

Sensorbasierte Aktivitätserkennung im Schwimmen

Projektbericht

Lukas Breiter, Alicia Jaramillo, Perinpagumar Munnathan

Modul: Sensor-based Activity Recognition

7. Juni 2026

Zusammenfassung

In diesem Bericht geht es um die Klassifizierung von vier Schwimmarten (Crawl, Rücken, Brust, Delfin) anhand von Sensordaten einer am Handgelenk getragener Uhr. Ziel der Challenge ist es, eine 90% Accuracy der Schwimmart-Klassifikation zu erreichen und ein Vergleich der Baseline Machine Learning Modelle und der Deep Learning Modelle zu bekommen. Um diese Klassifizierung zu verwirklichen, wurden zwei Baseline Machine Learning Modellen, der Logistischen Regression und dem Random Forest, und zwei Deep Learning Modellen, 1D-CNN und LSTM, trainiert. Bei allen Modellen fand eine Hyperparameter-Optimierung statt. Daraus resultierten insgesamt acht verschiedene Modelle. In der Challenge wurde herausgefunden, dass das getunte 1D-CNN das stärkste Modell ist. Auf Bahn-Ebene (ein Schwimmstil pro Bahn) erkennen aber alle Modelle den Stil fehlerfrei. Ungeklärt bleibt, wie das Modell auf andere Testsubjekte reagiert, mit verschiedenen Schwimmniveaus oder -stils. Dies kann in weiteren Untersuchungen herausgefunden werden.

Inhaltsverzeichnis

1	Einleitung und Zielsetzung	4
1.1	Zielsetzung und Erfolgskriterien	4
1.2	Abgrenzung	4
2	Projektorganisation und Vorgehen	4
2.1	Rollen im Team	4
2.2	Meilensteine	4
2.3	Repository- und Datenstruktur	5
2.4	Pseudonymisierung, Versionierung und Reproduzierbarkeit	5
3	Datenerfassung und Datensatz	5
3.1	Messaufbau und Geräte	5
3.2	Erfasste Sensoren und Abtastrate	6
3.3	Messprotokoll	6
3.4	Labeling und Splits	6
3.5	Trainings- und Test-Split	7
3.6	Datenartefakte pro Aufnahme	7
3.7	Datenqualität und Lückenerkennung	7
3.8	Limitationen der Datenerfassung	8
3.9	Preprocessing und Chunk-Indexierung	8
4	System- und App-Dokumentation	10
4.1	Architektur im Überblick	10
4.2	Geräte-Kommunikation über den Wearable Data Layer	10
4.3	Aufnahme auf der Uhr	12
4.4	Bedienoberfläche am Handy	12
4.5	Anbindung an die Prädiktions-API	12
4.6	Konfiguration und Betrieb	13
5	Modelle	14
5.1	Eingaberepräsentation und allgemeine Datenaufbereitung	14
5.2	Subjektbasierte Kreuzvalidierung	14
5.3	Klassische ML-Verfahren	14
5.3.1	Datenaufbereitung und Skalierung	14
5.3.2	Modellarchitekturen	15
5.3.3	Ergebniss der Subjektbasierten Kreuzvalidierung	15
5.3.4	Hyperparameter-Optimierung	16
5.3.5	Merkmalswichtigkeit (Permutationsanalyse)	18
5.3.6	Herausforderungen	18
5.4	Deep-Learning-Verfahren	19
5.4.1	Datenaufbereitung und Skalierung	19
5.4.2	Modellarchitekturen	19
5.4.3	Ergebniss der Subjektbasierten Kreuzvalidierung	20
5.4.4	Hyperparameter-Optimierung	20
5.4.5	Merkmalswichtigkeit (Permutationsanalyse)	22

6	Evaluation und Resultate	23
6.1	ONNX-Export und einheitliche Schnittstelle	23
6.2	Aggregation auf Bahn-Ebene: Soft- und Hard-Voting	23
6.3	Testaufbau	24
6.4	Gesamtranking	24
6.5	Konfidenz	24
6.6	Per-Chunk-Stimmverhalten	25
6.7	Auswertung pro Schwimmstil	27
6.8	Robustheit bei reduzierter Abtastrate	27
6.9	Fazit	28
7	Diskussion	28

1 Einleitung und Zielsetzung

Schwimmen lässt sich mit einer am Handgelenk getragenen Uhr nur schwer automatisch auswerten, weil das Wasser Funkverbindungen dämpft, die Uhr unter Wasser nicht bedienbar ist und kein einfacher Referenzwert für die Geschwindigkeit vorliegt. Wir nutzen die Inertialsensorik einer Samsung Galaxy Watch 4, um aus den Bewegungen am Handgelenk die Schwimmart zu erkennen und die Pace einzuschätzen. Vergleichbare Arbeiten zur inertialen Aktivitätserkennung im Schwimmen beschreiben dieselben Hürden und liefern die methodische Einordnung (Mooney et al., 2016).

1.1 Zielsetzung und Erfolgskriterien

Das Projekt erkennt eine Grösse aus den Sensordaten des Handgelenks:

- **Schwimmart** (Stroke-Klassifikation): Crawl, Rücken, Brust, Delfin.

Als messbare Ambitionswerte setzen wir mindestens 90 % Accuracy bei der Stroke-Klassifikation (vier Klassen) Dieser Wert dient als Erfolgskriterium, nicht als Garantie.

1.2 Abgrenzung

Innerhalb des Projekts liegen die Datenerhebung im Verein samt Standardprotokoll, eine reproduzierbare Pipeline von der Vorverarbeitung bis zur Evaluation sowie der Vergleich klassischer ML-Baselines mit Deep-Learning-Modellen für Zeitreihen. Ausserhalb liegen jede medizinische oder diagnostische Aussage, eine vollständige biomechanische Modellierung, ein exakter Momentangeschwindigkeits-Referenzwert ohne Video oder Marker-Splits sowie ein Echtzeit-Deployment auf der Uhr. Starts, Wenden und Stopps behandeln wir als Übergänge (Label „Transition“) und entfernen sie bei Bedarf aus den Trainingsfenstern.

2 Projektorganisation und Vorgehen

2.1 Rollen im Team

Wir teilen die Arbeit nach inhaltlichen Paketen auf. Datenerfassung, Organisation und die Android-Anwendung (Handy und Uhr) liegen bei einer Person; die klassischen ML-Verfahren und die Deep-Learning-Modelle bearbeiten die beiden anderen Teammitglieder. Tabelle 1 ordnet die Arbeitspakete den Verantwortlichen zu.

Arbeitspaket	Verantwortlich
Organisation, Repository, Datenstruktur	<i>Lukas Breiter, Alicia Jaramillo, Perinpagumar Munnathan</i>
Datenerfassung und Labeling	<i>Lukas Breiter</i>
Android-App (Handy, Uhr, Cloud-Anbindung)	<i>Lukas Breiter</i>
Klassische ML-Verfahren	<i>Alicia Jaramillo</i>
Deep-Learning-Verfahren	<i>Perinpagumar Munnathan</i>

Tabelle 1: Arbeitspakete und Zuständigkeiten.

2.2 Meilensteine

Tabelle 2 hält die geplanten Termine mit Definition of Done und Stand fest.

Termin	ID	Meilenstein (Definition of Done)	Stand
20.03.2026	Planung	Planungsdokument für Messaufbau, Datenstruktur und Evaluationsvorgehen finalisiert.	erledigt
25.03.2026	Erste Datenerhebung	Erste Daten im finalen Rohdatenformat mit Labels im Repository verfügbar.	erledigt
07.05.2026	Modellarchitektur & Stopp Datenaufnahme	Modelle entworfen und trainingsbereit, Datenerhebung abgeschlossen.	erledigt
07.06.2026	Abgabe Bericht	Bericht und Projekt eingereicht.	erledigt
18.06.2026	Präsentation	Projekt präsentiert.	offen

Tabelle 2: Meilensteine des Projekts.

2.3 Repository- und Datenstruktur

Wir legen Rohdaten, Vorverarbeitung, Code und Ergebnisse getrennt ab. Jede Rohaufnahme erhält einen eigenen Ordner, dessen Name die Schwimmart und die pseudonymisierte Person kombiniert.

```

data/train/raw/<stroke>_<subject_id>/      # eine Rohaufnahme, z.B. Brust_S001/
  samples.csv      # unveränderte Sensorsamples der Aufnahme
  events.csv       # Ereignisse während der Aufnahme (Start, Sensor-Gaps, ...)
  manifest.json    # Aufnahme-Metadaten aus der App
  status.json      # letzter Aufnahmestatus
  labels.csv       # Splits und Trial-Labels (manuell/automatisiert)
data/train/preprocessed/    # synchronisiert, gefenstert, gelabelt
data/test/                # ungesehene Personen fuer die finale Evaluation
src/                      # Preprocessing, Modellierung, Evaluation
tests/                   # Tests fuer Datenpipeline und Label-Verarbeitung
Notebooks/               # explorative Analysen und Prototypen
  HTML/                # html der Notebooks
results/                 # Modelle, Metriken, Visualisierungen

```

2.4 Pseudonymisierung, Versionierung und Reproduzierbarkeit

Personen tragen ausschliesslich pseudonyme Kennungen (S001, S002, ...); im Analyse-Datensatz stehen keine Namen. Die eigentliche Session-ID bleibt in den Metadateien der Aufnahme erhalten. Datensätze versehen wir mit festen Versionsnummern (z. B. `v0-pilot`, `v1-main`) und dokumentieren das Label-Schema, damit sich die Pipeline reproduzieren lässt.

3 Datenerfassung und Datensatz

3.1 Messaufbau und Geräte

Der Schwimmer trägt eine Samsung Galaxy Watch 4 am Handgelenk. Die Uhr (`:wear-App`) zeichnet die Inertialsensorik auf; ein Android-Handy (`:mobile-App`) steuert die Aufnahme, nimmt die Splits auf und übernimmt später den Upload. Pro Aufnahme schwimmt der Schwimmer 50 m. Der Bediener steht am Beckenrand, startet und stoppt die Aufnahme über das Handy und drückt die Splits manuell.

3.2 Erfasste Sensoren und Abtastrate

Die Uhr registriert fünf Sensoren des Android-Sensor-Frameworks. Tabelle 3 listet sie mit dem rohen `sensor_type`-Wert aus `samples.csv` und der Einheit.

<code>sensor_type</code>	Sensor	Einheit	Bemerkung
1	Accelerometer	m/s^2	inklusive Gravitation
2	Magnetometer	μT	Magnetfeld
4	Gyroscope	rad/s	Drehrate
9	Gravity Vector	m/s^2	geschätzte Gravitationskomponente
10	Linear Acceleration	m/s^2	ohne Gravitation

Tabelle 3: Aufgezeichnete Sensoren. Die Werte liegen im Geräte-Koordinatensystem (Body-Frame), nicht im Erdsystem.

Die Abtastrate beträgt standardmässig 50 Hz; die App unterstützt zusätzlich 25 Hz und 100 Hz. Für Aufnahmen unter Wasser erlaubt die Konfiguration eine Batching-Latenz der Sensor-Hardware (`maxReportLatencyUs`): Dösst die CPU trotz WakeLock kurz ein, füllt die Hardware ihren FIFO weiter und liefert die Samples beim Aufwachen nach. Beim Testen setzen wir die Latenz auf 0, damit Lücken in `events.csv` echte Aussetzer zeigen und keine erwartete Hardware-Verzögerung.

3.3 Messprotokoll

Eine Aufnahme folgt festen Schritten:

1. Am Handy Schwimmt, Personen-ID und Sensoren wählen; daraus erzeugt die App einen Session-Namen wie `Brust_S001_Session01`.
2. „Start Recording“ drücken. Das Handy sendet den Startbefehl an die Uhr, die Uhr beginnt zu schreiben.
3. Der Schwimmer schwimmt 50 m. Der Bediener startet parallel die Stoppuhr am Handy und drückt die Splits.
4. „Stop Recording“ drücken. Die Uhr schliesst die Dateien.
5. „Download from Watch“ drücken. Die Uhr packt die Session als ZIP und überträgt sie; das Handy mischt die Splits in eine `labels.csv`.

3.4 Labeling und Splits

Der Bediener erfasst die Zwischenzeiten über eine Stoppuhr am Handy. Jede Rohaufnahme erhält eine eigene `labels.csv` mit festem Schema:

```
stroke,athlete,split_10m_s,split_20m_s,turn_s,turn_end_35m_s,split_40m_s,split_50m_s
Brust,S001,7.41,15.88,24.10,28.55,36.02,44.77
```

Tabelle 4 erklärt die Spalten. Die Zeiten stehen in Sekunden ab dem Start der Stoppuhr; die Wende erfasst sowohl den Eintritt (`turn_s`) als auch das Ende (`turn_end_35m_s`).

Spalte	Bedeutung
stroke	Schwimmart der Aufnahme
athlete	pseudonyme Personen-ID
split_10m_s	Zeit bei 10 m
split_20m_s	Zeit bei 20 m
turn_s	Zeit beim Erreichen der Wende
turn_end_35m_s	Zeit beim Abschluss der Wende (bei 35 m)
split_40m_s	Zeit bei 40 m
split_50m_s	Zeit bei 50 m (Ziel)

Tabelle 4: Spalten der `labels.csv`.

Die Startpeaks (ein Tap zu Beginn) dienen als Align-Event zwischen der IMU-Zeitachse und der Stoppuhr. Starts, Wenden und Stopps labeln wir als „Transition“ und entfernen sie bei Bedarf aus den Trainingsfenstern.

