssen, was einen erheblichen Speicherbedarf bedeutet. Für den Einsatz auf mobilen Endgeräten, wie Smartphones, die nur begrenzten Speicherplatz zur Verfü-

gung stellen, stellt dies eine Herausforderung dar. Um das Modell für solche Anwendungsfälle anzupassen, ist es notwendig, dessen Speicherbedarf zu reduzieren, ohne dabei die Leistungsfähigkeit maßgeblich zu beeinträchtigen. Ein vielversprechender Ansatz zur Optimierung stellt der Prozess der *Quantisierung* dar. Laut [26] umfasst die Quantisierung verschiedene Techniken, um die Modellgröße zu verringern und gleichzeitig die ursprüngliche Leistungsfähigkeit möglichst zu bewahren. Eine besonders effektive, jedoch stark eingreifende Methode ist das sogenannte *Pruning*. Dabei werden gezielt oder zufällig Neuronen aus den Schichten des Modells entfernt, was zu einer Reduktion der Parameteranzahl führt. Eine weitere häufig eingesetzte Technik ist die Verringerung der Genauigkeit der Modellgewichte. Beispielsweise kann die Darstellung der Gewichte von der 32-Bit-Gleitkommadarstellung (FLOAT32) auf 16-Bit-Gleitkommadarstellung (FLOAT16) oder sogar auf Ganzzahlen (Integer) umgestellt werden. Dadurch wird die Speichergröße des Modells signifikant reduziert, während die grundlegende Struktur des neuronalen Netzes erhalten bleibt. Allerdings kann die Anwendung dieser Techniken leicht zu einem Verlust an Modellgenauigkeit führen. Um diesem Problem entgegenzuwirken, wird häufig ein Kalibrierungsdatensatz eingesetzt. Dieser wird verwendet, um das Modell vor der Quantisierung zu evaluieren und während des Quantisierungsprozesses die Gewichte so anzupassen, dass die Modellvorhersagen konsistent bleiben. Durch den Einsatz solcher Verfahren lässt sich die Größe eines Modells erheblich reduzieren, ohne einen signifikanten Einbruch in der Leistungsfähigkeit hinnehmen zu müssen. Dies ermöglicht den Einsatz hochentwickelter neuronaler Netze auch auf ressourcenbeschränkten Geräten wie Smartphones.

2.2. Schlafphasenanalyse

Die Schlafforschung spielt eine zentrale Rolle in der Untersuchung des Schlafes von Menschen und seiner biologischen Grundlagen. Nach [23] ist Schlaf ein essenzieller Bestandteil der menschlichen Gesundheit, der eine Vielzahl von physiologischen und psychischen Prozessen beeinflusst.

2.2.1. Schlafphasen

Nach [28] lässt sich der menschliche Schlaf in verschiedene Phasen unterteilen. Die Hauptphasen hierbei sind die folgenden:

- **Leichtschlaf** wird auch als Einschlafphase bezeichnet. Es ist der Momemt kurz vor dem Einschlafen. Diese Phase wird von der Person meist noch nicht als wirklicher Schlaf wahrgenommen.

- Im **Tiefschlaf** lässt sich eine Person nur noch schwer wecken. Das Bewusstsein ist nicht mehr aktiv und es werden Erinnerungen des Tages im Gehirn verarbeitet.

- Der **REM-Schlaf** beschreibt eine Phase des Schlafs, bei welcher eine hohe Augenbewegung einsetzt, daher der Name REM (Rapid-Eye-Movement - übersetzt „Schnelle Augenbewegungen“). In dieser Phase treten häufig Träume auf und es werden Informationen vom Kurzzeit- in das Langzeitgedächnis übertragen.

Im Wachzustand ist eine Person bei Bewusstsein und kann aktiv mit ihrer Umgebung interagieren. Der Wach-Zustand wird bei der Auswertung über ein KI-Modell ebenfalls als ein „Schlaf“-Zustand angenommen. Er tritt auf, wenn keiner der obigen Zustände den Daten entsprechend zugewiesen werden kann. Nach [28] ist das Durchlaufen aller Schlafphasen in einer Nacht, sowie die Dauer der einzelnen Phasen ein guter Indikator für die Qualität des Schlafes.

2.2.2. Stand der Technik

Es wird im folgenden auf die aktuellen Verfahren zum Schlaftracking eingegangen. Hierbei werden die verschiedenen Methoden in der Medizin und im Alltag näher beleuchtet.

Schlaftracking in der Medizin

Für die genaue Analyse und das Monitoring von Schlafphasen werden in der klinischen Forschung und Medizin oft polysomnographische Untersuchungen durchgeführt [28]. Diese Verfahren erfordern den Einsatz komplexer Geräte, die eine Vielzahl von physiologischen Parameter wie Gehirnströme, Augenbewegungen, Muskelaktivität und Herzfrequenz messen. Die Auswertung der Messergebnisse hin zu den Schlafphasen wird dann durch eine geschulte Person durchgeführt. Ein Laie würde mit den Messwerten hingegen nicht viel anfangen können. Darüber hinaus sind Polysomnographen teuer in der Anschaffung und aufwändig in der Verwendung, was den Einsatz für den persönlichen Gebrauch nahezu unmöglich macht. Technologische Fortschritte haben jedoch neue Möglichkeiten geschaffen, Schlafphasen kostengünstiger und einfacher zu überwachen. Da auch in der medizinischen Forschung ein Bedarf für die einfachere Schlafdatenerfassung besteht wurden sogenannte *Actigraphy*-Armbänder entwickelt [32]. Diese Geräte sind mit einer Reihe von Sensoren ausgestattet, die zur Messung der Bewegungen, Körpertemperatur und Herzfrequenz genutzt werden. Durch den Einsatz von Algorithmen, die diese Daten auswerten, ist es möglich, Schlafphasen zu identifizieren und zu analysieren. Die Algorithmen liefern Informationen darüber, wie lange eine Person in den verschiedenen Schlafphasen gewesen ist und geben so Hinweise auf die Qualität des Schlafes. Diese Geräte sind speziell auf die Bedürfnisse der Schlafforschung und klinischen Diagnostik abgestimmt und ermöglichen eine alternative, einfachere Erfassung der Schlafphasen.

Schlaftracking im Alltag

Die Sensorik, die in den Actigraphy-Armbändern verwendet wird, findet sich oft auch in Smartwatches wieder. Obwohl Smartwatches somit einen vielversprechenden Ansatz zur Schlafüberwachung bieten, gilt es auch hier Herausforderungen zu überwinden [34]. Die Genauigkeit der Analyse mittels Actigraphy oder Smartwatch ist weiterhin ein Thema der Forschung, da die Sensoren in diesen Geräten im Vergleich zu den aufwendigen polysomnographischen

Messmethoden nicht dieselbe Präzision bieten. Nichtsdestotrotz stellen diese tragbaren Geräte eine kostengünstigere und zugänglichere Möglichkeit für die breite Öffentlichkeit dar, ihren Schlaf zu überwachen und mögliche Schlafstörungen frühzeitig zu erkennen. Durch neuronale Netzwerke (siehe Unterunterabschnitt 2.1.3) lassen sich aus Sensordaten direkt die Schlafphasen ermitteln. Um ein solches Netzwerk darauf zu trainieren die entsprechenden Schlafphasen zu identifizieren benötigt es gelabelte Daten. Da verschiedene Smartwatches unterschiedliche Sensoriken nutzen und die Messwerte meistens auch noch durch eine interne Verarbeitung verändert werden, ist es notwendig ein Modell entsprechend auf Daten der entsprechenden Smartwatch zu trainieren. Das heißt, die gelabelten Daten müssen auch mit der verwendeten Smartwatch aufgezeichnet werden. Es gibt Smartwatches, die schon ein eingebautes Schlaftracking besitzen [29]. Wie das Schlaftracking genau implementiert wird geben die Hersteller jedoch nicht an. Wir können dennoch die Funktion zu unserem Vorteil nutzen, indem wir nur die Labels mit den zugewiesenen Uhrzeiten nehmen und sie an die Sensordaten einer anderen Uhr hängen, die zeitgleich getragen wurde. So kann ein neuer Datensatz auf Basis eines sogenannten „Silberstandards“ erstellt werden. Dieses Verfahren wird später beim Anwendungsbeispiel verwendet, um ein Modell für die Schlafphasenerkennung zu erstellen (siehe Kapitel 5). Es wird auch näher auf die aufgezeichneten Messwerte eingegangen, wie diese vorverarbeitet werden müssen und wie der Output des Modells interpretiert werden kann.

3. Softwaredesign und -konzept

Im folgenden wird nun das Design und Konzept der Anwendung AiXDroid erarbeitet, welche es ermöglicht mit Hilfe von lokal ausgeführten KI-Modellen Datensätze auf einem Android-Gerät zu verarbeiten. Es wird begonnen mit den grundlegenden Strukturen einer Android-Anwendung. Folgend wird eine Anforderungsanalyse erstellt, bei welcher die nicht-funktionalen und funktionalen Anforderungen an die Anwendung ausgearbeitet werden. Danach folgt die Auflistung der Anwendungsfälle, im folgenden *UseCases* genannt, die durch AiXDroid abgedeckt werden sollen.

3.1. Softwarearchitektur in Android-Apps

Die Anwendung wird speziell für das Betriebssystem *Android* erstellt. Android ist ein von Google entwickeltes Betriebssystem, das hauptsächlich auf mobilen Geräten verwendet wird [3]. Es basiert auf einem modifizierten Linux-Kernel und bietet eine offene Plattform für die Entwicklung von Apps [5]. Android ist bekannt für seine Flexibilität und unterstützt eine Vielzahl von Hardware-Komponenten und -Konfigurationen. Apps für Android können in Java, Kotlin oder C++ programmiert werden. Über die Android API kann auf verschiedene Funktionen des Geräts wie Kamera, GPS, Beschleunigungssensoren, usw. zugegriffen werden.

3.1.1. Android Runtime

Android verwendet ein Sandboxing-Modell, bei welchen jede App von anderen Apps sowie vom Betriebssystemkernel isoliert betrieben wird [4]. Jede

Anwendung wird in einer eigenen virtuellen Maschine betrieben. Das Framework, welches hierzu verwendet wird nennt sich *Android Runtime* (ART). So können Apps nur auf die Daten zugreifen, die ihnen zugewiesen wurden (siehe Abbildung 3.1). Für den Zugriff auf globale Ressourcen, wie dem Downloads-Verzeichnis oder privaten Ordnern muss eine Genehmigung durch den Anwender erteilt werden [11]. Verschiedene Arten von Berechtigungen geben dem Anwender dabei ein hohes Maß an Kontrolle über die Datenzugriffe der Anwendungen. So gibt es Berechtigungen, die nur für einen einzigen Zugriff Gültigkeit besitzen. Das bedeutet, der Anwender muss bei jedem weiteren Zugriff auf die Datei seine Zustimmung geben. Da dieses Vorgehen für viele Anwendungsfälle sehr hinderlich ist, hat jede App einen eigenen Speicherbereich. Hier können Dateien hingeschoben werden, die für die Anwendung jederzeit zur Verfügung stehen sollen.