3.5 Trainings- und Test-Split

Wir trennen Personen sauber zwischen Training und Test (subject-wise split), damit die Modelle über Personen hinweg generalisieren. Die Trainingsdaten stammen aus einem Hallenbad (Liestal). Für den Test schwamm eine Person, die in keiner Trainingsaufnahme vorkommt, dasselbe Protokoll in einem anderen Hallenbad (Brugg) mit kürzerer Bahn. Diese Trennung prüft, ob die Modelle eine fremde Person, ein anderes Becken und eine andere Bahnlänge verkraften.

3.6 Datenartefakte pro Aufnahme

Die Uhr schreibt pro Session vier Dateien, das Handy ergänzt die `labels.csv`. Die Kopfzeile von `samples.csv` enthält neben den Mess- und Zeitfeldern mehrere Tracing-Spalten, mit denen sich Reihenfolge, Thread und Flush-Batch der Samples nachvollziehen lassen:

```
session_id,sensor_type,event_timestamp_ns,received_elapsed_realtime_ns,wall_time_ms,accuracy,x,y,z,
sequence_id,callback_thread_id,flush_id,enqueue_time_ns,dequeue_time_ns,queue_size
```

Die `events.csv` hält Ereignisse mit Zeitstempel fest:

```
session_id,wall_time_ms,event_type,details
```

Die `manifest.json` dokumentiert die Aufnahme: Session-ID, Uhrmodell, App-Version, gewählte Sensoren, beabsichtigte Abtastrate, technische Sensorinfos (Hersteller, Auflösung, Messbereich) sowie Start- und Endzeit.

3.7 Datenqualität und Lückenerkennung

Während der Aufnahme prüft ein `GapDetector` den zeitlichen Abstand aufeinanderfolgender Samples je Sensortyp. Überschreitet der Abstand 200 ms, schreibt die App ein `gap_detected`-Ereignis und erhöht die `flush_id`; alle Samples nach einer Lücke tragen die neue `flush_id`, womit sich Bursts nach einem Aussetzer leicht erkennen lassen. Weitere Ereignistypen sind `session_started`, `session_stopped`, `service_restarted`, `sensor_unavailable`, `export_started` und `export_finished`. Die Fensterung und die Ableitung der Trainingslabels orientieren sich an der üblichen Vorgehensweise für inertielle Aktivitätserkennung (Bulling et al., 2014).

3.8 Limitationen der Datenerfassung

Mehrere Punkte schränken den Datensatz ein. Es gibt keinen exakten Momentangeschwindigkeits-Referenzwert; die Pace stützt sich auf manuell gedrückte Splits, deren Timing eine Reaktionszeit von einigen Zehntelsekunden enthält. Die Stichprobe ist klein und stammt aus zwei Hallenbädern. Wenden und Starts erzeugen Übergänge, die die Klassifikation stören können, weshalb wir sie gesondert behandeln. Diese Punkte greifen wir in der Diskussion erneut auf.

3.9 Preprocessing und Chunk-Indexierung

Zwischen der Rohaufnahme und der Modellierung steht eine zweistufige Vorverarbeitung im Paket `preprocessing_pipe` (Module `detection.py`, `build_index.py`, `feeder.py`). Sie überführt jede Rohaufnahme in trainingsfertige *Bahn-Chunks*: zuerst werden die Sensoren zusammengeführt und die einzelnen Bahnen erkannt (`build_index`), anschliessend liefert ein `ChunkFeeder` die Chunks direkt für das Training aus. Das trennt die teure, einmalige Erkennung von der günstigen, wiederholten Auslieferung und entkoppelt den Modellierungscode vom Rohdaten-Layout.

Sensormerge auf eine gemeinsame Zeitachse. Die fünf Sensoren werden asynchron mit leicht unterschiedlichen Zeitstempeln aufgezeichnet. Als Basis-Zeitachse dient der Accelerometer; die übrigen vier Sensoren (Magnetometer, Gyroskop, Gravitation, lineare Beschleunigung) werden per linearer Interpolation auf diese Zeitachse abgebildet (`merge_sensors_interpolated`), wobei die Ränder extrapoliert werden. Das Ergebnis ist ein zusammengeführter Datenrahmen mit `timestamp_ns` sowie fünf Sensoren $\times$ drei Achsen (16 Spalten). Die Zeitstempel werden auf den Aufnahmestart (0) normiert und die effektive Abtastrate wird über den Median der Sample-Abstände geschätzt.

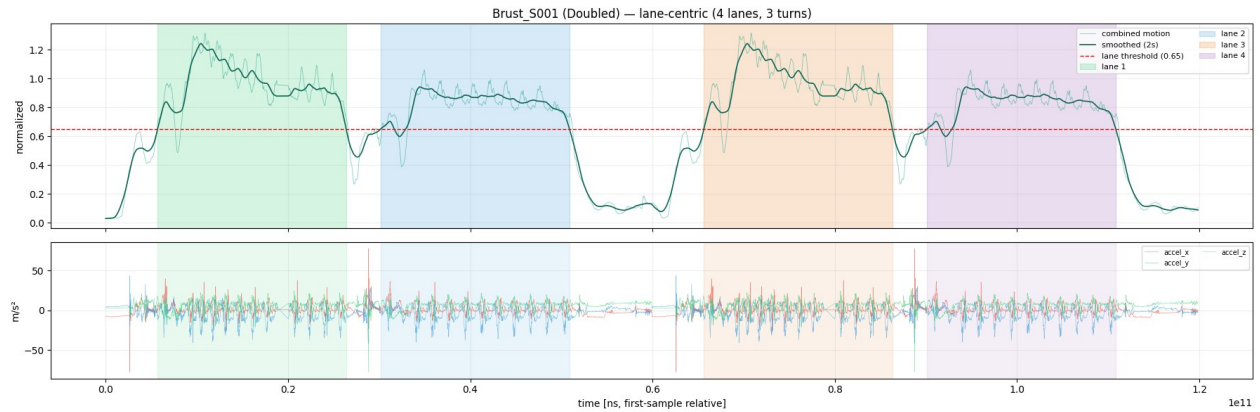
Kombiniertes Bewegungssignal. Für die Chunk-Erkennung wird aus den Sensoren ein robustes Bewegungssignal abgeleitet. Pro Achse werden die Gravitations- und die Linearbeschleunigungs-Kanäle leicht vorgeglättet (0,2s), differenziert und zu einer Änderungsrate (Betrag $\times$ Abtastrate) verrechnet, anschliessend über ein 2s-Fenster geglättet. Die beiden Raten (Gravitations- und Linearbeschleunigungs-Bewegung) werden je auf ihr 80 %-Perzentil normiert und sample-weise über das Minimum kombiniert. Dadurch ist das Signal robust gegen Wackeln innerhalb einer Bahn, das nur eine der beiden Raten betrifft.

Bahn-zentrierter Plateau-Detektor. Der Detektor (`extract_n_chunks`) glättet das Bewegungssignal stark (2s) zu klaren „Plateaus“. Ein adaptiver Schwellwert (Faktor 0,7 des aktiven Referenzniveaus, geschätzt als Median der oberen Signalhälfte) markiert die hohen Blöcke. Kurze Einbrüche unterhalb von `min_turn_s` (1,5s) werden als Wenden interpretiert und überbrückt, d. h. mit dem vorhergehenden Block verschmolzen; Blöcke kürzer als `min_swim_s` (3,0s) werden verworfen. Jeder verbleibende Block entspricht genau einer Bahn und wird mit Start- und End-Index, Start- und End-Zeitstempel sowie Dauer beschrieben.

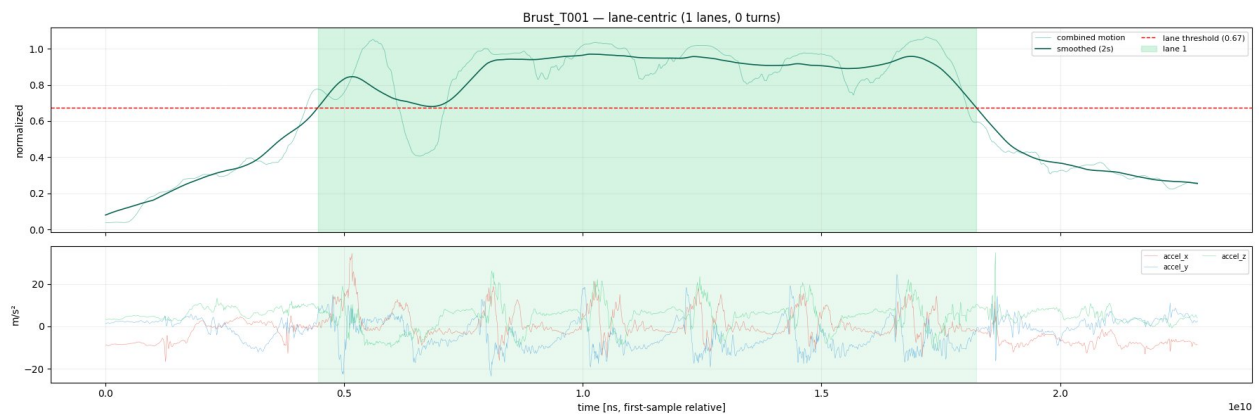
Indexaufbau. `build_index` durchläuft das Rohdatenverzeichnis, führt pro Aufnahme die Erkennung aus und schreibt einen `chunks_index.csv` im Langformat (eine Zeile pro Chunk) nach `data/<split>/preprocessed/`. Der Aufbau ist inkrementell: bereits indizierte Aufnahmen werden bei einem erneuten Lauf übersprungen, sofern nicht `overwrite=True` gesetzt ist. Der Index hat folgendes Schema:

```
recording,chunk,stroke,start_idx,end_idx,start_ns,end_ns,duration_s
```

Auslieferung über den ChunkFeeder. Der **ChunkFeeder** liest den Index, lädt die zusammengeführten Datenrahmen verzögert (*lazy*) und hält sie im Cache. Er ist iterierbar und liefert pro Chunk ein Sample-Objekt mit Aufnahme, Chunk-Index, Schwimmtart (**stroke**), Dauer und den eigentlichen Sensordaten (**data**); die Schwimmtart stammt dabei aus dem Ordnernamen der Aufnahme. Damit lässt sich der Feeder direkt in eine Trainingsschleife einsetzen, während `feeder.get(recording)` alle Chunks einer einzelnen Aufnahme zurückgibt (nützlich zum Debuggen).



(a) Künstlich verdoppelte Trainingsaufnahme (vier Bahnen, drei Wenden): Die kurzen Einbrüche an den Wenden liegen unter `min_turn_s` und werden überbrückt, sodass der Detektor die vier Bahnen sauber trennt.



(b) Reale Testaufnahme der ausgelagerten Person T001: eine einzelne 50 m-Bahn ohne Wende, die als genau ein Chunk erkannt wird.

Abbildung 1: Bahn-zentrierte Chunk-Erkennung. Oben je Teilbild das kombinierte Bewegungssignal mit geglätteter Variante (2s), adaptivem Schwellwert und den erkannten Bahnen (farbig hinterlegt), unten die rohen Accelerometer-Achsen mit denselben Bahn-Markierungen.

4 System- und App-Dokumentation

Die Anwendung besteht aus drei Kotlin-Modulen. Die UI baut auf Jetpack Compose, die Architektur folgt dem MVVM-Muster, die Cloud-Anbindung nutzt Retrofit und OkHttp, asynchrone Abläufe laufen über Kotlin Coroutines und Flow.

4.1 Architektur im Überblick

Das `:shared`-Modul stellt die gemeinsamen Datenmodelle, die Kommunikationspfade, die Befehlskonstanten und die JSON-Serialisierung bereit und wird von beiden Apps genutzt. Das `:mobile`-Modul steuert Aufnahme, Stoppuhr und Cloud-Prädiktion. Das `:wear`-Modul nimmt die Sensordaten auf Hardware-Ebene auf. Abbildung 10 zeigt die Module und ihre Kommunikationswege.