3.1.2. Prinzipien der App-Entwicklung

Da mobile Geräte eingeschränkt sind in ihren verfügbaren Ressourcen kann das Betriebssystem jederzeit einzelne Komponenten von Apps beenden. Zustände müssen demnach außerhalb der Instanzen der Komponenten gespeichert werden. Die Komponenten müssen außerdem unabhängig voneinander gestartet werden können. Mit steigendem Umfang der Anwendung, steigt auch die Anforderung an eine solide Architektur um Stabilität, sowie einfache Skalierbarkeit und Testbarkeit zu ermöglichen. Hierzu lässt sich das Prinzip der „Separierung von Verantwortlichkeiten“ (Separation of Concerns) anwenden. Nach [17] besagt es, dass ein System in klar abgegrenzte Module oder Komponenten unterteilt werden sollte, wobei jede Komponente eine spezifische Aufgabe oder Verantwortung übernimmt. Die Benutzeroberfläche (UI), Domänenlogik und Datenverwaltung werden also in separaten Schichten organisiert. Dadurch wird der Code übersichtlicher und es wird verhindert, dass Änderungen in einem Bereich des Systems Beeinträchtigungen in anderen Bereichen verursachen.

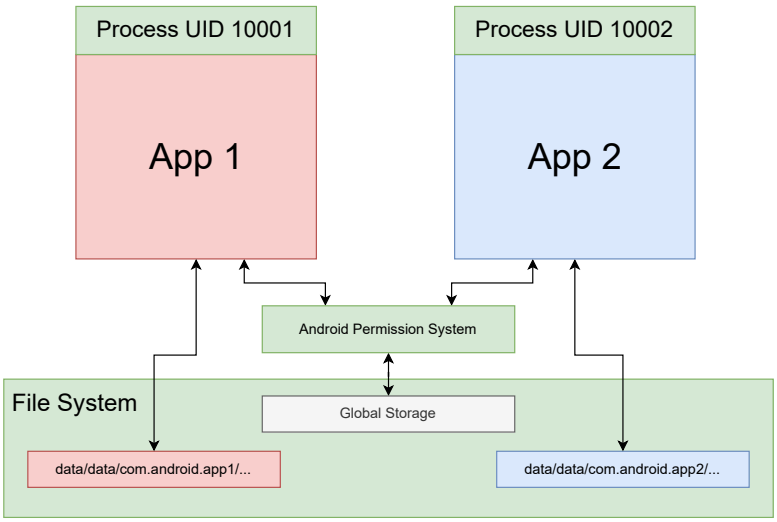

Abbildung 3.1.: Sandboxing von Android

Zwei Anwendungen laufen unabhängig voneinander auf einem Gerät. Sie können jederzeit auf den eigenen Speicherbereich zugreifen. Um jedoch auf einen gemeinsamen Speicher zugreifen zu können, wird eine aktive Zustimmung des Anwenders benötigt. Diese wird von Android verwaltet und kann, je nach Berechtigungsart, schon nach einmaligen Gebrauch wieder entzogen werden.

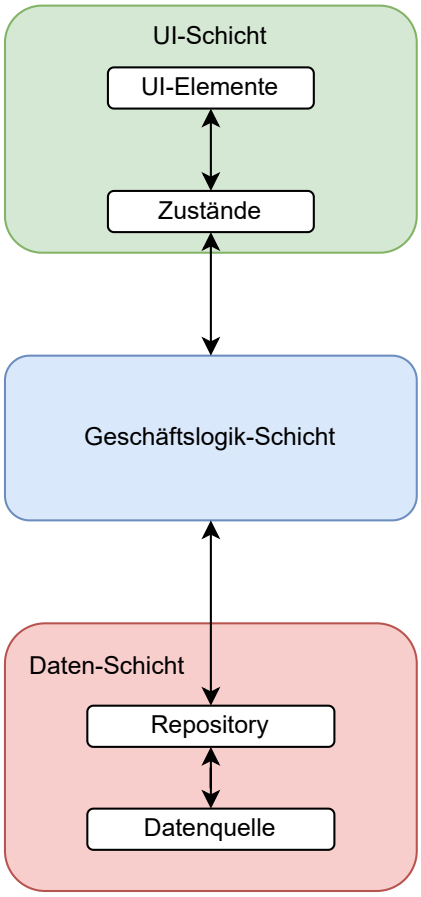

Abbildung 3.2.: Softwarearchitektur für Android-Apps

Die Separierung der Verantwortlichkeiten erhöht die Übersichtlichkeit des Programmcodes. Der unidirektionale Datenfluss ermöglicht eine konsistente Datenbereitstellung. Die zentrale Datenquelle wird über das Repository verwaltet.

3.1.3. Model View Viewmodel (MVVM)

Nach [6] ist *Model View Viewmodel* (MVVM) ein Architekturmodell, das auf dem bekannten *Model View Controller* (MVC) -Modell aufbaut. Es hilft dabei das Prinzip der Trennung von Verantwortlichkeiten durch eine klare Trennung zwischen Anwendungslogik und Benutzeroberfläche, zu beachten. Im folgenden werden die 3 Komponenten und ihre Aufgaben beschrieben.

Model

Das *Model* in der MVVM-Architektur enthält die Strukturen und Datenobjekte zusammen mit der Validierungslogik für diese Objekte. Diese Objekte stellen die Repositories, die UseCases, Datentranferobjekte (DTOs) und Entitätenmodelle dar.

View

Die *View* ist verantwortlich für die Struktur, das Layout und die Darstellung dessen, was der Anwender auf dem Bildschirm sieht. Sie stellt demnach die Benutzeroberfläche dar. Die View enthält keine Geschäftslogik. Das heißt, sie ist unabhängig von der Implementierung der eigentlichen Aufgaben der Anwendung. Über ein Viewmodel sendet die View Aktionen an die Geschäftslogik und erhält Daten und Zustände für ihre Anzeigeelemente.

Viewmodel

Das *Viewmodel* dient als Zwischenhändler von View und Model und steuert die Logik der View. Das Viewmodel interagiert mit dem Modell, indem es Methoden der Modellklassen aufruft. Es stellt dann die Daten aus dem Modell in einer Form bereit, die die View einfacher verwenden kann. Es ruft also Daten aus dem Modell ab und macht sie für die Benutzeroberfläche verfügbar, wobei es die Daten gegebenenfalls umformatiert, um sie handhabbarer zu machen. Zudem implementiert das Viewmodel Befehle, die in der View ausgelöst werden. Ein Beispiel dafür wäre, wenn ein Benutzer in der Benutzeroberfläche

auf einen Button klickt. Diese Aktion würde dann einen Befehl im Viewmodel auslösen , welcher wiederum Verarbeitungen in den Modellen auslöst. Das ViewModel kann für Zustandsänderungen verantwortlich sein, die ein Anzeigeelement in der Ansicht beeinflussen. So können zum Beispiel die Elemente einer Liste automatisch aktualisiert werden und vom Viewmodel der View bereitgestellt werden, ohne dass eine weitere Aktion der View notwendig wird.

3.1.4. Datenfluss und -haltung

Um die Daten, die der Nutzer über das UI in die App eingibt nicht durch mögliche Ressourcenfreigaben des Betriebssystems zu verlieren werden Zustände und Daten in Datenmodellen gestaltet. Datenmodelle repräsentieren die Daten einer App und sind unabhängig von den UI-Elementen und anderen Komponenten der App. Diese Modelle sind nicht an den Lebenszyklus des UI oder der Komponenten gebunden, werden aber dennoch zerstört, sobald das Betriebssystem den Prozess der App aus dem Speicher entfernt. Datenmodelle vereinfachen zusätzlich die Testbarkeit der Datenschicht.

Daten werden über eine zentrale Datenquelle (SSTO - „Single Source of Truth“) bereitgestellt [5]. Der Zugriff auf diese SSTO erfolgt in einem Unidirektionalen Datenfluss Muster (UDF). So gibt es zwei voneinander unabhängige Flüsse. Der Zustand fließt nur in Richtung der Datenquelle, während Ereignisse, die die Daten verändern, in die entgegengesetzte Richtung fließen (siehe UI-Schicht in Abbildung 3.2). Anwendungsdaten fließen von Datenquellen zum UI und Nutzeraktionen, wie das Drücken eines Buttons, wandern von der UI zur SSOT. Dort werden die Anwendungsdaten geändert und in einer unveränderlichen Form bereitgestellt. Durch Anwendung dieses Patterns wird die Konsistenz der Daten gesichert. Durch den definierten Fluss der Daten ist es außerdem einfacher Fehler zu entdecken.

In der Datenschicht wird das *Repository-Pattern* verwendet [14]. Ein *Code-Pattern*, oder kurz *Pattern* stellt eine Struktur für Code bereit, die eine leichte Erweiterbarkeit durch vorgegebene Abstraktionen ermöglicht. Das Repository-Pattern dient dazu die Datenzugriffsschicht klar von der Geschäftslogik zu trennen. Es beschreibt eine Struktur für den Vermittler zwischen den Da-

tenquellen (z.B. lokale Datenbanken oder entfernte APIs) und den übrigen Schichten der Anwendung (siehe Daten-Schicht in Abbildung 3.2). Durch eine einheitliche Schnittstelle für den Zugriff auf Daten werden Details der Datenquelle gekapselt. Dies ermöglicht es, die zugrunde liegende Datenquelle bei Bedarf auszutauschen, ohne die anderen Schichten der Anwendung anpassen zu müssen. Innerhalb des Repository-Patterns wird mit Interfaces gearbeitet, die konkrete Implementierungen verbergen. So können beispielsweise Netzwerkaufrufe oder Datenbankabfragen durch Mock-Daten ersetzt werden, was die Testbarkeit der Anwendung verbessert.

3.2. Anforderungsanalyse

Die Anforderungsanalyse dient als Grundstein für die Ausarbeitung, welche Voraussetzungen die Anwendung am Ende erfüllen soll. Sie wird unterteilt in nicht-funktionale und funktionale Anforderungen.

3.2.1. Funktionale Anforderungen und Anwendungsfälle

Die funktionalen Anforderungen spiegeln die Erwartungen der User an die Software wieder. Sie stellen mögliche Schnittstellen, Anwendungen und Nutzungsfälle dar.

- Wahl eines KI-Modells durch TFLite-Datei
- Lokale Verarbeitung von Daten mittels KI-Modell
- Lokale Datenspeicherung
- Übersichtliche Darstellung und einfache Bedienung
- Zugriff und Verarbeitung von Daten durch externe Anwendungen

Über die UseCase-Diagramme werden Abhängigkeiten zwischen den einzelnen Anwendungsfällen der beteiligten Akteure dargestellt. Hier werden die Schlüsselkomponenten der Software erstmals zusammengetragen und

genauer spezifiziert. Die Anwendungsfälle sind in 3 Kategorien unterteilt: KI-Modell Manager, Datenverwaltung und Zugriffsverwaltung. Im KI-Modell Manager kann der Anwender Modelle hinzufügen, die Metadaten der Modelldateien einsehen und die Modelle wieder entfernen (siehe Abbildung 3.3). Modelle die hier abgelegt werden sind für die Datenverwaltung und Zugriffsverwaltung zugänglich. In Tabelle 3.1 und Tabelle 3.2 sind die Beschreibungen der Anwendungsfälle „Modell hinzufügen“ und „Modelldetails einsehen“ dargestellt. Hier werden auch Details, die bei der Implementierung beachtet werden müssen genauer behandelt. Die Datenverwaltung gibt dem Anwender eine Schnittstelle zu den abgelegten Datenreihen und Datensätzen. Es können Datenreihen importiert, zu Datensätzen zusammengefasst und über Modelle verarbeitet werden (siehe Abbildung 3.4).Die Zugriffsverwaltung gibt dem Anwender die Möglichkeit anderen Apps Zugriff auf Daten zu gewähren. Es können Zugriffsrechte für Apps hinzugefügt, bearbeitet und entfernt werden (siehe Abbildung 3.5).

3.2.2. Nicht-Funktionale Anforderungen

Unter nicht-funktionalen Anforderungen versteht man Bedingungen an das System, welche nicht direkt mit der Funktion als solches verbunden sind.

Betriebssystem

Die Anwendung beschränkt sich auf das Betriebssystem Android ab dem API Level 31 bzw. der Android-Version 12.

Programmiersprache

Es wird die Programmiersprache Kotlin verwendet. Diese wird von Google seit dem Jahr 2019 als primäre Entwicklungssprache für Android-Anwendungen empfohlen [16].

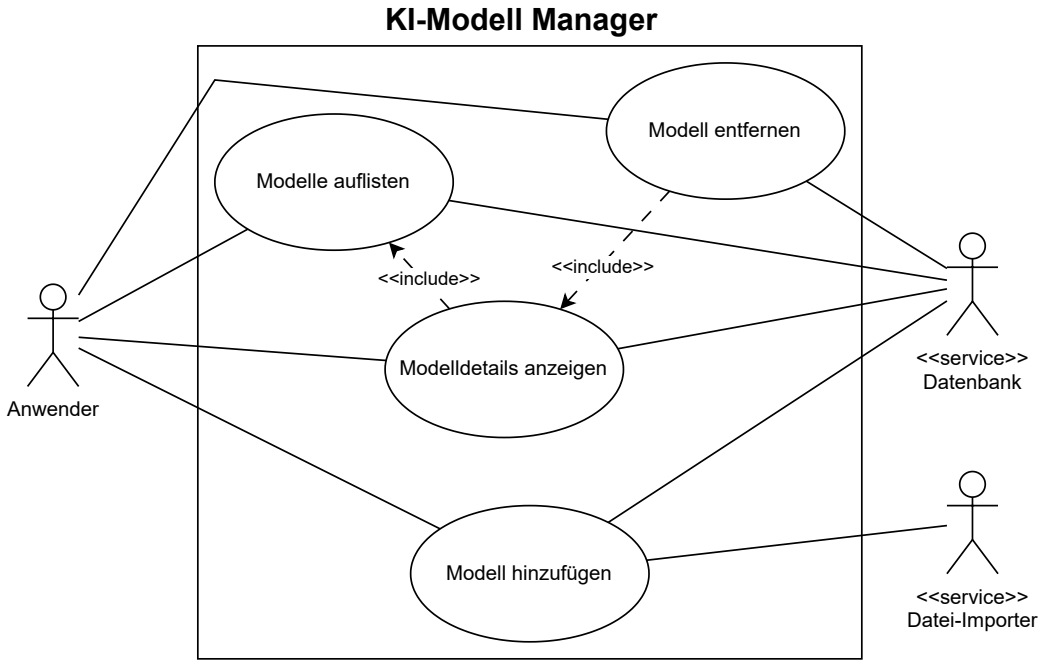

Abbildung 3.3.: UseCase-Diagramm des KI-Modell Managers

Im UseCase Diagramm für den KI-Modell Manager sind die Aktuere „Anwender“, „Datenbank“ und „Datei-Importer“ mit ihren UseCases angegeben. Die Datenbank und der Datei-Importer sind hierbei technische Akteure, die als Dienste in der Anwendung tätig sind.

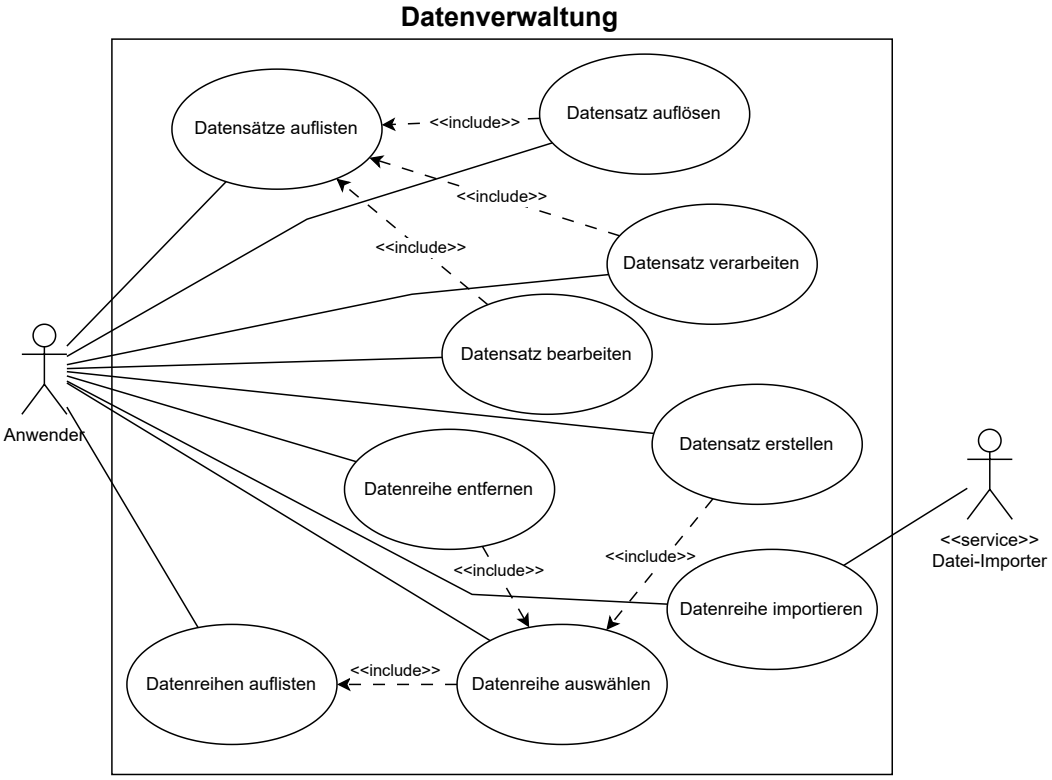

Abbildung 3.4.: UseCase-Diagramm der Datenverwaltung

In der Datenverwaltung kann der Anwender Datenreihen und Datensätze hinzufügen, bearbeiten und entfernen. Der Dienst-Akteur „Datenbank“ hat mir allen UseCases in diesem Bereich zutun, weshalb er als globaler Akteur angegeben ist. Über die include-Beziehungen werden Abläufe dargestellt. So muss eine Datenreihe beispielsweise erst ausgewählt werden bevor sie entfernt werden kann.

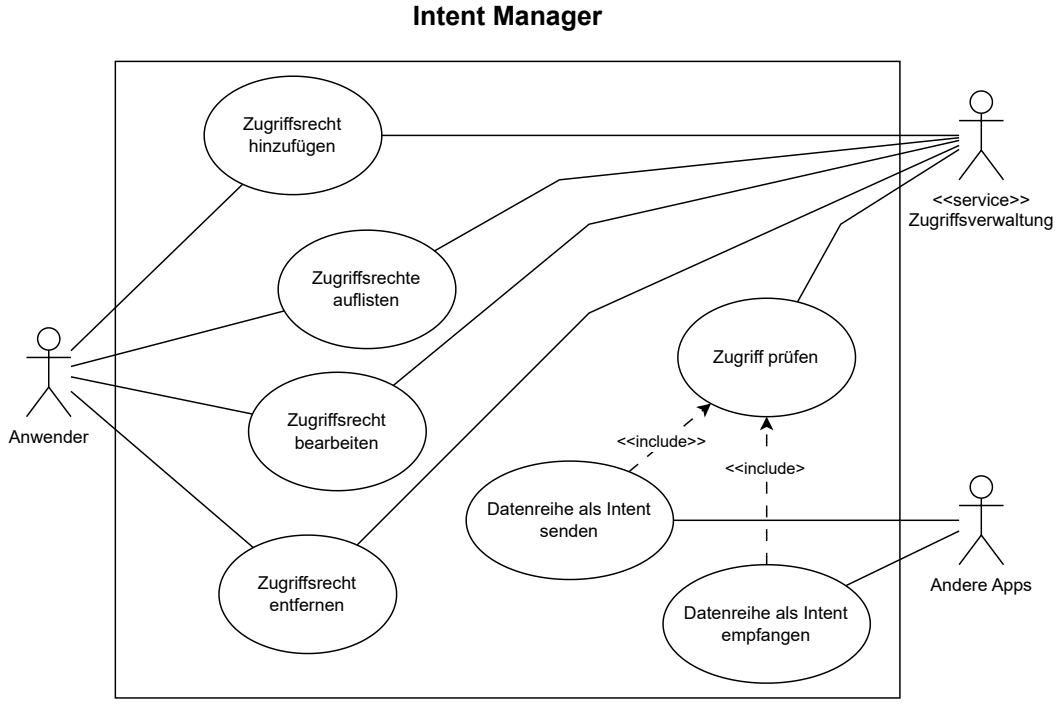

Abbildung 3.5.: UseCase-Diagramm des Intent Managers

Im Intent-Manager werden die Zugriffsrechte für andere Apps auf die von der Datenverwaltung bereitgestellten Daten eingerichtet und verarbeitet. Der Anwender hat dabei volle Kontrolle über die Zugriffsrechte, während die „Anderen Apps“ nur Daten senden oder empfangen können, wenn sie entsprechende Rechte bekommen haben.