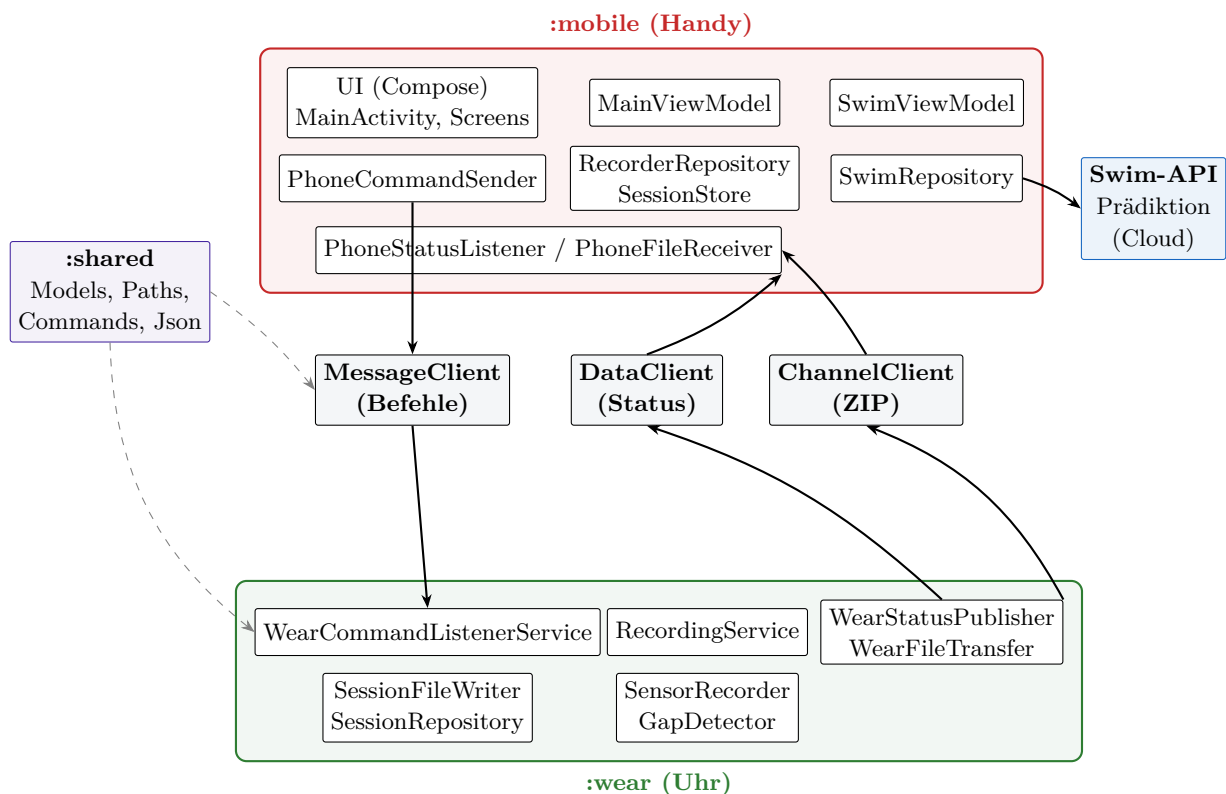


Abbildung 2: Modul- und Kommunikationsarchitektur. Handy und Uhr tauschen über drei Mechanismen des Wearable Data Layer aus; das Handy spricht zusätzlich die Cloud-API an. Gestrichelte Pfeile sind Abhängigkeiten auf das `:shared`-Modul.

4.2 Geräte-Kommunikation über den Wearable Data Layer

Handy und Uhr kommunizieren über die Wearable-API von Google Play Services. Drei Mechanismen teilen sich die Aufgaben (Tabelle 5): ein `MessageClient` für kurze Befehle, ein `DataClient` für synchronisierten Status und ein `ChannelClient` für den Transfer der ZIP-Datei. Alle Pfade und Befehle stehen als Konstanten im `:shared`-Modul (`Paths.kt`, `Commands.kt`).

Aktion	Quelle	Ziel	Mechanismus	Nutzlast
Aufnahme starten	Handy	Uhr	Message	SessionConfig
Aufnahme stoppen	Handy	Uhr	Message	—
Statusupdate	Uhr	Handy	Data Item	SessionStatus
Export anfragen	Handy	Uhr	Message	ExportRequest
Daten exportieren	Uhr	Handy	Channel	ZIP-Datei

Tabelle 5: Kommunikationsprotokoll zwischen Handy und Uhr.

Abbildung 3 zeigt den vollständigen Fluss einer Aufnahme: vom Startbefehl über den Statuswechsel und den ZIP-Transfer bis zum Upload und dem Polling gegen die Prädiktions-API. Für grosse Transfers über langsame Bluetooth-Verbindungen wartet das Handy bis zu 30 Minuten; nach einem Resume gleicht der `MainViewModel` den UI-Zustand über `refreshStatus()` mit dem echten Zustand der Uhr ab.

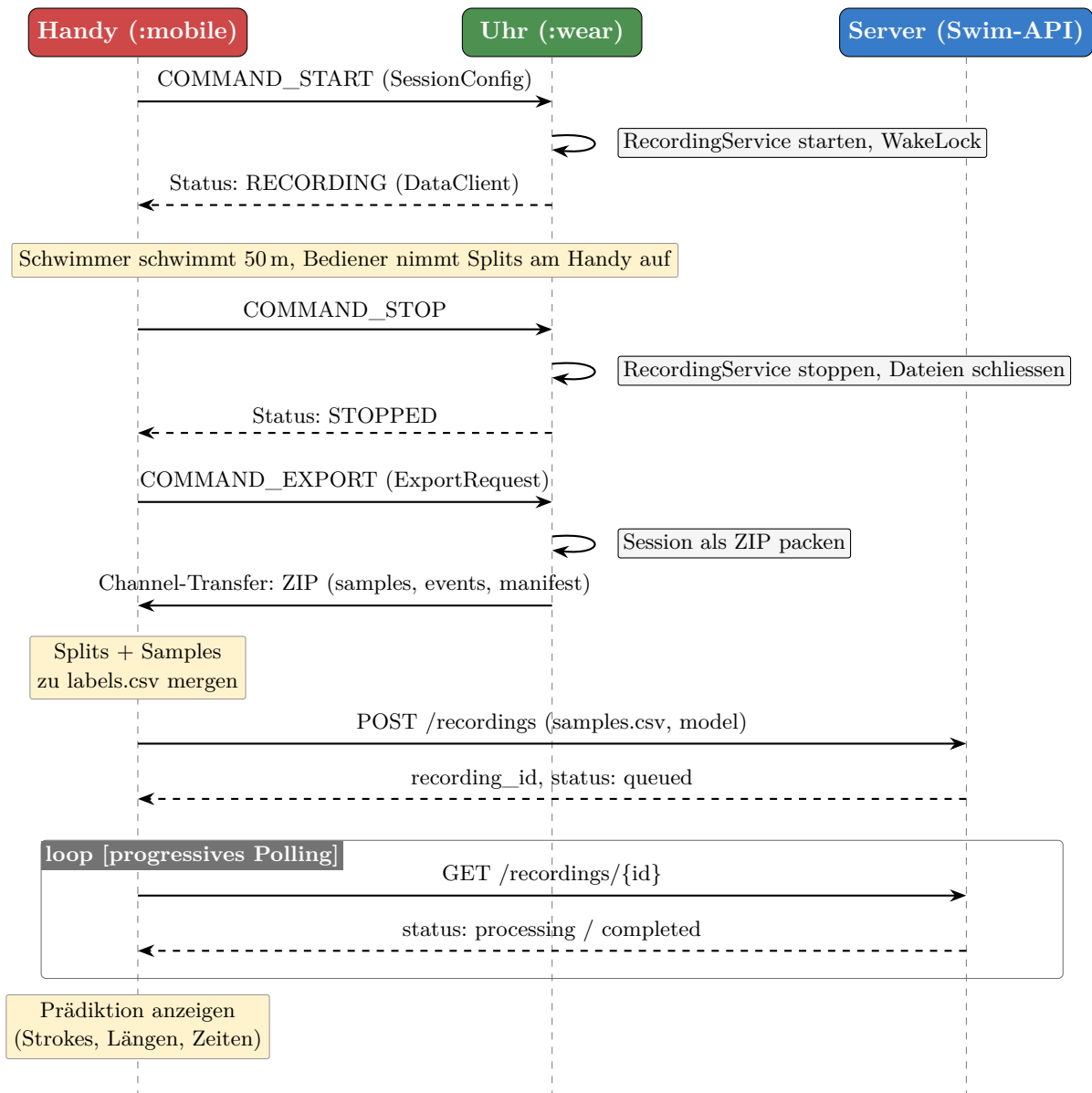


Abbildung 3: Netzwerk- und Kommunikationsfluss einer Aufnahme: Steuerbefehle und Status über den Wearable Data Layer (Handy ↔ Uhr), ZIP-Transfer über einen Channel, anschliessend Upload und Polling gegen die Prädiktions-API.

4.3 Aufnahme auf der Uhr

Ein `RecordingService` als `Foreground-Service` hält die Aufnahme am Leben und hält über einen `PARTIAL_WAKE_LOCK` die CPU wach, auch wenn der Bildschirm aus ist oder die Uhr unter Wasser liegt. Der `SensorRecorder` registriert die `Sensor-Listener` auf einem eigenen `Handler-Thread`, vergibt eine fortlaufende `sequence_id` und erkennt Lücken über den `GapDetector`. Der `SessionFileWriter` schreibt `samples.csv`, `events.csv`, `manifest.json` und `status.json` in den Session-Ordner. Abbildung 4 zeigt das Zustandsmodell des Service.

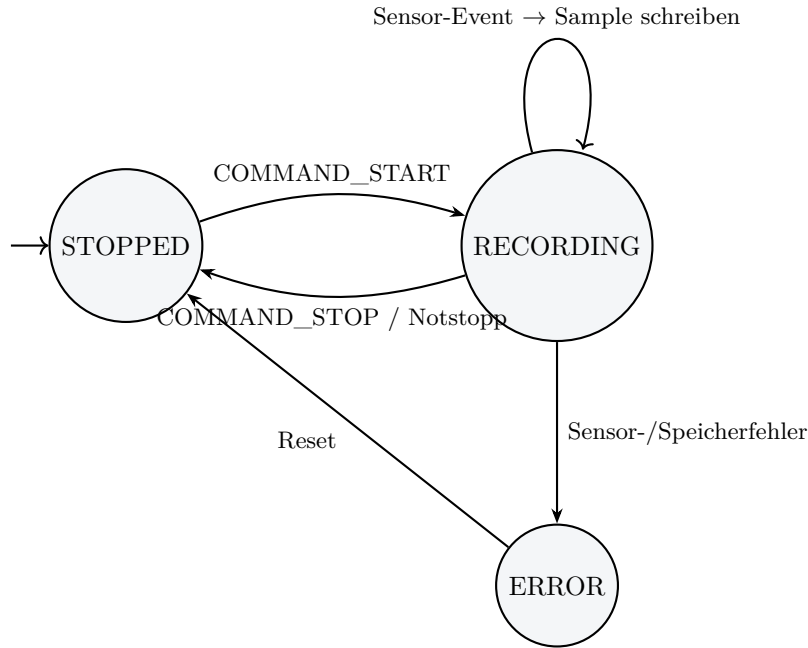


Abbildung 4: Zustandsmodell des `RecordingService` auf der Uhr. Während `RECORDING` schreibt der `Writer-Thread` jeden `Sensor-Event` nach `samples.csv`. Die Aufzählung `RecordingState` kennt zusätzlich `IDLE` vor der ersten Aufnahme.

4.4 Bedienoberfläche am Handy

Die Handy-App führt durch vier Bildschirme. Auf dem `HomeScreen` stellt der Bediener Schwimmart, Personen-ID, Sensoren und Abtastrate ein und startet die Aufnahme. Der `StopwatchScreen` liefert eine Stoppuhr mit Tasten für die Splits (10 m, 20 m, Wende, 40 m, 50 m). Der `SwimPredictionScreen` fasst die automatisierte Prädiktion zusammen: Modellwahl, Start und Stopp der Uhr, Upload und Anzeige der Ergebnisse samt Verlauf. Der `ExportScreen` listet empfangene Sessions und exportiert sie als Datei.

4.5 Anbindung an die Prädiktions-API

Die Handy-App lädt die extrahierte `samples.csv` an einen entfernten Inferenzdienst (Swim-API) und holt die Schwimmart-Prädiktion ab. Basis-URL und Token kommen aus `BuildConfig` (`SWIM_BASE_URL`, `SWIM_API_TOKEN`); die Authentifizierung läuft über einen Bearer-Token im `Authorization-Header`. Tabelle 6 fasst die Endpunkte zusammen.

Methode	Endpunkt	Zweck
GET	/models	verfügbare ML-Modelle auflisten
POST	/recordings	<code>samples.csv</code> als multipart/form-data hochladen
GET	/recordings/{id}	Status und Ergebnis einer Aufnahme abfragen
GET	/recordings	Verlauf verarbeiteter Aufnahmen (Parameter <code>limit</code>)

Tabelle 6: Endpunkte der Prädiktions-API.

Der Upload (POST `/recordings`) sendet das Feld `model` (z. B. `cnn_tuned`), das Flag `verbose`, optional einen `recording_name` und die Datei `file`. Die Antwort liefert eine `recording_id` mit Status `queued`. Danach pollt das `SwimRepository` über einen Coroutine-Flow mit progressivem Intervall: zuerst jede Sekunde, nach 10 Sekunden alle 3 Sekunden, später alle 10 Sekunden, mit Abbruch nach 5 Minuten oder bei Status `completed` bzw. `failed`. Eine fertige Antwort enthält die Schwimmart-Sequenz, die Anzahl Längen sowie Gesamt- und aktive Zeit:

```
{
  "code": 200,
  "recording_id": "rec_8f1c...",
  "status": "completed",
  "model": "cnn_tuned",
  "result": {
    "length_count": 2,
    "pred_sequence": "breaststroke,breaststroke",
    "tot_time": "44.77",
    "act_time": "41.20",
    "chunks": [
      { "chunk": 0, "stroke": "breaststroke", "confidence": 0.97, "duration_s": 22.1 }
    ]
  }
}
```

Fehler fängt das Repository über `runCatching` ab und gibt `Result`-Objekte zurück; `OkHttp` arbeitet mit 30 Minuten Lese- und Schreib-Timeout für grosse Uploads, und der `SwimViewModel` zeigt Fehlermeldungen auf dem Bildschirm an.