Name	Modell hinzufügen
Beschreibung	Es wird ein Modell zur Verwendung hinzugefügt, so dass es in der Liste der Modelle zu finden ist.
Beteiligte Akteure	Anwender, System
Auslöser	UI-Element zum Hinzufügen eines neuen Modells wurde vom Anwender gedrückt
Pre-Conditions	keine
Normalablauf	1. Anwender drückt UI-Element zum Hinzufügen eines neuen Modells 2. Anwender wählt gültige TFLite-Datei aus 3. System liest TFLite-Datei ein und speichert Metadaten 4. Liste der Modelle wird aktualisiert
Alternativer Ablauf	1. Anwender drückt UI-Element zum Hinzufügen eines neuen Modells 2. Anwender wählt nicht-gültige Datei aus 3. System erkennt nicht gültige Datei 4. Dialog öffnet sich, der dem Anwender mitteilt, dass die Datei nicht gültig ist
Ergebnis	Ein neues Modell ist in der Anwendung verfügbar
Post-Conditions	keine
Hinweise	keine

Tabelle 3.1.: Use-Case: Modell hinzufügen

Name	Modelldetails einsehen
Beschreibung	Es werden die URI und die Metadaten des Modells dem Nutzer angezeigt.
Beteiligte Akteure	Anwender, System
Auslöser	UI-Element zum Anzeigen der Details eines gewählten Modells wurde gedrückt
Pre-Conditions	Mindestens ein Modell wurde hinzugefügt
Normalablauf	1. Anwender drückt UI-Element zum Öffnen der Modelldetails 2. Es werden die Modelldetails angezeigt
Alternativer Ablauf	keiner
Ergebnis	Es werden die Modelldetails angezeigt
Post-Conditions	keine
Hinweise	keine

Tabelle 3.2.: Use-Case: Modelldetails einsehen

Verfügbarkeit

Die lokale Ausführbarkeit sorgt für eine ständige Verfügbarkeit der Modelle. Der Anwender ist nicht auf eine Internetverbindung angewiesen und es müssen keine Server bereitgestellt werden.

Transparenz und Datenschutz

Durch die lokale Ablage und Verarbeitung der Daten ist der Datenschutz für den Anwender gewährleistet. Die Struktur des Android-Betriebssystems sorgt dafür, dass nur Apps mit entsprechender Berechtigung auf Netzwerkfunktionen zugreifen können. Da diese für meine Anwendung nicht notwendig sind, wird auch hier dem Anwender gesichert, dass seine Daten nicht an Dritte weitergegeben werden. Zusätzlich ist der Quellcode auf Github zur Verfügung gestellt [**<empty citation>**]. Somit kann der Anwender, bei entsprechenden

Kentnissstand den Quellcode einsehen, was zu einer vollständigen Transparenz führt.

4. Implementierung

In diesem Kapitel wird näher auf die Implementierung der besprochenen Strukturen zu einem Gesamtprogramm eingegangen. Zu Beginn werden die grundlegenden Entwicklungstools, wie Programmiersprache und Entwicklungsumgebung vorgestellt. Es folgt eine Darstellung die Systemarchitektur. Danach werden die Frameworks, welche bei der Umsetzung der Implementierungen genutzt werden genauer beleuchtet.

4.1. Entwicklungswerkzeuge

Im folgenden werden die im Projekt für die Implementierung verwendeten Tools und Frameworks vorgestellt.

4.1.1. Programmiersprache und Entwicklungsumgebung

Für Android lassen sich Anwendungen in C++, Java und Kotlin entwickeln. Java ist hier die gängigere Programmiersprache, da sie flexibler für unterschiedliche Geräte einsetzbar ist und es eine Vielzahl von verwendbaren Bibliotheken gibt. Seit dem Jahr 2019 wird von Google, dem Entwickler der Android-Plattform, die Sprache Kotlin als primäre Entwicklungssprache für Android-Anwendungen empfohlen [16]. Aus diesem Grund wurde diese Sprache für die Umsetzung des Projektes gewählt. Kotlin ist zwar eine eigene Programmiersprache, sie baut jedoch auf der Java Runtime Environment auf. Kotlin-Code lässt sich eins zu eins in Java Code umsetzen, wie auch Java Code zurück in Kotlin. Desweiteren können Java-Klassen direkt in das Kotlin-Programm eingebaut werden, was die Kompatibilität mit vorhandenem Code und Biblio-

theken erhöht. Als Entwicklungsumgebung wird Android-Studio verwendet. Durch die direkte Integration eines Android-Emulators und der Möglichkeit eine Vorschau der Benutzeroberfläche zu generieren ohne das Programm starten zu müssen, wurde sich für dieses Tool entschieden. Um Abhängigkeiten einfach zu verwalten wird das Tool *Gradle* verwendet. Dies ist ebenso in Android-Studio integriert. Über die

build.module.kts-Datei lassen sich verschiedene Bibliotheken in das Projekt einfügen. Die Datei selbst ist, wie auch der Rest des Projektes in Kotlin geschrieben.

4.1.2. Android Frameworks

Die *Android-API* vereinfacht es einem die Standardelemente des Androidsystems in der eigenen Anwendung zu implementieren. Eine der signifikantesten Komponenten hierbei stellt die *Aktivität* (Activity) dar. Aus [2] geht hervor, dass eine Activity einer Tätigkeit ist, die durch den Anwender gesteuert werden kann. Sie stellt ein Fenster für Eingabe- und Anzeigeelemente bereit.

Eine weitere, wichtige Klasse, die Verwendung findet ist *Viewmodel*. Es handelt sich hierbei um eine abstrakte Klasse, die Funktionen für die Implementierung des in der MVVM-Architektur vorgesehenen Viewmodels ermöglicht (siehe Unterabschnitt 3.1.3). Die konktrete Implementierung des Viewmodels dient zur Speicherung von Zuständen für die Benutzeroberfläche und als Schnittstelle zur Weiterreichung von Aktionen nach Benutzereingaben, wie dem Betätigen einer Schaltfläche und das Aufrufen des entsprechenden UseCases. Die Zustände können hierbei dynamisch im Viewmodel abgelegt sein, als sogenannte *StateFlow*-Objekte. Diese Objekte geben den aktuellsten Zustand des ihnen zugewiesenen *Flow*-Objekt wieder. Flow-Objekte sind sich dynamisch aktualisierende Objekte, welche von dem Datenbankmanagementsystem zurückgegeben werden können.

Das Datenbankmanagementsystem *Room* wird von Android bereitgestellt. Es erleichtert die Verwaltung einer SQLite-Datenbank. Einträge in der Datenbank werden durch *Data-Class-Objects* (DCOs) realisiert. Innerhalb der „@Entity“ Annotation, welche für Entitäten von Room gesetzt werden muss, können

dabei Fremdschlüsselbeziehungen, Primärschlüssel und andere Werte definiert werden. Über *Data-Access-Objects* (DAOs) werden die Zugriffe auf die Datenbank in einer Schnittstelle zusammengetragen. Es können dabei mehrere DAOs definiert werden, um, zum Beispiel unterschiedliche Domänen voneinander zu trennen.

Die Klasse *BroadcastReceiver* [12] wird als Basisklasse verwendet, wenn Intents von innerhalb oder außerhalb der App empfangen werden sollen. Es lassen sich BroadcastReceiver über die „AndroidManifest.xml“ definieren oder dynamisch registrieren.

Jetpack-Compose [30] stellt Komponenten bereit für die Erstellung einer zustandslosen Benutzeroberfläche. Das bedeutet, es werden keine Zustände innerhalb der Benutzeroberflächenlogik gespeichert. Die Speicherung erfolgt in dem entsprechenden Viewmodel. Bei der Verwendung von Jetpack-Compose wird nur eine einzige Activity implementiert. Sie dient als Einstiegspunkt und enthält die Grundlegende Fensterverwaltung für die gesamte Anwendung. Um verschiedene Tätigkeiten innerhalb der Anwendung voneinander getrennt zu modellieren werden *Composables* erstellt. Jedes Composable stellt hierbei eine Verschachtelung von anderen Composables dar. So kann eine Übersichtsseite eine Liste beinhalten, welches wiederum ein definiertes Listenelement aufruft, dieses dann ein Card-Composable und zulest ein Text-Composable (siehe **??**). So wird über die gesamte Anwendung das Design von einzelnen Elementen einfach wiederverwendbar gemacht. Dadurch, dass Composables ebenfalls in Kotlin definiert werden bleibt die Sprache über das gesamte Projekt hinweg gleich.

Um innerhalb der Composables auf die Viewmodels zugreifen zu können, muss dieses an das Composable als Parameter übergeben werden. Durch die vielen verschiedenen Composables könnte es hier leicht passieren, dass mehr als eine Instanz des Viewmodels erzeugt wird. Eine aufwendige Strukturierung wäre nötig, um dies zu vermeiden. Zur Zentralisierung der Instanzierung von Viewmodels und anderen Objekten wird das *Hilt*-Framework verwendet [1]. Hilt ermöglicht es, die Instanzierung von Viewmodels und anderen Abhängig-

keiten so zu gestalten, dass sie an den Lebenszyklus der jeweiligen Android-Komponenten gebunden sind.

Um ein Viewmodel mit Hilt zu verwenden, wird es zunächst mit der Annotation „@HiltViewModel“ versehen, um Hilt mitzuteilen, dass dieses Viewmodel von Hilt verwaltet werden soll. Anschließend kann es mit der Funktion „hiltViewModel“ direkt in einem Composable abgerufen werden, ohne es explizit als Parameter durch die Composable-Hierarchie zu reichen. Dies reduziert Boilerplate-Code und sorgt für eine saubere und übersichtliche Implementierung.

4.1.3. Tensorflow Framework

Für die Entwicklung einer Anwendung mit KI-Modellen wird eine Bibliothek zur Verwaltung von KI-Modellen benötigt. Es gibt hier verschiedene Frameworks die verwendet werden können. Diese Arbeit fokussiert sich auf die Verwendung von *Tensorflow* bzw. *Tensorflow Lite* [**<empty citation>**]. Tensorflow Lite basiert auf der größeren Bibliothek Tensorflow, welche Funktionen zum Erstellen von Modellen für das maschinelle Lernen in den Bereichen Desktop, Mobile, Web und Cloud bereitstellt. Entwickelt wurde es ursprünglich von Google zur Forschung im Bereich des maschinellen Lernens. Nun ist es ein Open-Source Projekt, welches frei zur Verfügung steht und bei dem jeder mitwirken kann. Tensorflow Lite ist eine speziell für mobile und IoT Geräte optimierte Version. Sie ermöglicht es größere Modelle in das TFLite-Format umzuwandeln. Hierzu wird unter anderem eine Technik namens Quantisierung angewendet, bei der die Modellparameter in ihrer Genaugigkeit reduziert werden, um so auch auf Systemen angewendet werden zu können, die nur über begrenzte Speicherkapazitäten verfügen.

liteRT

Die neuste Implementierung des Tensorflow-Lite Frameworks wurde in *liteRT* umbenannt [33]. Es wurden keine Änderungen an der API vorgenommen, die Funktionsaufrufe und Klassennamen sind gleich geblieben. Jediglich der Na-

me des Paketes in der Gradle-Implementierung hat sich geändert. Die Implementierung der liteRT Abhängigkeiten führte jedoch zu erheblichen Problemen bei der Erzeugung der *Interpreter*-Instanz, welche für das dynamische Laden von KI-Modellen benötigt wird. Die Anwendung stürzt ohne einen Fehler ab. Mittels Debugger konnte die Ursache auf einen Aufruf einer Funktion innerhalb des Frameworks zurückgeführt werden. Da es sich bei dieser jedoch um den Aufruf einer C-Funktion handelt konnte hier nicht weiter mit dem Debugger gearbeitet werden und auch keine Lösung für das Problem gefunden werden. Daher wird in dem Projekt weiterhin die Tensorflow-Lite Implementierung verwendet und auf eine zukünftige Implementierung verwiesen (siehe Abschnitt 6.1).

Versionsdowngrade

Die letzte Version von Tensorflow-Lite vor der Umbenennung in liteRT ist „2.17.0“. Da es hier aber zu Inkompabilitäten mit dem *Tensorflow-lite Metadata* Package gekommen ist, wurde die Version so weit reduziert, bis es zu keinen Problemen mehr mit dem Package gekommen ist. Das Tensorflow-lite Metadata Package dient dem Auslesen der Metainformationen innerhalb einer tflite-Datei. Es ist er spät aufgefallen, dass das Repo, in welchem dieses Package entwickelt wird seit über 2 Jahren keine Aktualisierung mehr bekommen hat. Auf eine alternative Herangehensweise, um nicht mehr auf dieses Package angewiesen zu sein wird in Abschnitt 6.1 eingegangen.

4.2. Systemarchitektur

Die Systemarchitektur beschreibt den allgemeinen Aufbau des Programms. Es wird auf die grundlegende Architektur eingegangen und die verschiedenen Schichten beschrieben, in welcher die Software aufgeteilt ist.

In Unterabschnitt 3.1.3 wurde auf die allgemein übliche Struktur von Android-Anwendungen näher eingegangen. Im folgenden wird nun die Implemen-

tierung der MVVM-Struktur in dem gegebenen Szenario beschrieben. Das Programm wird in 3 Schichten gegliedert:

- **Datenschicht** (Data-Layer)
- **Domänenlogikschicht** (Domain-Layer)
- **Benutzeroberflächenschicht** (UI-Layer)

4.2.1. Model

Die Aufteilung von Model, View und Viewmodel ist über die 3 Schichten verteilt implementiert. So gibt es in der Datenschicht die Datenbankmodelle, welche die Werte und Abhängigkeiten der Tabellen innerhalb der Datenbank beschreiben und zusätzlich in der Domänenlogik die Domänenmodelle, welche die Datenbankentitäten abstrahieren. Dies vereinfacht die Handhabung der Informationen aus der Datenbank. So werden zum Beispiel Datenreihen als Listen einem Datensatz-Objekt zugewiesen. Beide Elemente lassen sich der Model-Komponente der MVVM-Architektur zuweisen, befinden sich jedoch in unterschiedlichen Schichten. Für einen übersichtlichen Übergang von den Datenmodellen zu den Domänenmodellen wird ein Repository-Pattern angewendet [7]. In diesem wird eine Schnittstelle für den Datenzugriff bereitgestellt, welcher für alle UseCases gleicht bleibt. Dies erleichtert zusätzlich die Erweiterung der Anwendung durch andere Datenhaltungsmethoden, wie zum Beispiel einer alternativen Datenbankstruktur (siehe Abschnitt 6.1).

4.2.2. View

Die View, welche das Bild auf den Bildschirm bringt ist mittels *Jetpack-Compose* definiert. Dieses Framework basiert ebenfalls auf der Sprache Kotlin und sorgt für eine dynamische und flexible Erstellung der Benutzeroberfläche. Elemente in diesem Framework werden als *Composables* annotiert. Dadurch, dass die Benutzeroberfläche zustandslos (stateless) bleibt, kommt es bei dem Wechsel

von einer Aktivität zur Nächsten zu keinem Datenverlust. Die Viewmodels, welche in den entsprechenden Views verwendet werden, werden zu Beginn vom oberen Composable geladen und an die darunterliegenden weitergereicht.

4.2.3. Viewmodel

Die Viewmodels befinden sich in der Benutzeroberflächenschicht und dienen dem Halten der aktuellen Werte zur Darstellung des Displays, sowie dem Aufrufen von UseCases. Auch hier wurden wieder 3 Klassen erstellt, welche für die entsprechenden Umgebungen zuständig sind. Sie kommunizieren über die UseCases der Domänenlogikschicht mit den Repositories. Dadurch werden der Zugriff auf Daten und die Datenverarbeitung allein in der Geschäftslogik behandelt.

4.3. Datenschicht

Die Datenschicht beinhaltet alle Strukturen und Logiken zur Haltung und Verwaltung der Daten. Dies beinhaltet die Entitäten-Klassen für das Room-Framework zur Erstellung der Datenbank, die Definitionen der Repositories für den Zugriff auf die Datenbank sowie die Umwandlung der Datenbank-Modelle in die einfacher handhabbaren Domain-Modelle.

4.3.1. Data Access Object

Der Zugriff auf die Datenbank-Modelle erfolgt über *Data Access Objects* (DAOs). Diese verknüpfen Methoden mit direkten SQL-Anweisungen. Die DAOs werden nur in den Repository-Implementierungen verwendet, um den Zugriff auf die Datenbank zentralisiert zu halten. So werden Datenbank-Modelle auch nicht an die Domänenlogik weitergereicht, sondern nur ihre entsprechenden Domain-Modelle. Die Umwandlung der Domain-Modelle zurück in die Datenbank-Modelle wird ebenso von den Repository-Implementierungen übernommen.

Es gibt 3 explizite DAOs, wobei jedes für einen eigenen Abschnitt in der Anwendung zuständig ist. Das *DataSetDao* verwaltet die Datenpunkte, Datenreihen und Datensätze. Das *ModelDataDao* ist für die Verwaltung der KI-Modelle zuständig. Das *IntentDataDao* speichert Profile für externe Anwendungen, die über den External-Intent-Service (Unterabschnitt 4.7.1) auf Daten zugreifen dürfen. Ebenso gibt es 3 Repository-Definitionen mit je einer Implementierung (DataRepository, ExternalIntentRepository und ModelRepository). Die genaue Struktur der Datenbank lässt sich in **??** nachsehen.

4.4. Domänenlogikschicht

In der Domänenlogikschicht werden die Zugriffe auf Daten sowie deren Verarbeitung vorgenommen. Sie beinhaltet alle UseCases, die die Anwendung unterstützt. Desweiteren befinden sich hier auch die *Domänenmodelle*. Diese Modelle sind, wie in der Datenschicht bereits erwähnt, vereinfachte Strukturen für die Verwaltung und Verarbeitung der Daten aus der Datenbank. So werden beispielsweise Datensätze mit ihren entsprechenden Datenreihen in einem Objekt zusammengefasst. Die Datenpunkte werden nicht in den Datenreihen hinterlegt, da diese die Größe der Datenreihen-Objekte erheblich erhöhen könnte. Desweiteren wird zur Verwaltung der Datenreihen meist nur die Meta-Information der entsprechenden Reihe benötigt. Der Zugriff auf die Datenpunkte erfolgt über einen separaten UseCase.

Die UseCases selbst sind, wie auch schon die Repositories und DAOs in 3 Oberkategorien gegliedert: *Model Manager*, *Data Manager* und *Intent Manager*. Innerhalb der Oberkategorien werden mehrere UseCases, die auf die selben Repositories angewiesen sind und im selben Bereich arbeiten in einer Klasse zusammengefasst. So gibt es, zum Beispiel die Klasse *DataSeriesUseCases*, welche alle für die Datenreihen erforderlichen Verwaltungsaufgaben erfüllt, wie das Einfügen einer neuen Datenreihe, die Bearbeitung einer vorhandenen Reihe oder das Hinzufügen von Datenpunkten zu einer Datenreihe. Die einzelnen UseCase-Klassen sind hierbei durch Interfaces definiert, wodurch das Testen der UseCases vereinfacht wird.

Es werden im Folgenden die Implementierungen der UsesCases für den Dateiimport *DataImportUseCase* und für die Modellprognose *InferenceUseCase* vorgestellt.

4.4.1. DataImport UseCase

Um Daten auch ohne eine andere App in die Anwendung einzugeben können CSV-Dateien importiert werden. Bei einer CSV-Datei ist in Klartext verfasst. Die Spalten werden durch Komma („,“) voneinander getrennt und die Zeilen durch Zeilenumbrüche. Es wird eine festgelegte Syntax erwartet, um zwischen Zeitindex und Datennamen zu unterscheiden. Der Zeitindex steht in der ersten Spalte und besitzt in der ersten Zeile den Text „Time“. Als Format wird ein Unix-Timestamp in Millisekunden erwartet. Es folgen beliebig viele Datenreihen im gleichen Schema. Die erste Zeile benennt hierbei den Namen der zu erstellenen Datenreihe. Als Format werden Float-Werte mit Punkt-Notation erwartet. Jeder Datenpunkt muss einem Zeitpunkt zugewiesen sein. Es können Zeitintervalle ausgelassen werden und auch die Reihenfolge ist nicht vorgegeben. Jedoch sollte für einen Zeitpunkt nur genau ein Datenwert pro Datenreihe angegeben werden. Andernfalls wird der zuletzt angegebene Wert übernommen. Die Datei kann an einem beliebigen Ort auf dem Gerät liegen (zum Beispiel dem Download-Verzeichnis). Der Import wird über die Schaltfläche mit dem Pfeil nach oben Abbildung 4.1 ausgewählt. Er startet sobald im Datei-Manager eine .CSV-Datei ausgewählt wurde. Über eine Anzeige am oberen Bildschirmrand kann nun der Fortschritt des Import-Prozesses gesehen oder der Import-Vorgang abgebrochen werden. Wenn der Vorgang abgebrochen wird, verbleiben die bisher importierten Daten weiterhin in der Datenbank.