4.6 Konfiguration und Betrieb

Vor dem Bauen trägt man Token und Basis-URL in `local.properties` ein:

```
SWIM_API_TOKEN=dein_token
SWIM_BASE_URL=https://<host>/
```

Das Handy benötigt mindestens API-Level 24 und die Berechtigungen `INTERNET` sowie `WAKE_LOCK`. Die Uhr (empfohlen API-Level 30 oder neuer) nutzt `FOREGROUND_SERVICE` (Typ `dataSync`), `BODY_SENSORS`, `HIGH_SAMPLING_RATE_SENSORS` und `WAKE_LOCK`. Gebaut werden die Module `:mobile` und `:wear` in Android Studio; vor dem End-to-End-Test prüft man die Verbindung zwischen Handy und Uhr.

5 Modelle

Für die Klassifikation der vier Schwimmstile (Rücken, Brust, Delfin/Schmetterling und Kraul) wurden zwei Baseline-Modelle und zwei Deep-Learning-Architekturen umgesetzt und verglichen:

- Für die Baselinemodelle die Logistische Regression und ein Tree-basierter Ansatz,
- Für die Deep-Learning-Modelle ein eindimensionales Convolutional Neural Network (1D-CNN) sowie ein rekurrentes Netz auf Basis von Long-Short-Term-Memory-Zellen (LSTM).

5.1 Eingaberepräsentation und allgemeine Datenaufbereitung

Der Datensatz wird strikt subjektbasiert aufgeteilt: Die Trainingsmenge umfasst 56 Bahn-Chunks von acht Proband:innen (S001-S007, S100), die Testmenge vier Chunks einer separaten, im Training nicht gesehenen Person (T001). Diese Trennung verhindert, dass subjektspezifische Bewegungsmuster in beide Mengen gelangen, und liefert eine realistische Schätzung der Generalisierungsfähigkeit auf neue Schwimmer:innen.¹ Die zusammengeführten IMU-Aufnahmen (Beschleunigung, Magnetometer, Gyroskop, Gravitation und lineare Beschleunigung, je drei Achsen) ergeben einen Datenstrom mit 15 Kanälen bei einer Abtastrate von 100 Hz. Aus diesem Strom werden mit einem gleitenden Fenster Sequenzen fester Länge extrahiert (*seq_len* = 100 Abtastwerte, Schrittweite *step* = 50, also 50 % Überlappung).

5.2 Subjektbasierte Kreuzvalidierung

Zur Modellauswahl und Hyperparameter-Optimierung wird eine subjektbasierte Leave-*N*-Out-Kreuzvalidierung mit acht Folds eingesetzt, bei der pro Fold jeweils eine primäre Person als Validierungsmenge dient. Um sicherzustellen, dass jede Validierungsmenge alle vier Klassen abdeckt, werden bei Bedarf einzelne „Companion“-Subjekte ergänzt (Klassenabdeckungs-Garantie). Diese Aufteilung respektiert die Subjektentrennung auch innerhalb der Validierung und verhindert eine optimistische Verzerrung der Kennzahlen.

5.3 Klassische ML-Verfahren

Für die klassischen ML-Modellen wurden je ein lineares, parametrisches Modell und ein baumbasiertes, regelorientiertes Modell gewählt. Für das lineare, parametrische Modell die Logistische Regression und für das baumbasierte, regelorientierte zuerst ein Decision Tree danach aber ein Random Forest. Dieser Wechsel vom einzelnen Entscheidungsbaum hin zu einem Random Forest erfolgte aufgrund der veränderten Input-Datengröße und der beobachteten Overfitting-Tendenzen des einzelnen Baums.²

5.3.1 Datenaufbereitung und Skalierung

Die gleitenden Fenstersequenzen der allgemeinen Datenaufbereitung werden für die klassischen Modelle noch weiter vorbereitet. Über jedes Fenster wird der Mittelwert (*mean*), der Median (*median*), die Standardabweichung (*std*), das Maximum (*max*) und das Minimum (*min*) aggregiert und als Eingabewerte für die beiden Modelle verwendet.

Die aggregierten Fensterdaten werden mit dem `StandardScaler` von `sklearn` normiert.

¹Die Lanen-Chunks stammen aus `StrokeChunkDetection.html` bzw. `ChunkIndexing.html`.

²Sämtliche in diesem Abschnitt berichteten Ergebnisse – Training, Kreuzvalidierung, Hyperparameter-Tuning und Permutationsanalyse – wurden mit dem Notebook `ML_Models.html` erzeugt.

5.3.2 Modellarchitekturen

Die **logistische Regression** modelliert die aggregierten Fensterinputdaten als lineare Kombination der Eingangsvariablen und wandelt diese über eine Sigmoid-Funktion in Klassenwahrscheinlichkeiten um.

Der **Random Forest** aggregiert hingegen viele Entscheidungsbäume, die jeweils auf zufälligen Teilstichproben und Merkmalsunterräumen trainiert werden. Dadurch ist das Modell robuster gegenüber Ausreißern und reduziert die Varianz eines einzelnen Baums durch Mittelung über viele Teilmodelle.

5.3.3 Ergebniss der Subjektbasierten Kreuzvalidierung

Der Vergleich der subjektbasierten Kreuzvalidierung der beiden Modelle zeigt sich folgendermassen:

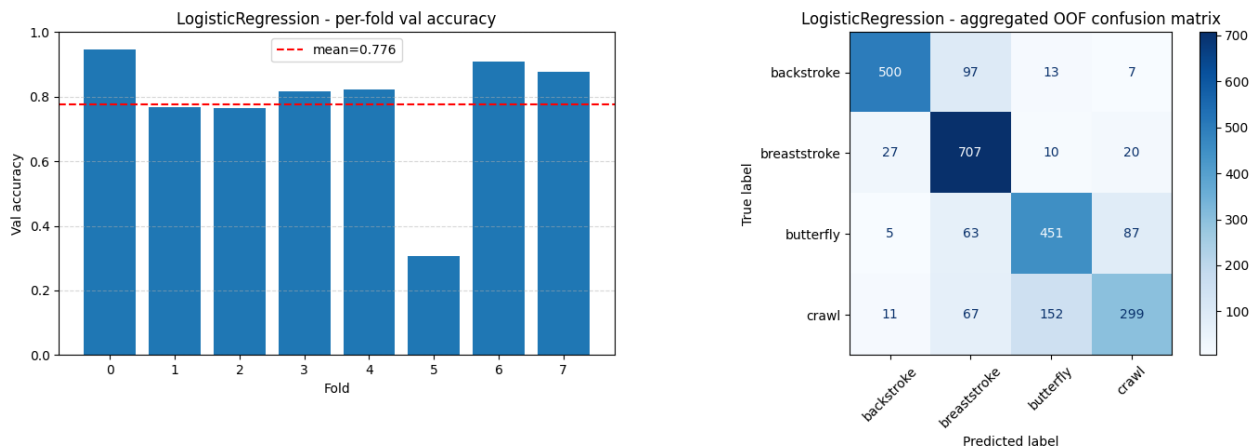


Abbildung 5: Logarithmische Regression-Kreuzvalidierung. Links: Validierungs-Accuracy je Fold (gestrichelt: Mittelwert 0.776, Einbruch bei Fold 5). Rechts: aggregierte Out-of-Fold-Konfusionsmatrix über alle Folds.

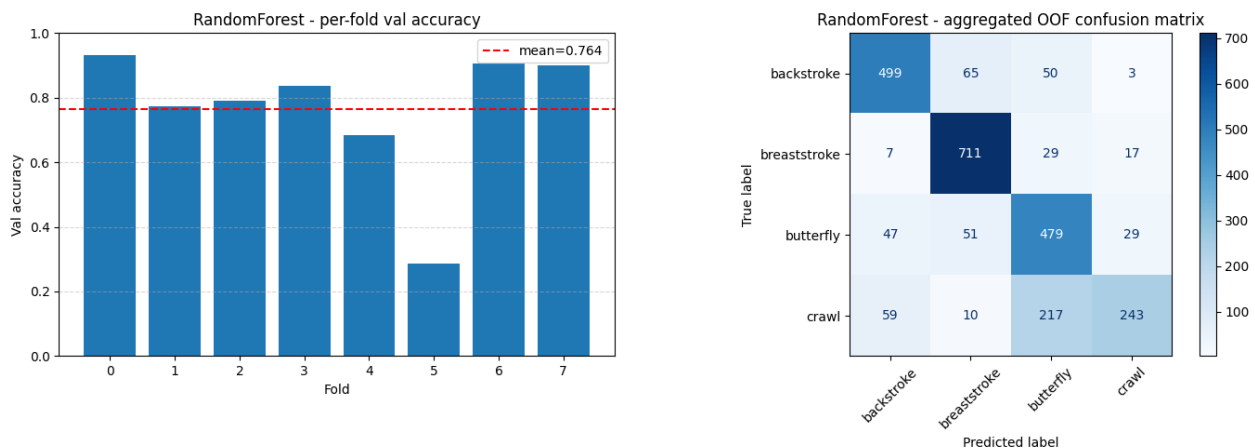


Abbildung 6: Random Forest-Kreuzvalidierung. Links: Validierungs-Accuracy je Fold (gestrichelt: Mittelwert 0.764, Einbruch bei Fold 5). Rechts: aggregierte Out-of-Fold-Konfusionsmatrix über alle Folds.

Es fällt auf dass bei beiden Modellen Fold 5 (Person S005) im Vergleich zu allen anderen Folds deutlich schlechter abschneidet. Bei beiden Modellen befinden sich alle restlichen Folds im Bereich des jeweiligen Validierungsmittelwerts. Einzig beim Random Forest schneidet Fold 4 noch leicht schlechter ab. Fold 0, 6 und 7 schneiden bei beiden Modellen leicht überdurchschnittlich ab. In der

aggregierten OOF Konfusionsmatrix wird ersichtlich, dass beide Modelle vor allem Mühe bei der Unterscheidung zwischen *Butterfly* und *Crawl* hatten, wenn *Crawl* klassifiziert werden musste.

5.3.4 Hyperparameter-Optimierung

Die Hyperparameter beider Modelle wurden mit OPTUNA optimiert, wobei als Zielgrösse die mittlere CV-Accuracy diente. Dabei wurde darauf geachtet, dass alle möglichen Parameterkombinationen berücksichtigt wurden und dementsprechende Trials durchgeführt.

Bei der Logistischen Regression wurden insgesamt 10 Trials, beim Random Forest 12 Trials durchgeführt. Die besten gefundenen Hyperparameter sind in Tabelle 7 und Tabelle 8.

Hyperparameter	Logistische Regression (tuned)
Maximale Iterationen	1000
Regularisierungsstärke	0.01
Klassen Gewichtung	balanced
CV-Accuracy (best)	0.788

Tabelle 7: Beste durch Optuna gefundene Hyperparameter der Logistischen Regression (10 Trials).

Hyperparameter	Random Forest (tuned)
Anzahl Entscheidungsbäume	30
Kriterium	gini
Maximale Tiefe	8
Mindestproben pro Split	2
Mindestproben pro Blatt	3
Klassen Gewichtung	balanced
CV-Accuracy (best)	0.764

Tabelle 8: Beste durch Optuna gefundene Hyperparameter des Random Forests (12 Trials).

Mit den besten Hyperparametern wurden die Modelle über die subjektbasierte Kreuzvalidierung trainiert und deren CV Scores berechnet (Abbildung 7 und 8). Nun können alle 4 Modellevariationen miteinander verglichen werden (Tabelle 9).

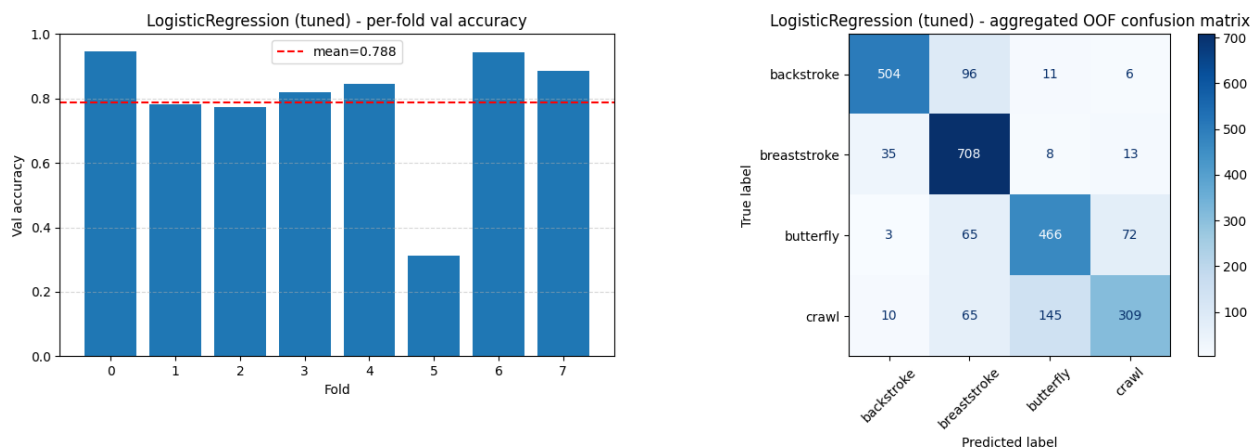


Abbildung 7: Logarithmische Regression (tuned)-Kreuzvalidierung. Links: Validierungs-Accuracy je Fold (gestrichelt: Mittelwert 0.788, Einbruch bei Fold 5). Rechts: aggregierte Out-of-Fold-Konfusionsmatrix über alle Folds.