4.4.2. Inference UseCase

Für die Durchführung einer Inferenz auf Basis eines Datensatzes werden die Details des Datensatzes abgerufen. Anschließend werden die Features in eine Tensor Map überführt, um eine Datenstruktur zu erstellen, die die Datenpunkte ihren Zeitpunkten zuordnet. Daraus wird ein Array gebildet, das die Zeit-

Abbildung 4.1.: Screenshot des Data-Managers

Es ist der Data-Manager mit einem erstellten Datensatz und einer nicht zugewiesenen Datenreihe zu sehen. Über die Pfeil-Schaltfläche lassen sich neue Daten aus CSV-Dateien hinzufügen. Im oberen Bereich ist zu sehen, dass ein Dateiimport im Gange ist. Dieser lässt sich über die Cancel-Schaltfläche abbrechen.

punkte ebenfalls als Feature enthält. Bevor die Zeitpunkte integriert werden, erfolgt eine Konvertierung durch Sinus- und Cosinus-Funktionen, basierend auf der vom Modell vorgegebenen Zeitperiode.

Da nun alle Features als Float-Werte vorliegen, können sie in ein Array von ByteBuffer-Objekten umgewandelt werden, welche für die Verwendung des Interpreters erforderlich sind. Die Ergebnisse der Inferenz werden in einen weiteren ByteBuffer geladen und anschließend als Float-Werte ausgelesen.

Durch die Klassifikation mit *SparseCategoricalCrossentropy* gibt das Modell entsprechend der Anzahl der möglichen Klassen viele Float-Werte zurück. Diese Werte stellen sogenannte *Pseudowahrscheinlichkeiten* dar. Die Klasse mit dem höchsten Wert wird als vorhergesagte Klasse identifiziert. Pseudowahrscheinlichkeiten stellen jedoch keine tatsächlichen Wahrscheinlichkeiten dar [25]. Daher kann an ihnen nicht die Qualität der Erkennung abgelesen werden.

4.5. Benutzeroberflächenschicht

Die Benutzeroberfläche stellt die Schnittstelle zwischen Benutzer und Anwendung dar. Sie ist so gestaltet, dass man leicht zwischen den 3 Hauptverwaltungen wechseln kann und so einen schnellen Überblick über den momentanen Stand der Daten sowie der laufende Verarbeitung hat. Sie besteht aus dem *Model Manager*, dem *Data Manager* und dem *Intent Manager*.

4.5.1. Model Manager

Über den *Model Manager* lassen sich die KI-Modelle, die zur Verarbeitung der Daten genutzt werden, verwalten. Es werden in einer Liste alle Modelle aufgeführt, die über einen Klick im Detail betrachtet werden können. Ein Modell wird hinzugefügt, indem der Plus-Knopf in der Übersicht gedrückt wird. Anschließend muss eine TFLite-Datei ausgewählt werden, welche über Metainformationen verfügt. Die Datei wird in den lokalen Anwendungsspeicher kopiert und kann nach dem Hinzufügen entfernt werden. In der Detailansicht werden die Metainformationen des Modells und der Datei-Name der abgespeicherten Modelldatei angezeigt. Das Modell kann über den roten Knopf am unteren Bildschirmende gelöscht werden.

4.5.2. Data Manager

Der *Data Manager* gibt einem die Möglichkeit eigene Daten zu importieren und vorhandene Daten zu verwalten. Über das Plus-Symbol lassen sich Dateien im CSV-Format importieren Unterabschnitt 4.4.1. Diese werden dann als Datenreihen in der Datenbank abgelegt. Die zu keinem Datensatz zugewiesenen Datenreihen werden in dem obersten Eintrag angezeigt (siehe Abbildung 4.1). Durch Drücken auf das oberste Element gelangt man zu der Liste der Datenreihen. Hier kann man nun, durch langes Drücken der Elemente, Datenreihen auswählen und zu einem Datensatz zusammenfügen. Für die Erstellung eines Datensatzes muss ein Name und eine Beschreibung angegeben werden. Es öffnet sich die Detailansicht des gerade erstellten Datensatzes mit den Informationen über Name, Beschreibung, Anzahl an Datenpunkten, frühester Datenzeitpunkt und spätester Datenzeitpunkt im Datensatz (siehe **??**). Über die Schlaltfläche „configure Inference“ lässt sich ein KI-Modell dem Datensatz zuweisen. Zuerst werden alle verfügbaren und kompatiblen KI-Modelle angezeigt, aus denen man auswählen kann. Danach folgt die Zuweisung der Datenreihen zu den entsprechenden Input-Tensoren des Modells. Abschließend wird der Name und die Einheit der erzeugten Datenreihe angegeben und die Konfiguration gespeichert. Wenn das Feld „Auto-Inference“ aktiviert

wurde, wird die Inference automatisch neu gestartet, sobald neue Daten den betreffenden Datenreihen hinzugefügt wurden. Sobald die Konfiguration abgeschlossen wurde wird unabhängig von der Einstellung der Auto-Inference eine Inference gestartet. Dies ist an einem Ladebalken am oberen Bildschirmrand erkennbar, welcher den momentanen Fortschritt der Inference anzeigt. Wenn gewünscht lässt sich dieser Vorgang Über die Schaltfläche „Cancel“ abbrechen. Ein Datensatz kann auch auf Basis von berechneten Datenreihen erstellt werden. So lassen sich beispielsweise mehrere Modelle, die auf den Ergebnissen der vorherigen Modellen beruhen hintereinander schalten.

4.5.3. Intent Manager

Der *Intent Manager* dient der Verwaltung von Profilen für externe Anwendungen, die Zugriff auf die AiXDroid-Datenbank erhalten sollen. Im Rahmen der Profilerstellung können Anwendungen benannt und Berechtigungen zugewiesen werden. Nach der Anlage eines Profils ist es der externen Anwendung möglich, einen Intent zu senden, der entweder Daten zur Eingabe in die Datenbank oder eine Anfrage zum Abrufen bestimmter Daten aus der Datenbank enthält. Sofern die erforderlichen Zugriffsrechte vorliegen, werden die angeforderten Daten als Intent an die externe Anwendung zurück übermittelt. Das Empfangen von externen Intents wird von einem im Hintergrund laufenden Service gesteuert (siehe Unterabschnitt 4.7.1).

4.6. Dependency Injection

Es gibt viele verschiedenen Abhängigkeiten innerhalb des Programms, zum Beispiel der UseCases von Repositorys oder der Viewmodels von den UseCases. Um hier einen übersichtlichen Zugriff zu erhalten wird das Prinzip der *Dependency Injection* verwendet [13]. Hierbei werden die Klassen, welche Abhängigkeiten besitzen mit *@Inject* annotiert. Das Framework Hilt [1] kann nun durch vordefinierte Module diese Abhängigkeiten füllen, ohne das mehrere Instanzen der selben Klasse erstellt werden müssen oder manuell Referen-

zen im Code übergeben werden müssen. Die Module, welche die Instanzen der Abhängigkeiten bereitstellen (Provider) befinden sich in einem eigenen Abschnitt. Jedes Module beinhaltet dabei wieder die für einen bestimmten Bereich zuständigen Provider, wie zum Beispiel das *DatabaseModule* die eigentliche Datenbank-Instanz bereitstellt, sowie die Instanzen der DAOs.

4.7. Hintergrunddienste

Für die Verarbeitung der Daten, ohne dass der Anwender die App ständig geöffnet haben muss, werden *Hintergrunddienste* (Services) [31] verwendet. In Android gibt es drei verschiedene Arten von Services: Foreground-Service, Background-Service und Bound-Service.

Der *Foreground*-Service beschreibt eine operation, welche für den Anwender sichtbar ist, zum Beispiel die Steuerung einer Musik-App während dem Abspielen. Für einen Foreground-Service muss eine Notification im Display sichtbar sein, die dem Anwender zeigt, dass der Prozess gerade läuft und ihm Kontrolle über diesen gibt.

Der *Background*-Service ist für Vorgänge, die für den Anwender nicht direkt sichtbar sind. So könnte ein solcher Service eine Komprimierung von Daten im App-eigenen Datenspeicher vornehmen. Ein wichtiger Unterschied zum Foreground-Service ist hier jedoch, dass dieser Service jederzeit durch das Betriebssystem unterbrochen werden kann, um Ressourcen einzusparen. Außerdem dürfen diese Anwendungen bestimmte Aufgaben, wie das Verarbeiten von BroadcastReceivern nicht durchführen.

Zuletzt gibt es noch den *Bound*-Service, welcher für eine Client-Server Verbindung zwischen Komponenten sorgt. Es können Anfragen gesendet und Ergebnisse empfangen werden, auch über den eigenen Prozess hinaus. Solange eine Verbindung besteht, wird auch der Service weiterhin laufen. Es können mehrere Komponenten an einen Service zugleich gebunden werden. Sobald keine Komponenten mehr verbunden sind, wird auch der Service beendet.

4.7.1. External Intent Service

Die AiXDroid-Anwendung startet zu Beginn den Service *ExternalIntentService*. Dieser geht die Liste der Profile im Intent-Manager durch, sobald eine Änderung in der Datenbank-Tabelle stattfindet. Es wird für jedes Profil ein *ExternalIntentReceiver* Objekt erstellt und registriert. Diese Klasse implementiert den von Android bereitgestellten *BroadcastReceiver*. Der BroadcastReceiver empfängt ankommende Intents und stellt eine Schnittstelle für die Verarbeitung dieser bereit. Die ExternalIntentReceiver erweitern die Funktion des BroadcastReceivers um die angegebene *IntentAction* in Form eines *Strategy*-Patterns. IntentAction ist ein Interface, um jede Aktion eines entsprechenden Intents zu abstrahieren. Die zugewiesene IntentAction-Implementierung enthält die Funktionen zur Verarbeitung des empfangenen Intents. Sollte sich die Konfiguration der Profile geändert haben, werden die laufenden Receiver abgemeldet, angepasst und wieder angemeldet. Über UseCases werden die Aktionen der IntentAction-Implementierungen abgebildet, um doppelten Code zu verhindern. Die Nutzung des Strategy-Patterns an dieser Stelle vereinfacht auch die Ergänzung neuer IntentAction-Implementierungen.