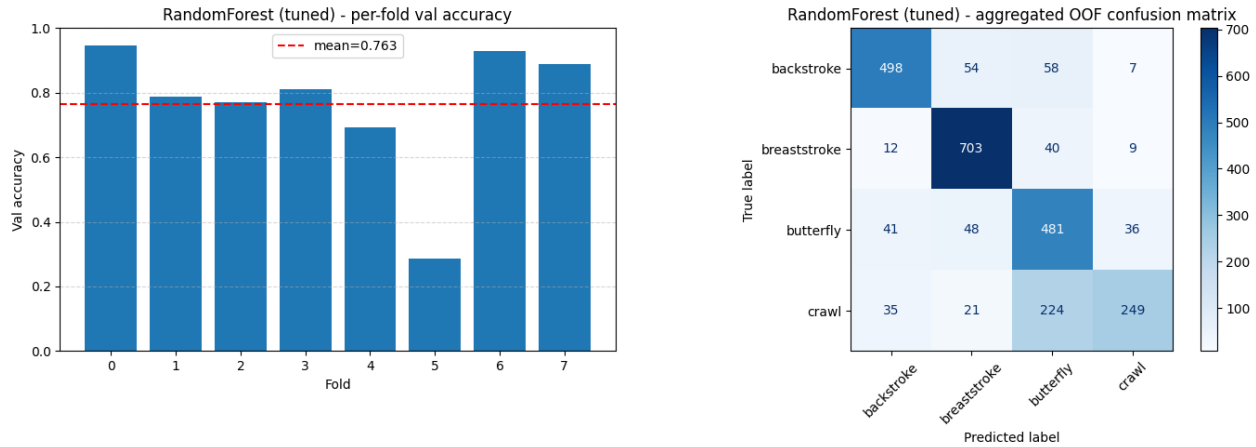


Abbildung 8: Random Forest(tuned)-Kreuzvalidierung. Links: Validierungs-Accuracy je Fold (gestrichelt: Mittelwert 0.761, Einbruch bei Fold 5). Rechts: aggregierte Out-of-Fold-Konfusionsmatrix über alle Folds.

Wie schon bei den nicht optimierten Modellen ist bei den optimierten Modellen zu sehen, dass der Fold 5 ein Ausreisser ist. Ausserdem haben auch diese beiden Modelle weiterhin eine Schwierigkeit den *Crawl* zu klassifizieren und weisen ihm fälschlicherweise relativ oft dem *Butterfly* zu.

Der Vergleich der 4 Modellvariationen wird anhand der Validierungs-Accuracy und des Macro F1 Werts über alle Folds sowie der Testdaten gemacht.

Modell	CV-Accuracy	CV-Macro-F1	Test-Accuracy	Test-Macro-F1
LogisticRegression (baseline)	0.776 $\pm$ 0.200	0.730 $\pm$ 0.246	0.897	0.897
LogisticRegression (tuned)	0.788 $\pm$ 0.204	0.746 $\pm$ 0.249	0.874	0.872
RandomForest (baseline)	0.764 $\pm$ 0.210	0.720 $\pm$ 0.245	0.897	0.900
RandomForest (tuned)	0.763 $\pm$ 0.211	0.719 $\pm$ 0.247	0.908	0.911

Tabelle 9: Kreuzvalidierungs-Ergebnisse und Test-Ergebnisse der Baseline-Modelle (Mittelwert $\pm$ Standardabweichung über acht Folds).

Während sich die Kreuzvalidierungs-Werte der logistischen Regression mit der Optimierung verbessern, bleiben die des Random Forest weitgehend unverändert. Dies lässt sich dadurch erklären, dass der Random Forest in seiner Baseline-Form bereits robust ist und Hyperparameter-Tuning nur geringen zusätzlichen Nutzen bringt. Die logistische Regression hingegen profitiert von optimierter Regularisierung, was zu höherer CV-Accuracy und CV-Macro-F1 führt.

Anders verhält es sich bei den Testdaten. Beim Random Forest verbessert sich das optimierte Modell erwartungsgemäss auf dem Testset (0.897 $\rightarrow$ 0.908 Accuracy), was auf eine erfolgreiche Generalisierung hindeutet. Bei der logistischen Regression zeigt sich jedoch ein paradoxe Situation: Das optimierte Modell schneidet auf dem Testset schlechter ab als die Baseline (0.874 vs. 0.897 Accuracy), obwohl die CV-Werte besser sind. Diese Phänomen könnte daran liegen, dass der Testdatensatz relativ klein ist (pro Schwimmstil nur eine Aufnahme auf einer kürzeren Bahn). Eine einzelne korrekte oder falsche Vorhersage kann die Accuracy und den Macro-F1 deutlich verschieben.

5.3.5 Merkmalswichtigkeit (Permutationsanalyse)

Um herauszufinden welche Aggregation und welche Sensorgruppe für die Entscheidung der trainierten Modelle relevant sind, wird mit einer Permutationsanalyse gearbeitet. Pro Aggregation und Sensorfamilie werden die zugehörigen Kanäle im Testdatensatz permutiert und der daraus hervorgehende Accuracy-Verlust gemessen (Abbildung 9).

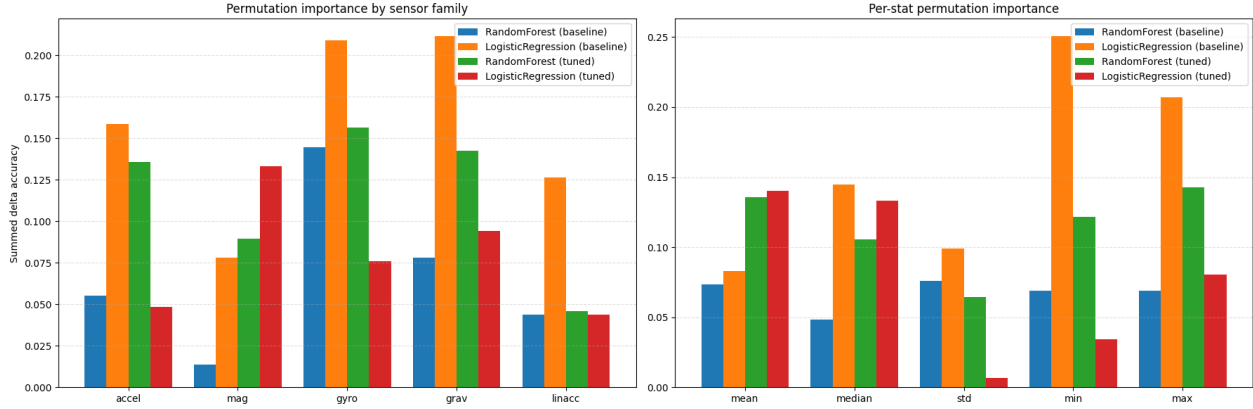


Abbildung 9: Permutations-Wichtigkeit pro Sensorfamilie und Aggregation (summierter Accuracy-Verlust beim Permutieren der jeweiligen Kanäle).

Bei den Sensorfamilien ist allgemein zu sagen, dass das Gyroskop der wichtigste Sensor ist. Das kann daran liegen, dass die Schwimmbewegung durch Armzug- und Rotationsdynamik geprägt ist. Die Analyse der Aggregation zeigt, dass die Merkmalswichtigkeit je nach Modell variiert. Die Unterschiede zwischen Baseline und getuntem Modell sind bei der logistischen Regression deutlich ausgeprägter. Die beiden Random Forest Modelle sind hingegen relativ ähnlich. Die Modelle reagieren also bei den Sensoren auf dynamische Bewegungsmerkmale was bei der Klassifizierung von Schwimmstilen Sinn macht.

Das **Gyroskop** (gyro) verursacht bei der logistischen Regression, im Baseline und getunten Modell, bei der Permutation den grössten Leistungsabfall. Es wird beim Random Forest ebenfalls stark genutzt, die Information werden aber etwas breiter auf mehrere Sensorgurppen verteilt. Bei beiden Random Forest Modellen ist das **Magnetometer** (mag) kaum relevant, bei der logistischen Regression gewinnt es aber besonders bei getunten Modell an Relevanz. Im Mittelfeld liegen die Sensorenfamilien **Linear Acceleration** (linacc), **Gravity Vector** (grav) und **Accelerometer** (accel), sie liefern ergänzende Information für die Klassifikation. Beim Baseline-Modell der Logistischen Regression sind **Minimum** (min) und **Maximum** (max) besonders wichtig. Beim getunten Modell fallen beide jedoch hinter den **Mittelwert** (mean) und den **Median** (median). Die Random Forest Modelle sind insgesamt robuster gegenüber einzelnen Aggregationen, wobei die **Standardabweichung** (std) besonders relevant ist.

5.3.6 Herausforderungen

Anfangs wurde für das baumbasierte Modell zunächst ein Decision Tree verwendet. Da jedoch die Fenster-Schrittweite erhöht wurde und daraus ein kleinerer Datensatz resultierte, kam es beim Baum zu starkem Overfitting. Dadurch nahm die Modellgüte deutlich ab. Um diesem Problem entgegenzuwirken, wurde ein Random Forest eingesetzt. Dieser ist robuster gegenüber Overfitting und konnte auch mit dem kleineren Datensatz eine gute Modellgüte erreichen.

5.4 Deep-Learning-Verfahren

Für das Deep-Learning-Verfahren wurden ein eindimensionales Convolutional Neural Network (1D-CNN) und ein rekurrentes Netz auf Basis von Long-Short-Term-Memory-Zellen (LSTM) umgesetzt. Beide Modelle wurden auf identischen Eingabedaten, mit derselben subjektbasierten Kreuzvalidierung und demselben Tuning-Verfahren trainiert, um einen fairen Vergleich zu gewährleisten (vgl. Shperkin, 2025).³

5.4.1 Datenaufbereitung und Skalierung

Jedes Fenster der allgemeinen Datenaufbereitung bildet einen Tensor der Form (100, 15), der direkt als Eingabe für beide Netze dient.

Die Skalierung erfolgt kanalbewusst, da die einzelnen Sensorgruppen unterschiedliche Wertebereiche und Ausreissercharakteristiken aufweisen: Für Beschleunigung, Magnetometer, Gyroskop und lineare Beschleunigung wird ein **RobustScaler** verwendet (unempfindlich gegenüber Ausreissern), während die Gravitationskanäle mit einem **StandardScaler** normiert werden. Die Skalierungsparameter werden ausschliesslich auf den Trainingsdaten geschätzt und beim Export fest in den ONNX-Graphen integriert, sodass die exportierten Modelle direkt rohe Fenster der Form (batch, 100, 15) entgegennehmen.

5.4.2 Modellarchitekturen

Das **1D-CNN** verarbeitet die Zeitreihe entlang der Zeitachse mit mehreren Faltungsblöcken (Convolution, Aktivierung, Pooling), gefolgt von einer globalen Aggregation und einem vollverbundenen Klassifikationskopf mit Dropout-Regularisierung. Faltungen erfassen lokale Bewegungsmuster (z. B. den charakteristischen Armzug-Rhythmus eines Stils) translation invariant über das Fenster hinweg.

Das **LSTM** modelliert die Sequenz hingegen explizit zeitlich-sequenziell: Die rekurrenten Zellen propagieren einen Zustand über die 100 Zeitschritte und erfassen so längerfristige Abhängigkeiten innerhalb eines Fensters. Der finale verborgene Zustand wird über einen vollverbundenen Kopf auf die vier Klassen abgebildet.

Die schematischen Architekturen beider Netze sind in Abbildung 10 dargestellt (links das CNN, rechts das LSTM; Tensorformen jeweils rechts neben den Blöcken, B = Batch-Grösse). Das CNN besteht aus zwei Faltungsblöcken (32 bzw. 64 Filter, Kernel 3, Padding 1 über 15 Eingangskanäle) mit ReLU und Max-Pooling, gefolgt von zwei vollverbundenen Schichten ($1600 \rightarrow 128 \rightarrow 4$). Das LSTM stapelt mehrere rekurrente Schichten (Input = 15, Hidden = 64) und bildet den letzten Zeitschritt über Dropout und einen linearen Kopf ($64 \rightarrow 4$) ab.

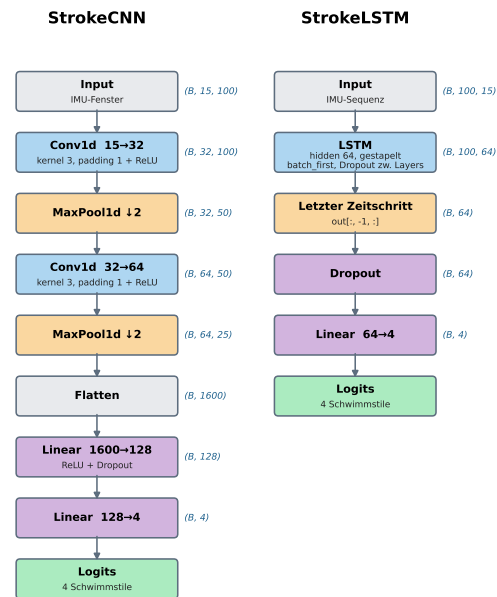


Abbildung 10: Schematische Architekturen der beiden Deep-Learning-Modelle. Links das StrokeCNN, rechts das StrokeLSTM. Tensorformen jeweils rechts neben den Blöcken.