Die folgenden Implementierungen des IntentAction-Interface sind erstellt worden.

ReadDataExternalIntentAction

Die ReadDataExternalIntentAction erwartet in den Extras des gesendeten Intents ein Bundle-Objekt mit folgenden Parametern:

- „seriesName“: Der Name der zu lesenden Datenreihe im String-Format.
- „lastTimeValue“: Der maximal zulässige Zeitstempel (im Long-Format) der auszulesenden Daten.
- „dataCount“: Die Anzahl der zurückzugebenden Daten im Integer-Format.

WriteDataExternalIntentAction

Die WriteDataExternalIntentAction erwartet ein Bundle-Objekt mit den folgenden drei Einträgen:

- „seriesName“: Der Name der zu schreibenden Datenreihe. Sollte diese noch nicht existieren, wird sie neu angelegt.
- „timeValueArray“: Ein Array von Zeitpunkten im Long-Format, die in die Datenreihe aufgenommen werden sollen.
- „dataValueArray“: Ein Array von Float-Werten, die den Zeitpunkten aus timeValueArray zugeordnet werden.

Die Werte im „dataValueArray“ werden nach ihrer Reihenfolge den Zeitstempeln im „timeValueArray“ zugeordnet. Existiert die Datenreihe bereits und sind die angegebenen Zeitstempel vorhanden, so überschreiben die neuen Werte die bisherigen.

4.8. GadgetBridge Implementierung

Bei der *GadgetBridge* handelt es sich um eine Open-Source Software zur Verwaltung und Steuerung von Smartwatches [15]. Durch den Open-Source Charakter soll sie eine alternative zu den propritären Programmen der Smartwatch-Hersteller geben. In diesem Projekt wird die App durch eine Schnittstelle zu AiXDroid ergänzt, um Daten von Smartwatches, im speziellen der BangleJS [9] an AiXDroid zu schicken, dort auszuwerten und anschließend, für eine visuelle Darstellung der ausgewerteten Daten, zurück an die Gadgetbridge zu übertragen

Die Gadgetbridge-App ist im Gegensatz zu AiXDroid in Java geschrieben und benutzt für die Benutzeroberfläche die Auszeichnungssprache XML. Die Gadgetbridge-Anwendung besitzt zwar eine abstrakte Implementierung zur Verwaltung von Smartwatches, jedoch gibt es keine bestehende Abstraktion, um einen Dienst bei einer Smartwatch-Art zu registrieren und so eine allgemeine Implementierung für das Sensor-Event zu erstellen. Es muss für jeden

Smartwatch-Typen eine eigene Implementierung der Schnittstelle erstellt werden. Daher wird in dieser Arbeit ausschließlich eine Implementierung für die BangleJS erstellt. Für die Implementierung meiner Anwendung wurde die Einstellungsseite „AiXDroidPreferencesActivity“ hinzugefügt, sowie die Serviceklasse „AiXDroidSender“. Auf der Einstellungsseite lässt sich die Schnittstelle aktivieren und deaktivieren. Außerdem wird hier das Default-Gerät ausgewählt von welchen Daten empfangen werden und an AiXDroid übermittelt werden sollen. Es können auch einzelne Sensoren für die Erfassung hinzugefügt oder ausgeschlossen werden. In der Serviceklasse werden die allgemeine Kommunikation mit AiXDroid und die Datenaufbereitung der von der Smartwatch empfangenen Sensorwerte vorgenommen. Über die Klasse „AiXDroidReceiver“, welche einen BroadcastReceiver implementiert (siehe Unterabschnitt 4.7.1), werden die Werte, die aus der AiXDroid-Datenbank gelesen werden sollen an die Gadgetbridge geschickt, anschließend im AiXDroidSender verarbeitet und abgespeichert.

5. Anwendung am Beispiel der Schlafphasenanalyse

5.1. Erstellung des KI-Modells

In diesem Kapitel wird der gesamte Prozess der Erstellung des KI-Modells für die Schlafphasenanalyse anhand von Smartwatch-Daten detailliert beschrieben. Das Modell wurde mithilfe von Python und TensorFlow entwickelt, um auf Grundlage von Smartwatch-Daten Schlafphasen zu erkennen und zu klassifizieren. Der Prozess umfasst mehrere Schritte, angefangen bei der Datenerhebung, über die Datenvorverarbeitung und die Definition des Modells, bis hin zum Training und der abschließenden Evaluierung.

5.1.1. Datenerhebung und -aufbereitung

Für die Aufzeichnung der Daten wurde eine Garmin-Smartwatch über mehrere Nächte getragen. Es handelt sich hierbei um die Garmin Venu-3. Sie ist unter anderem ausgestattet mit Beschleundigungs und Herzfrequenzssensoren. Die Werte dieser Sensoren sowie eine Vorhersage der aktuellen Schlafphase lassen sich über die Webseite von Garmin abrufen. Es gibt jedoch keinen direkte Download oder Export Funktion um die Daten zu erhalten. Eine Untersuchung der Webseite über die browser-interne Netzwerkanalyse zeigt jedoch einen Aufruf eines REST-Services, welcher die benötigten Daten enthält. Über ein Python-Skript wurde dieser Aufruf mittels Curl modifiziert, so dass die Daten manuell abgefragt und gesammelt werden können.

Die Rohdaten sind nun vorhanden und müssen für die weitere Verwendung aufbereitet werden. So muss zum Beispiel der Zeitindex angepasst werden, da dieser Relativ auf einen gegebenen Wert gesetzt wurde. Außerdem werden die Bewegungsdaten und Herzschlagdaten auf einen gemeinsamen Zeitindex gemapped. Die aufbereiteten Daten werden als JSON-Datei abgelegt um sie für das Training einfach wieder abrufbar zu haben.

5.1.2. Modelldefinition

Es wird nun die Struktur des KI-Modells beschrieben, das für die Schlafphasenanalyse auf Basis der Smartwatch-Daten verwendet wird. Für dieses Projekt wurde ein Modell auf Basis von Long Short-Term Memory (LSTM)-Netzwerken entwickelt, da diese Architektur besonders gut geeignet ist, um zeitabhängige Daten wie die Schlafphasen zu verarbeiten [18]. LSTM-Netzwerke sind in der Lage, langfristige Abhängigkeiten in den Daten zu lernen, was sie ideal für die Analyse von Zeitreihen, wie sie bei der Schlafüberwachung auftreten, macht. Es werden sowohl die Architektur des Modells als auch die spezifischen TensorFlow-Komponenten erläutert, die zur Implementierung des Modells verwendet wurden.

Es wurde die Klasse *TimeSeriesLSTM* entwickelt um die Modellstruktur zu kapseln und die wichtigsten Prozesse, wie die Datenvorverarbeitung, das Modelltraining sowie die Modellvorhersage zu steuern. Über die Datenstruktur *LSTMConfig* lassen sich Hyperparameter des Modells definieren, wie die Anzahl der Zeitschritte, auf die das Modell pro Berechnungsschritt zurückgreifen soll („n_steps“), die Batch-Größe, in welcher das Modell trainiert wird und die Anzahl der Epochen für das Training. Die folgenden Schichten (Layer) werden in dem verwendeten Modell eingesetzt.

- Input-Layer: Jede Feature-Spalte aus den Smartwatch-Daten wird als separate Eingabe behandelt.
- LSTM-Layer: Jede Eingabe wird durch eine LSTM-Schicht verarbeitet, um die zeitlichen Abhängigkeiten zu lernen.

- Concatenate: Die Ausgaben der LSTM-Schichten werden miteinander kombiniert.
- Dense Layer: Eine vollverbundene Schicht, die die Merkmale der LSTM-Ausgaben weiter verarbeitet.
- Output-Layer: Eine Softmax-Schicht, die die Wahrscheinlichkeiten für die verschiedenen Schlafphasen berechnet.

5.1.3. Modelltraining und -evaluation

Für das Modelltraining werden die Daten, welche zuvor als JSON-Dateien abgelegt wurden wieder geladen und an das Modell übergeben. Die Datenstruktur des TimeSeriesLSTM vereinfacht einem diesen Vorgang und übernimmt das eigentliche Training. Während des Trainings lässt sich der momentane Fortschritt anhand des aktuellen Loss im Terminal beobachten. Abschließend wird über den Evaluationsdatensatz das Modell überprüft und bewertet.

5.1.4. Integration der Metadaten

Im Tensorflow-Lite Framework werden Metadaten nachträglich der Modell-Datei hinzugefügt. Dies geschieht mit Hilfe des *tflite-support* Pakets mittels Python. Es werden die Angaben zu den Input-Tensoren angegeben, wie Name, Datentyp (in diesem Fall *float32*) sowie eine Beschreibung. Außerdem werden dem Modell selbst ebenfalls ein Name, eine Beschreibung, eine Versionsnummer, ein Lizenzeintrag und ein Author angefügt. Zur Ausführung in der AiXDroid-App sind nur der Name, und die Beschreibungen der Input-Tensoren notwendig. Die anderen Felder können leer gelassen werden. Zum Hinzufügen der Informationen wird ein Python-Skript verwendet. Anschließend kann die Datei auf das Android-Gerät übertragen werden.

5.2. Praktische Anwendung

6. Fazit und zukünftige Entwicklungen

In dieser Arbeit wurde die Grundlagen zum Verständnis von Android-spezifischen Tools und Frameworks erklärt, die grundlegenden Begrifflichkeiten und Forschungsstände in der Schlafforschung erläutert, sowie die Konzeption und Implementierung beschrieben, für eine Android-Anwendung, die mittels KI-Modell Daten lokal auf dem mobilen Gerät verarbeiten kann. Zur Demonstration wurde ein grundlegendes Modell zur Schlafphasenanalyse konzipiert, trainiert und auf Daten angewendet. Die App ist in der Lage mit anderen Anwendungen zu kommunizieren und Daten untereinander auszutauschen, um so dem Anwender die manuelle Ausführung der Datenverarbeitung zu ersparen und App-Entwicklern die Möglichkeit zu geben eine definierte Schnittstelle zur Ausführung von KI-Modellen zu verwenden.

Bei der Konzeption wurden Industriestandards in der Android-Entwicklung beschrieben und anschließend in der Implementierung umgesetzt. Durch die Anwendung der MVVM-Architektur in Verbindung mit Repository-Pattern und der Aufteilung in Datenschicht, Domänenlogikschicht und Benutzeroberflächenschicht wurde das Prinzip der Trennung von Verantwortlichkeiten und die Vorbereitung für einfaches Testen mit in die Struktur des Programms aufgenommen.