³Sämtliche in diesem Abschnitt berichteten Ergebnisse – Training, Kreuzvalidierung, Hyperparameter-Tuning und Permutationsanalyse – wurden mit dem Notebook `DEL_Models.html` erzeugt.

5.4.3 Ergebniss der Subjektbasierten Kreuzvalidierung

Auffällig ist, dass Fold 5 (Person S005) über alle Modelle hinweg kollabiert (Validierungs-Accuracy um 0.28). Dieser Ausreisser senkt den CV-Mittelwert deutlich und erhöht die Standardabweichung; er ist auf datenseitige Besonderheiten dieser Aufnahme und nicht auf das Modell zurückzuführen. Die übrigen sieben Folds liegen durchgehend im Bereich von rund 0.89 bis 0.98. Abbildung 11 veranschaulicht beides: die fold-weise Validierungs-Accuracy (inkl. des Einbruchs bei Fold 5) sowie die aggregierte Out-of-Fold-Konfusionsmatrix, deren Fehler vor allem zwischen Brust, Delfin und Kraul auftreten.

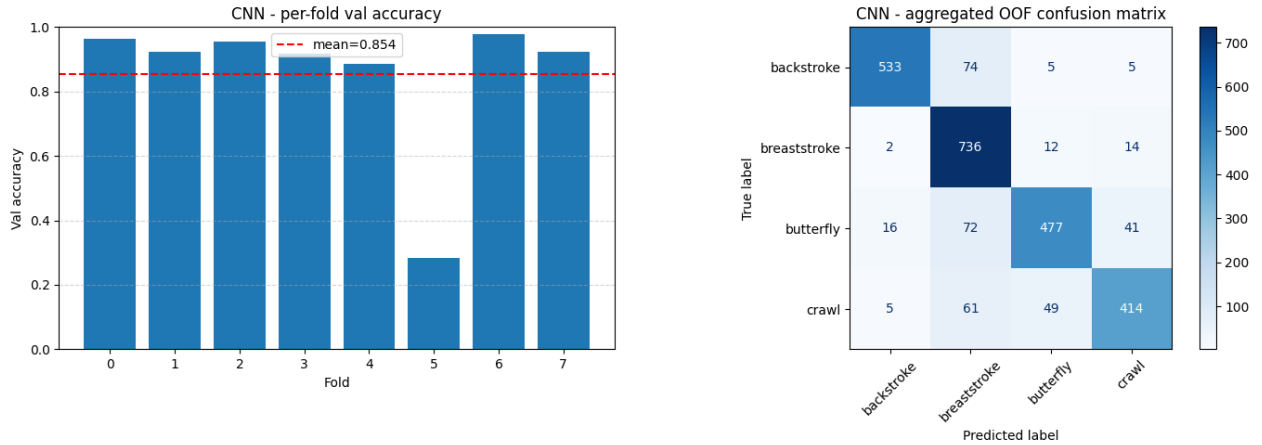


Abbildung 11: CNN-Kreuzvalidierung. Links: Validierungs-Accuracy je Fold (gestrichelt: Mittelwert 0.854, Einbruch bei Fold 5). Rechts: aggregierte Out-of-Fold-Konfusionsmatrix über alle Folds.

5.4.4 Hyperparameter-Optimierung

Die Hyperparameter beider Architekturen wurden mit OPTUNA über jeweils 20 Trials optimiert, wobei als Zielgrösse die mittlere CV-Accuracy diente. Pro Trial wurde die vollständige Acht-Fold-Kreuzvalidierung durchlaufen. Die besten gefundenen Konfigurationen sind in Tabelle 10 zusammengefasst.

Hyperparameter	CNN (tuned)	LSTM (tuned)
Lernrate	$3.19 \cdot 10^{-4}$	$2.95 \cdot 10^{-3}$
Weight Decay	$1.12 \cdot 10^{-4}$	$1.86 \cdot 10^{-7}$
Batch-Grösse	32	256
Filter / Hidden	128 Filter	64 Units
Kernel / Layers	Kernel 3	3 Layer
Dropout	0.33	0.22
CV-Accuracy (best)	0.870	0.873

Tabelle 10: Beste durch Optuna gefundene Hyperparameter (20 Trials je Modell).

Anschliessend wurden die besten Konfigurationen mit voller Epochenzahl (80 Epochen) neu trainiert. Bewusste Designentscheidungen waren der Verzicht auf Early Stopping, das Beibehalten einer unangetasteten Testmenge sowie der Einsatz von `ReduceLROnPlateau` zur Lernratensteuerung. Tabelle 11 zeigt die CV-Kennzahlen der vier resultierenden Modellvarianten.

Modell	CV-Accuracy	CV-Macro-F1
CNN (Baseline)	0.854 ± 0.232	0.825 ± 0.290
CNN (tuned)	0.855 ± 0.231	0.828 ± 0.291
LSTM (Baseline)	0.788 ± 0.211	0.755 ± 0.268
LSTM (tuned)	0.858 ± 0.221	0.842 ± 0.254

Tabelle 11: Kreuzvalidierungs-Ergebnisse der Deep-Learning-Modelle (Mittelwert $\pm$ Standardabweichung über acht Folds).

Der Vergleich von CV-Mittelwert und Held-out-Test-Accuracy (Abbildung 12) zeigt, dass die Test-Accuracy durchweg über dem CV-Mittel liegt. Dies ist eine Folge des Fold-5-Ausreissers, der den CV-Mittelwert nach unten zieht, sowie der kleinen, „gutartigen“ Testperson; die grossen CV-Fehlerbalken spiegeln dieselbe Streuung wider.

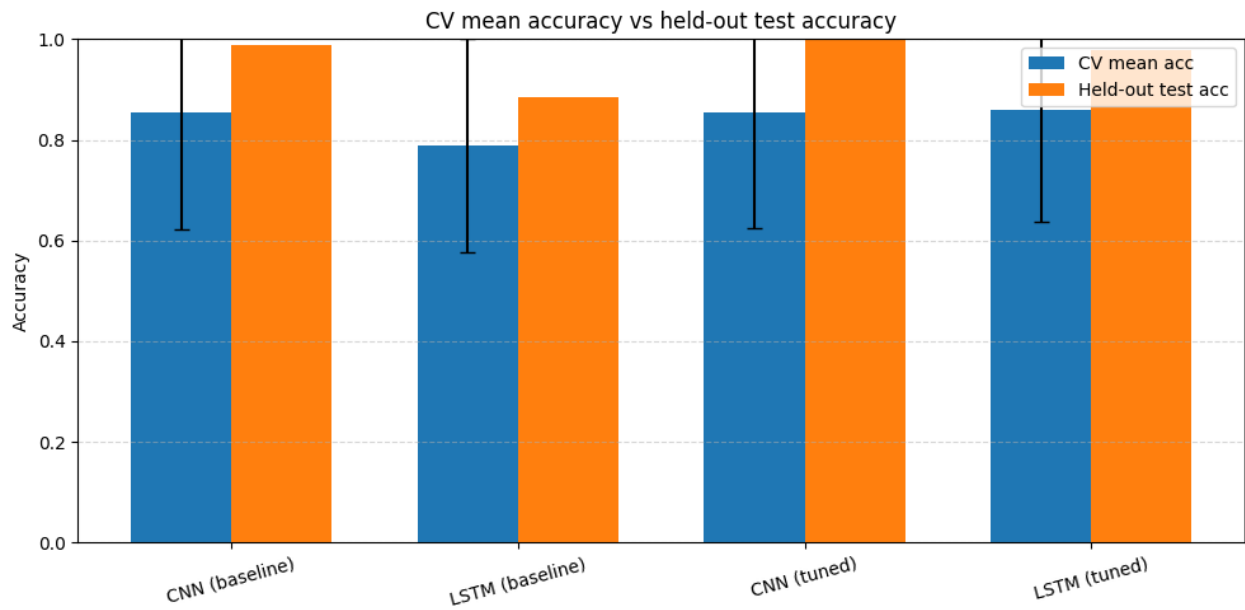


Abbildung 12: CV-Mittel-Accuracy (mit Standardabweichung als Fehlerbalken) gegenüber der Accuracy auf der ausgelagerten Testperson für die vier Deep-Learning-Varianten.

5.4.5 Merkmalswichtigkeit (Permutationsanalyse)

Abschliessend wird untersucht, welche Sensorgruppen für die Entscheidung der trainierten Modelle relevant sind. Dazu werden in einer Permutationsanalyse pro Sensorfamilie die zugehörigen Kanäle im Testdatensatz permutiert und der resultierende Accuracy-Verlust gemessen (Abbildung 13). Über alle Modelle hinweg tragen die Gravitations- und Gyroskop-Kanäle am stärksten zur Klassifikation bei, während das Magnetometer den geringsten Beitrag liefert.

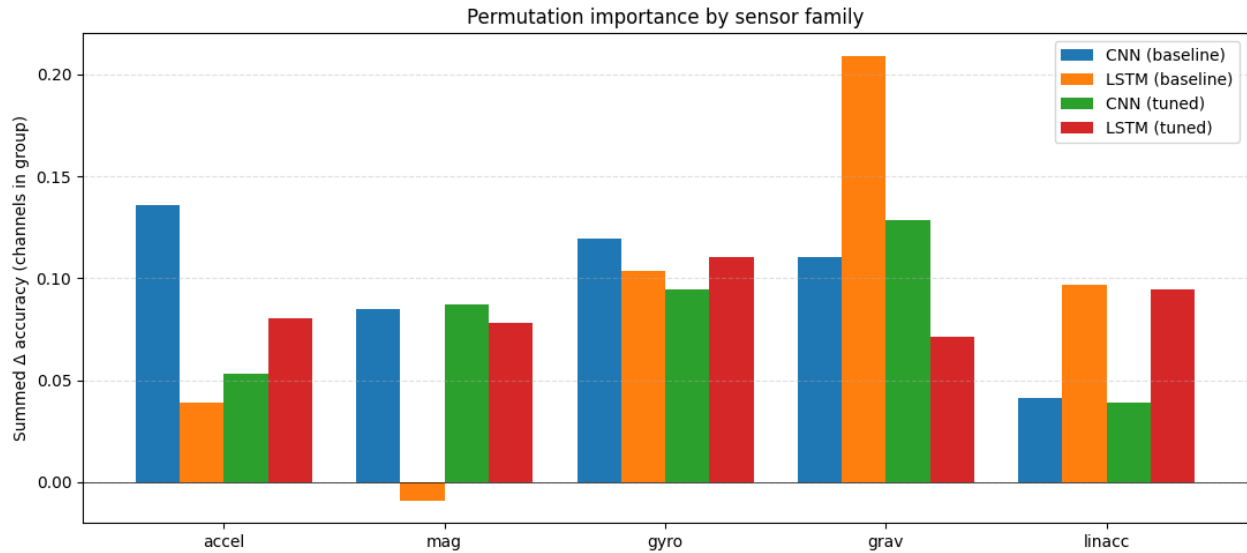


Abbildung 13: Permutations-Wichtigkeit pro Sensorfamilie (summierter Accuracy-Verlust beim Permutieren der jeweiligen Kanäle). Gravitation und Gyroskop sind die informativsten Sensorgruppen.

6 Evaluation und Resultate

Die Evaluation vergleicht alle Modelle direkt auf den exportierten ONNX-Artefakten. Einbezogen sind die vier Deep-Learning-Varianten aus Abschnitt 5.4 sowie die klassischen ML-Modelle (RandomForest und Logistische Regression, jeweils als Baseline- und Tuned-Variante).⁴

6.1 ONNX-Export und einheitliche Schnittstelle

Alle Modelle werden als eigenständige ONNX-Dateien exportiert – die vier Deep-Learning-Varianten ebenso wie die klassischen ML-Modelle. In jeden Graphen ist der Skalierer fest integriert, und eine gemeinsame `labels.json` hält die Klassenreihenfolge [`backstroke`, `breaststroke`, `butterfly`, `crawl`] fest. Damit ist das Inferenzformat vollständig vom Trainingscode entkoppelt: Die Graphen nehmen rohe Fenster der Form $(batch, 100, 15)$ direkt entgegen. So liefert jeder Klassifikator über dieselbe Logit-Schnittstelle seine Fenster-Wahrscheinlichkeiten. Erst diese Vereinheitlichung macht den fairen, modellübergreifenden Vergleich in den folgenden Abschnitten möglich.

6.2 Aggregation auf Bahn-Ebene: Soft- und Hard-Voting

Innerhalb einer Bahn wechselt eine schwimmende Person ihren Stil nicht; jeder Chunk (eine Bahn) gehört daher zu genau einer Klasse. Die fensterweisen Vorhersagen werden deshalb zu einer Bahn-Entscheidung zusammengefasst. Eine Bahn bestehe aus n Fenstern mit den Softmax-Verteilungen $\mathbf{p}_i \in \mathbb{R}^K$ ($i = 1, \dots, n$; $K = 4$ Klassen). Dafür kommen zwei Aggregationsregeln in Frage.

Beim **Soft-Voting** (kanonische Regel) wird zunächst die mittlere Verteilung über alle Fenster gebildet und anschliessend deren Maximum gewählt:

$$\bar{\mathbf{p}} = \frac{1}{n} \sum_{i=1}^n \mathbf{p}_i, \quad \hat{y}_{\text{soft}} = \arg \max_k \bar{p}_k, \quad c_{\text{soft}} = \max_k \bar{p}_k. \quad (1)$$

Die Konfidenz c_{soft} ist also die mittlere Gewinner-Wahrscheinlichkeit – ein Mass dafür, wie „scharf“ die gemittelte Verteilung ausfällt.

Beim **Hard-Voting** stimmt jedes Fenster mit genau einer Stimme für seine Argmax-Klasse ab; gewählt wird die Klasse mit den meisten Stimmen:

$$c_k = |\{i : \arg \max_j p_{i,j} = k\}|, \quad \hat{y}_{\text{hard}} = \arg \max_k c_k, \quad c_{\text{hard}} = \frac{c_{\hat{y}_{\text{hard}}}}{n}. \quad (2)$$

Die Konfidenz c_{hard} ist hier der Stimmenanteil der Gewinnerklasse – also ein Mass für die Übereinstimmung der Fenster.

Beide Regeln nutzen dieselbe fensterweise Softmax-Matrix und unterscheiden sich nur darin, wann das Argmax gebildet wird: Soft-Voting mittelt zuerst und wählt dann, Hard-Voting wählt zuerst pro Fenster und zählt dann. Daraus ergeben sich unterschiedliche Stärken. Soft-Voting behält die vollständige Wahrscheinlichkeitsinformation und bildet unsichere Fenster feiner ab. Hard-Voting reduziert jedes Fenster auf eine Stimme und ist dadurch robuster gegenüber einzelnen, überkonfident falsch klassifizierten Fenstern. Auf der vorliegenden Testmenge erreichen ohnehin beide Regeln die maximale Bahn-Genauigkeit (1.000).

Das Soft-Voting hat einen weiteren Vorteil: Es erhält die fensterweisen Wahrscheinlichkeitsverläufe entlang der Zeitachse. Ein Schwimmstil-Wechsel innerhalb einer Bahn würde sich darin als zeitliche Verschiebung der dominanten Klasse zeigen. Damit eignet sich der Soft-Ansatz grundsätzlich, solche

⁴Die modellübergreifende Evaluation (Kennzahlen, Rankings, Robustheitstest und Abbildungen) wurde im Notebook `ModelEvaluation.ipynb` durchgeführt. Die ausgewerteten ONNX-Modelle stammen aus `DEL_Models.ipynb` (Deep Learning) bzw. `ML_Models.ipynb` (klassische Verfahren).

Wechsel zu erkennen. Das geht über die hier getroffene Annahme „eine Bahn entspricht genau einem Stil“ hinaus und ist ein lohnender Ansatzpunkt für zukünftige Arbeiten.

6.3 Testaufbau

Als Testmenge dient die subjektbasiert ausgelagerte Aufnahme der Person T001: vier Bahn-Chunks (je ein Schwimmstil) mit insgesamt 87 Fenstern. Berichtet werden Fenster-Accuracy und Fenster-Macro-F1 sowie die Bahn-Accuracy unter beiden Aggregationsregeln (Soft- und Hard-Voting). Zusätzlich prüfen wir die Robustheit bei reduzierter Abtastrate (50 Hz); das Vorgehen beschreibt Abschnitt 6.8.

6.4 Gesamtranking

Unter Soft-Voting erreichen *alle* acht Modelle die maximale Bahn-Accuracy (1.000): Jedes Modell klassifiziert jede der vier Bahnen korrekt. Zur Unterscheidung der Modelle taugt diese Kennzahl deshalb nicht. Primäres Ranking-Kriterium ist daher die **Fenster-Macro-F1**. Bei Gleichstand entscheiden die Robustheit bei 50 Hz und die Hard-Vote-Konsistenz. Tabelle 12 zeigt das Gesamtranking.

Tabelle 12: Gesamtranking aller Modelle auf der Testmenge (Person T001). Bahn-Accuracy (Soft-Vote) ist für alle Modelle 1.000; das Ranking basiert auf der Fenster-Macro-F1.

Rang	Modell	Fenster-Acc	Fenster-F1
1	CNN (tuned)	1.000	1.000
2	CNN (Baseline)	0.989	0.990
3	LSTM (tuned)	0.977	0.977
4	RandomForest (tuned)	0.908	0.911
5	RandomForest	0.897	0.900
6	LogReg	0.897	0.897
7	LSTM (Baseline)	0.885	0.885
8	LogReg (tuned)	0.874	0.872

Die Convolutional-Netze führen das Feld klar an: Das CNN (tuned) erreicht auf Fensterebene perfekte 1.000, gefolgt von CNN (Baseline) und LSTM (tuned). Die klassischen ML-Modelle liegen zwischen 0.87 und 0.91 Macro-F1. Auf Bahn-Ebene verschwinden diese Unterschiede jedoch vollständig: Das Soft-Voting mittelt fensterweise Fehler aus, sodass schon das schwächste Modell jede Bahn fehlerfrei klassifiziert.

6.5 Konfidenz

Da die Bahn-Accuracy gesättigt ist, wird die Konfidenz zum eigentlichen Unterscheidungsmerkmal. Wir berichten die in Abschnitt 6.2 definierten Grössen c_{soft} (Gleichung 1) und c_{hard} (Gleichung 2). Die Deep-Learning-Modelle erzeugen deutlich „schärfere“ Verteilungen als die ML-Modelle: Die Soft-Konfidenz auf Bahn-Ebene liegt beim CNN (tuned) bei rund 0.973, beim RandomForest dagegen nur bei etwa 0.67 und bei der Logistischen Regression bei rund 0.79. Eine höhere Konfidenz bedeutet einen grösseren Sicherheitsabstand – und damit mehr Reserve gegenüber verrauschten oder untypischen Fenstern.

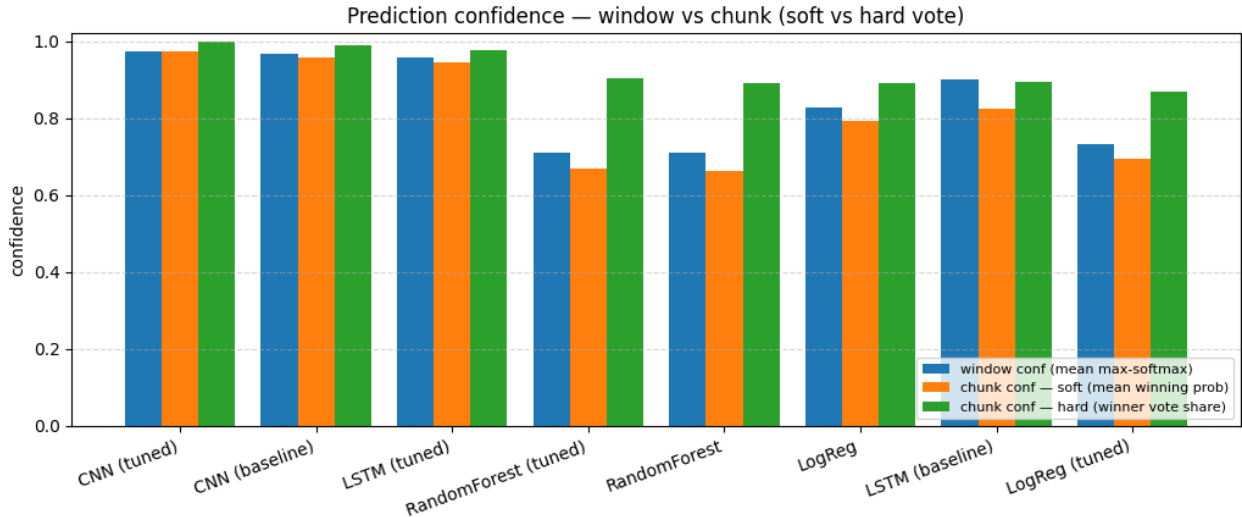


Abbildung 14: Vorhersage-Konfidenz pro Modell: Fenster-Konfidenz (mittleres Max-Softmax) sowie Bahn-Konfidenz unter Soft- und Hard-Voting. Die Deep-Learning-Modelle sind deutlich „schärfer“ als die ML-Modelle.

6.6 Per-Chunk-Stimmverhalten

Abbildung 15 fasst das Abstimmungsverhalten aller acht Modelle auf den vier Test-Bahnen zusammen und verbindet so die beiden vorangehenden Befunde. Pro Bahn zeigt der obere Streifen die gestapelten Per-Fenster-Wahrscheinlichkeiten – die Grundlage des Soft-Votings. Der untere Streifen zeigt das Hard-Vote-Urteil je Stride-Cell. Alle Bahn-Urteile sind korrekt (Häkchen); das veranschaulicht die gesättigte Bahn-Genauigkeit aus Tabelle 12 direkt.

Zugleich tritt der Konfidenzunterschied aus Abschnitt 6.5 klar hervor: Die Deep-Learning-Modelle – vor allem CNN (tuned) und LSTM (tuned) – erzeugen nahezu einfarbige Stapel, in denen die korrekte Klasse praktisch jedes Fenster dominiert. Bei den ML-Modellen, insbesondere RandomForest und LogReg, sind die Stapel deutlich gemischter und enthalten sichtbare Anteile konkurrierender Klassen. Die Mehrheit – und damit das Bahn-Urteil – bleibt aber korrekt. Einzelne falsch klassifizierte Fenster, etwa im Kraul-Chunk, werden durch die Aggregation zuverlässig überstimmt. Das erklärt anschaulich, warum die Bahn-Genauigkeit trotz unvollkommener Fensterklassifikation gesättigt bleibt.

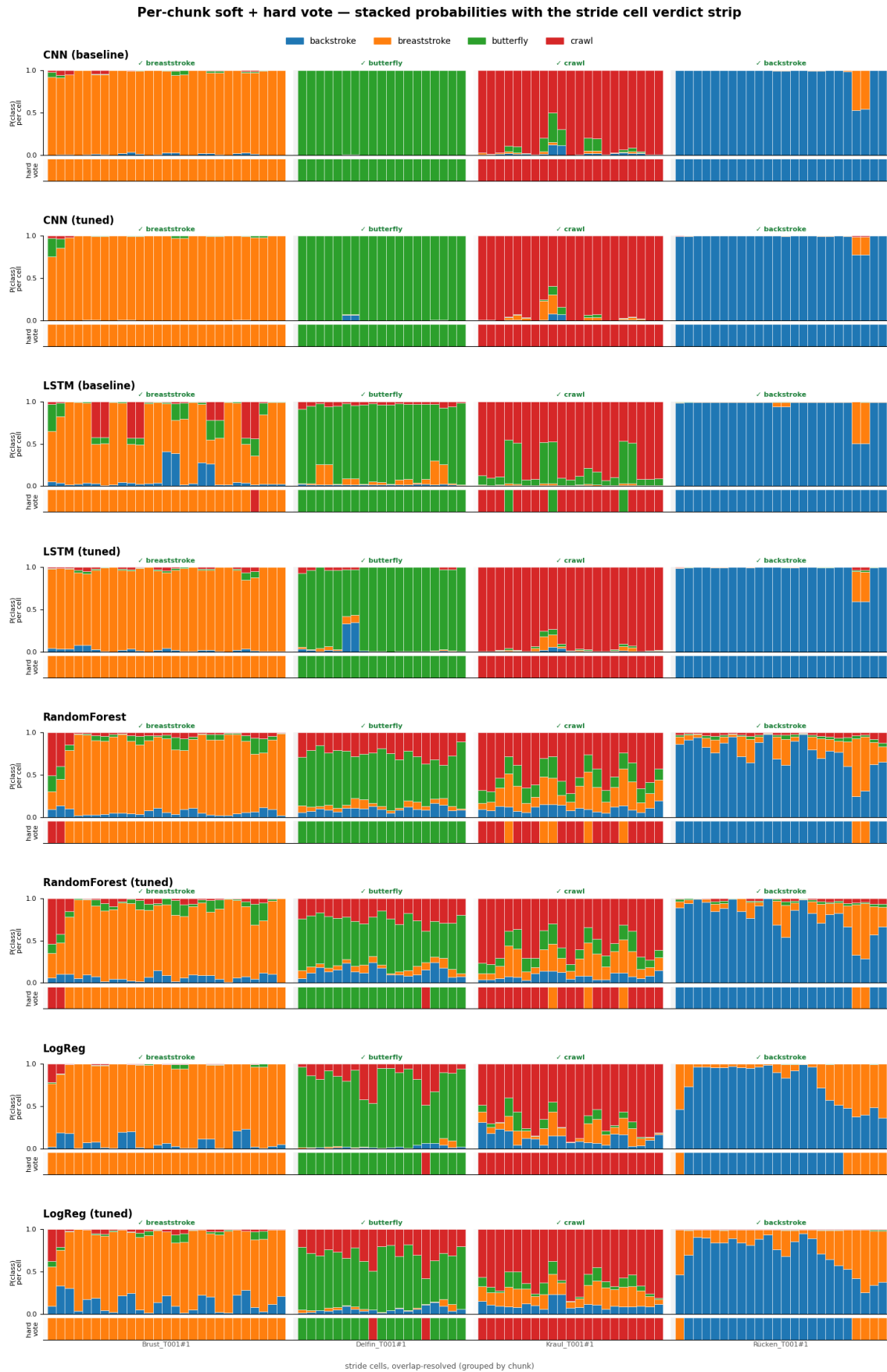


Abbildung 15: Per-Chunk-Abstimmung aller Modelle auf den vier Test-Bahnen. Je Modell und Bahn: oben die gestapelten Per-Fenster-Wahrscheinlichkeiten (Soft-Vote-Grundlage), darunter der Hard-Vote-Verdict-Streifen pro Stride-Cell. Alle Bahn-Urteile sind korrekt; die Deep-Learning-Modelle stapeln nahezu einfarbig, die ML-Modelle deutlich gemischter.