6.1. Zukünftige Entwicklungen

Die Anwendung ist in einem Zustand, in dem die Grundfunktionen bereits funktionsfähig implementiert sind. Es gibt jedoch noch Anwendungsfälle in

welchen Fehler auftreten können oder die nicht wie erwartet funktionieren. Daher sind die folgenden Punkte mögliche Anpassungen / Ergänzungen die für die Finalisierung der Anwendung nötig sind oder dem Anwender die Nutzung erleichtern würden.

Bug Fixing

Unittests

Anpassungen am Oberflächendesign

Das Farbschema ist noch nicht vollständig ausgearbeitet und kann in manchen Fällen unpassend sein. Über eine globale Farbschema-Datei ist die Farbgebung der Elemente in der Benutzeroberfläche definiert. Dadurch bleibt die Farbgebung über alle Elemente hinweg gleich und es lassen sich einfach neue Themen erstellen. Die Erstellung dieser Farbthemen ist nur auf Entwicklerseite möglich, da diese als Code in der „Theme.kt“ Datei festgelegt sind.

Mehr Konfigurationsmöglichkeiten durch Metadaten

Hinzufügen weiterer IntentActions

Optimierung der Inference-Logik

Implementierung einer Graph-Datenbank

A. Liste der Abkürzungen

NN	Neural Network
ML	Maschinelles Lernen
KI	Künstliche Intelligenz
NLP	Natural Language Processing
REM	Rapid Eye Movement
MSE	Mean Squared Error
MAE	Mean Absolute Error
LSTM	Long-Short-Term Memory
RNN	Recurrent Neural Network
DAO	Data Access Object
DCO	Data Class Object
MVVM	Model View Viewmodel
MVC	Model View Controller
IoT	Internet of Things

Literatur

[1] *Abhängigkeitsinjektion mit Hilt*. Android Developers. URL: https://developer.android.com/training/dependency-injection/hilt-android?hl=de (besucht am 09. 01. 2025).

[2] *Activity | Android Developers*. URL: https://developer.android.com/reference/android/app/Activity (besucht am 09. 01. 2025).

[3] *Android | Mehr Möglichkeiten mit Google für Android-Smartphones und -Geräte*. URL: https://www.android.com/intl/de_de/ (besucht am 20. 01. 2025).

[4] *Android Runtime and Dalvik*. Android Open Source Project. URL: https://source.android.com/docs/core/runtime (besucht am 20. 01. 2025).

[5] *Architecture Overview*. Android Open Source Project. URL: https://source.android.com/docs/core/architecture (besucht am 20. 01. 2025).

[6] Archiveddocs. *The MVVM Pattern*. 4. Okt. 2012. URL: https://learn.microsoft.com/en-us/previous-versions/msp-n-p/hh848246(v=pandp.10) (besucht am 06. 01. 2025).

[7] Archiveddocs. *The Repository Pattern*. 27. Apr. 2010. URL: https://learn.microsoft.com/en-us/previous-versions/msp-n-p/ff649690(v=pandp.10) (besucht am 08. 01. 2025).

[8] Aayush Bajaj. *Performance Metrics in Machine Learning [Complete Guide]*. neptune.ai. 21. Juli 2022. URL: https://neptune.ai/blog/performance-metrics-in-machine-learning-complete-guide (besucht am 06. 12. 2024).

[9] *Bangle.Js - Hackable Smart Watch*. URL: https://banglejs.com/ (besucht am 20. 01. 2025).

[10] G. M. Barthlen. „Schlafdiagnostik (Polysomnographie)“. In: *Klinische Pneumologie*. Hrsg. von Heinrich Matthys und Werner Seeger. Berlin, Heidelberg: Springer, 2002, S. 103–111. ISBN: 978-3-662-08120-4. DOI: 10.1007/978-3-662-08120-4_8. URL: https://doi.org/10.1007/978-3-662-08120-4_8 (besucht am 20. 01. 2025).

[11] *Berechtigungen auf Android-Geräten*. Android Developers. URL: https://developer.android.com/guide/topics/permissions/overview?hl=de (besucht am 18. 11. 2024).

[12] *BroadcastReceiver*. Android Developers. URL: https://developer.android.com/reference/android/content/BroadcastReceiver (besucht am 09. 01. 2025).

[13] *Dependency Injection in Android*. Android Developers. URL: https://developer.android.com/training/dependency-injection (besucht am 20. 01. 2025).

[14] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley, 9. März 2012. 558 S. ISBN: 978-0-13-306521-3. Google Books: vqTfNFDzzdIC.

[15] *Gadgetbridge*. URL: https://gadgetbridge.org/ (besucht am 20. 01. 2025).

[16] *Google I/O 2019: Empowering Developers to Build the Best Experiences on Android + Play*. Android Developers Blog. URL: https://android-developers.googleblog.com/2019/05/google-io-2019-empowering-developers-to-build-experiences-on-Android-Play.html (besucht am 12. 11. 2024).

[17] *Guide to App Architecture*. Android Developers. URL: https://developer.android.com/topic/architecture (besucht am 18. 11. 2024).

[18] Sepp Hochreiter und Jürgen Schmidhuber. „Long Short-term Memory“. In: *Neural computation* 9 (1. Dez. 1997), S. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

[19] *How Artificial Intelligence in Impacting Real Life Every Day*. URL: https://www.researchgate.net/publication/321348028_How_Artificial_Intelligence_in_Impacting_Real_Life_Every_day (besucht am 18. 11. 2024).

[20] Elif Kartal. „A Comprehensive Study on Bias in Artificial Intelligence Systems: Biased or Unbiased AI, That's the Question!“ In: *International Journal of Intelligent Information Technologies* 18 (1. Jan. 2022), S. 1–23. DOI: 10.4018/IJIIT.309582.

[21] Yann LeCun, Y. Bengio und Geoffrey Hinton. „Deep Learning“. In: *Nature* 521 (28. Mai 2015), S. 436–44. DOI: 10.1038/nature14539.

[22] *Maschinelles Lernen: Kompetenzen, Forschung, Anwendung*. Fraunhofer-Allianz Big Data
und Künstliche Intelligenz. URL: https://www.bigdata-ai.fraunhofer.de/de/publikationen/ml-studie.html (besucht am 05. 12. 2024).

[23] Lisa Matricciani u. a. „Rethinking the Sleep-Health Link“. In: *Sleep Health* 4.4 (1. Aug. 2018), S. 339–348. ISSN: 2352-7218. DOI: 10.1016/j.sleh.2018.05.004. URL: https://www.sciencedirect.com/science/article/pii/S2352721818300809 (besucht am 14. 11. 2024).

[24] Detlef Nauck, Frank Klawonn und Rudolf Kruse. *Neuronale Netze Und Fuzzy-Systeme*. 1. Aufl. Grundlagen Des Konnektionismus, Neuronaler Fuzzy-Systeme Und Der Kopplung Mit Wissensbasierten Methoden. vieweg. ISBN: 3-528-05265-1.

[25] J. A. Nelder und Y. Lee. „Likelihood, Quasi-Likelihood and Pseudolikelihood: Some Comparisons“. In: *Journal of the Royal Statistical Society Series B: Statistical Methodology* 54.1 (1. Sep. 1992), S. 273–284. ISSN: 1369-7412, 1467-9868. DOI: 10.1111/j.2517-6161.1992.tb01881.x. URL: https://academic.oup.com/jrsssb/article/54/1/273/7035820 (besucht am 20. 01. 2025).

[26] Kimessha Paupamah, Steven James und Richard Klein. „Quantisation and Pruning for Neural Network Compression and Regularisation“. In: *2020 International SAUPEC/RobMech/PRASA Conference*. 2020 International SAUPEC/RobMech/PRASA Conference. Jan. 2020, S. 1–6. DOI: 10.1109/SAUPEC/RobMech/PRASA48453.2020.9041096. URL: https://ieeexplore.ieee.org/document/9041096/?arnumber=9041096 (besucht am 13. 11. 2024).

[27] Marius-Constantin Popescu u. a. „Multilayer Perceptron and Neural Networks“. In: *WSEAS Transactions on Circuits and Systems* 8 (1. Juli 2009).

[28] Simone Cornelia Scharl. „Die Effekte von Schlaf auf die physiologische Reizverarbeitung und die emotionale Gedächtnisbildung“. In: ().

[29] *Schlafüberwachung | Gesundheitswissenschaft | Garmin-Technologie | Garmin*. URL: https://www.garmin.com/de-DE/garmin-technology/health-science/sleep-tracking/ (besucht am 14. 11. 2024).

[30] *Schreiben | Jetpack | Android Developers*. URL: https://developer.android.com/jetpack/androidx/releases/compose?hl=de (besucht am 09. 01. 2025).

[31] *Services Overview | Background Work*. Android Developers. URL: https://developer.android.com/develop/background-work/services (besucht am 09. 01. 2025).

[32] Tzu-An Song u. a. „AI-Driven Sleep Staging from Actigraphy and Heart Rate“. In: *PLOS ONE* 18.5 (17. Mai 2023). Hrsg. von Kathiravan Srinivasan, e0285703. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0285703. URL: https://dx.plos.org/10.1371/journal.pone.0285703 (besucht am 24. 07. 2024).

[33] *TensorFlow Lite Is Now LiteRT- Google Developers Blog*. URL: https://developers.googleblog.com/en/tensorflow-lite-is-now-litert/ (besucht am 20. 01. 2025).

[34] Olivia Walch u. a. „Sleep Stage Prediction with Raw Acceleration and Photoplethysmography Heart Rate Data Derived from a Consumer Wearable Device“. In: *Sleep* 42.12 (24. Dez. 2019), zsz180. ISSN: 0161-8105, 1550-9109. DOI: 10.1093/sleep/zsz180. URL: https://academic.oup.com/sleep/article/doi/10.1093/sleep/zsz180/5549536 (besucht am 24. 07. 2024).

[35] Phil Wennker. „Machine Learning“. In: *Künstliche Intelligenz in der Praxis: Anwendung in Unternehmen und Branchen: KI wettbewerbs- und zukunftsorientiert einsetzen*. Hrsg. von Phil Wennker. Wiesbaden: Springer Fachmedien, 2020, S. 9–37. ISBN: 978-3-658-30480-5. DOI: 10.1007/978-3-658-30480-5_2. URL: https://doi.org/10.1007/978-3-658-30480-5_2 (besucht am 08. 01. 2025).