6.7 Auswertung pro Schwimmstil

Abbildung 16 zeigt die Fenster-Macro-F1 pro Schwimmstil und Modell bei 100 Hz. Das CNN (tuned) ist in jeder der vier Klassen das beste Modell und erreicht überall 1.000. Die übrigen Modelle sind bei Brust und Kraul am schwächsten, wo die ML-Modelle teils unter 0.85 fallen. Rücken und Delfin werden dagegen durchgehend zuverlässiger erkannt.

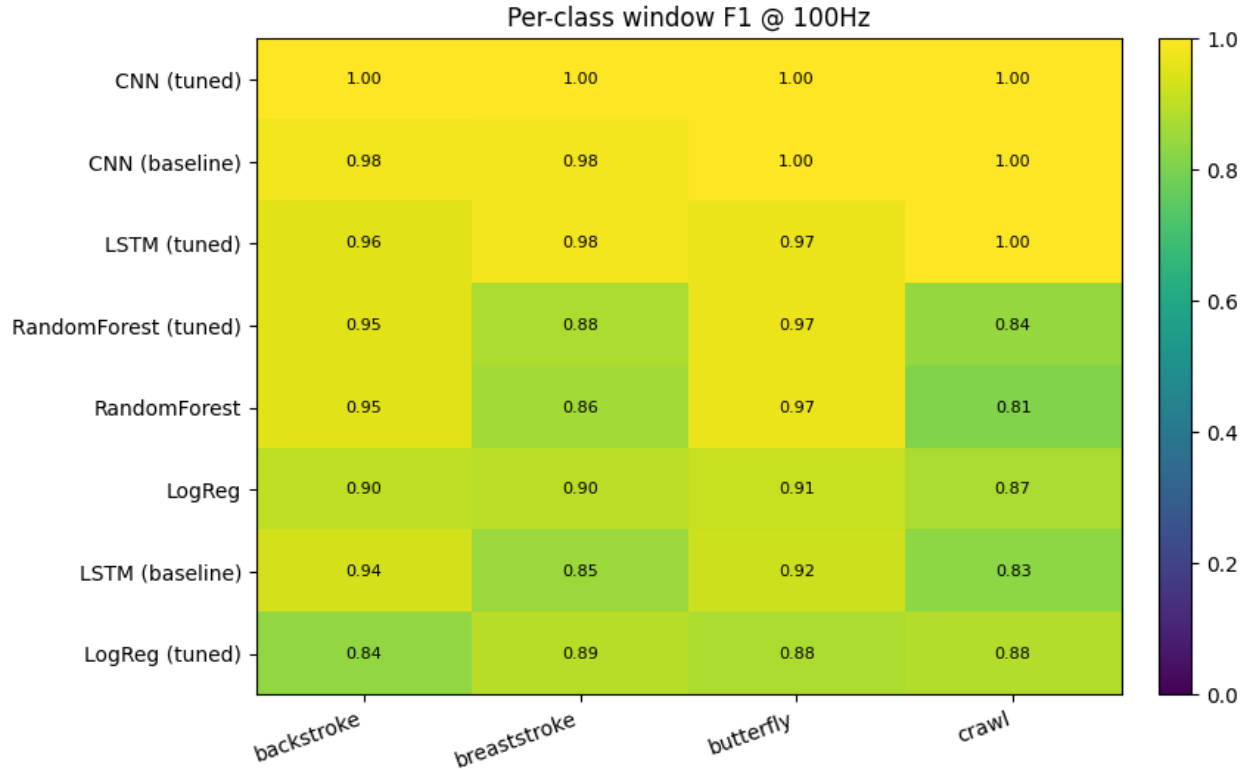


Abbildung 16: Fenster-Macro-F1 pro Schwimmstil und Modell bei 100 Hz. Kraul und Brust sind für die ML-Modelle am schwierigsten, das CNN (tuned) erreicht durchgehend 1.00.

6.8 Robustheit bei reduzierter Abtastrate

Der 50 Hz-Test misst, wie stark die Leistung bei halbierter Abtastrate einbricht. Die *Retention* ist dabei das Verhältnis der Fenster-Macro-F1 bei 50 Hz zu jener bei 100 Hz. Um die niedrigere Abtastrate ohne Neutraining nachzubilden, wird jedes Fenster zunächst auf 50 Hz dezimiert und anschliessend per *linearer Interpolation* wieder auf die ursprüngliche Länge (*seq_len* = 100 Abtastwerte) gebracht. So bleibt der feste Graph-Eingang der Form (*batch*, 100, 15) erhalten, und derselbe ONNX-Graph lässt sich ohne Anpassung weiterverwenden – verändert wird allein der Informationsgehalt innerhalb des Fensters. Tabelle 13 fasst die Ergebnisse zusammen. Die Bahn-Accuracy (Hard-Vote) bleibt bei allen Modellen auch bei 50 Hz bei 1.000.

Die Deep-Learning-Modelle und die Logistische Regression sind praktisch verlustfrei robust (*Retention* $\approx$ 1.000). Nur die RandomForest-Varianten verlieren spürbar an Fensterleistung (*Retention* 0.97–0.99); das liegt an ihrer stärkeren Abhängigkeit von feinkörnigen, frequenzabhängigen Fenster-Statistiken. Auf Bahn-Ebene bleibt die Klassifikation aber auch hier fehlerfrei.

Tabelle 13: Robustheit bei 50 Hz. Retention = Fenster-Macro-F1₅₀ / Fenster-Macro-F1₁₀₀.

Modell	F1 ₁₀₀	F1 ₅₀	Retention
CNN (tuned)	1.000	1.000	1.000
CNN (Baseline)	0.990	0.990	1.000
LSTM (tuned)	0.977	0.977	1.000
LogReg	0.897	0.897	1.000
LSTM (Baseline)	0.885	0.885	1.000
LogReg (tuned)	0.872	0.872	1.000
RandomForest	0.900	0.887	0.986
RandomForest (tuned)	0.911	0.887	0.973

6.9 Fazit

Das CNN (tuned) ist über alle Kriterien hinweg das stärkste Modell: perfekte Fenster- und Bahn-Klassifikation, höchste Konfidenz und volle Robustheit bei 50 Hz. Auf Bahn-Ebene erkennen allerdings alle Modelle den Stil fehlerfrei; die aussagekräftigeren Vergleichsgrößen sind daher die Fenster-Macro-F1 und die Konfidenz. Insgesamt zeigt die Evaluation, dass die Kombination aus kanalbewusster Skalierung, gleitender Fensterung und Soft-Voting eine robuste und gut generalisierende Schwimmstil-Klassifikation auf bislang ungesehene Personen ermöglicht. Eine Einschränkung bleibt die kleine, subjektbasierte Testmenge (eine Person, vier Bahnen); künftige Arbeiten sollten die Ergebnisse an mehr Testpersonen absichern. Da das Soft-Voting die fensterweisen Wahrscheinlichkeitsverläufe erhält, bietet es ausserdem einen natürlichen Ansatzpunkt, um künftig Schwimmstil-Wechsel innerhalb einer Bahn zu erkennen.

7 Diskussion

Die vorangehenden Abschnitte haben die einzelnen Bausteine getrennt bewertet. Abbildung 17 führt sie an einem Beispiel zusammen und zeigt die gesamte Verarbeitungskette für eine einzelne Bahn (Brust, Testperson T001) – vom Rohsignal bis zur fertigen Vorhersage.

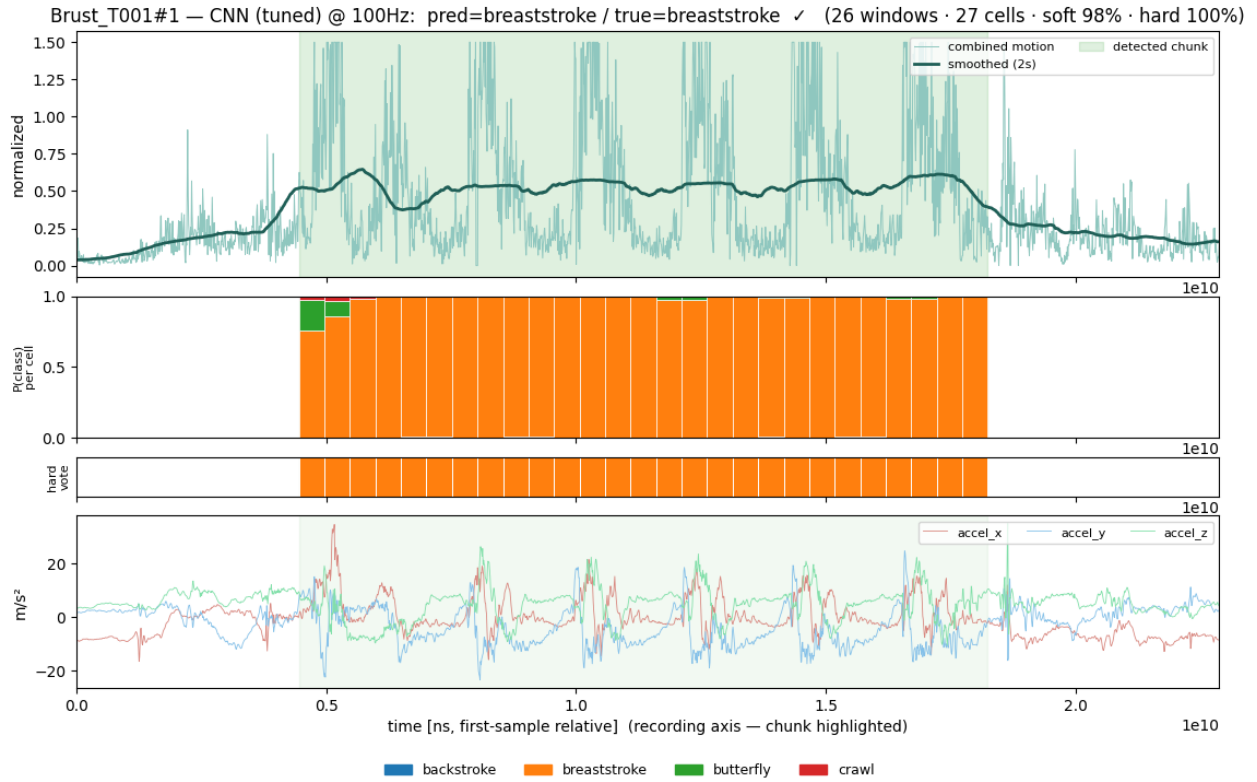


Abbildung 17: End-to-End-Verarbeitung einer Bahn (CNN (tuned), 100 Hz)

Die Kette durchläuft vier Stufen. Zuerst werden die fünf synchronisierten IMU-Sensoren zu einer kombinierten Bewegungsrate verrechnet und geglättet (oberstes Panel). Aus diesem Signal trennt ein Plateau-Detektor den eigentlichen Bahn-Abschnitt (grün) von Start, Wende und Auslauf ab – das ist die Chunk-Extraktion. Innerhalb des Chunks wird das Signal anschliessend in überlappende Fenster zerlegt ($seq_len = 100$, $step = 50$). Jedes Fenster durchläuft den exportierten ONNX-Graphen und liefert eine Klassenwahrscheinlichkeit; das zweite Panel zeigt diese Werte gestapelt pro Stride-Cell, das dritte das daraus abgeleitete Hard-Vote. Im letzten Schritt aggregiert das Soft-Voting die Fenster zu einer einzigen Bahn-Entscheidung.

Das Beispiel macht das Zusammenspiel der Stufen anschaulich. Einzelne Fenster – typischerweise am Chunk-Rand oder während kurzer Gleitphasen – sind weniger eindeutig, doch die Aggregation mittelt diese Unsicherheiten aus und liefert eine klare, korrekte Bahn-Vorhersage. Genau dieser Mechanismus erklärt die durchgehend fehlerfreie Bahn-Klassifikation aus Abschnitt 6, obwohl die Fensterleistung der schwächeren Modelle darunter liegt.

Insgesamt zeigt sich, dass eine schlanke Kette aus Chunk-Erkennung, gleitender Fensterung, einheitlicher ONNX-Inferenz und Soft-Voting eine zuverlässige Schwimmstil-Klassifikation aus einem einzigen, am Handgelenk getragenen IMU ermöglicht – robust gegenüber einer ungesesehenen Person, einem anderen Becken und sogar einer halbierten Abtastrate. Die Grenzen liegen weniger im Modell als in den Daten: Die Testmenge ist klein (eine Person, vier Bahnen), und die Pipeline nimmt implizit an, dass jede Bahn genau einem Stil entspricht. Die Limitationen der Datenerfassung sind in Abschnitt 3.8 beschrieben. Da das Soft-Voting die fensterweisen Wahrscheinlichkeitsverläufe erhält (zweites Panel), bietet sich als nächster Schritt an, Stilwechsel innerhalb einer Bahn zu erkennen und die Ergebnisse an mehr Testpersonen abzusichern.

Literatur

- Bulling, A., Blanke, U., & Schiele, B. (2014). A Tutorial on Human Activity Recognition Using Body-Worn Inertial Sensors. *ACM Computing Surveys*, 46(3), 33:1–33:33. <https://doi.org/10.1145/2499621>
- Mooney, R., Corley, G., Godfrey, A., Quinlan, L. R., & OLaighin, G. (2016). Inertial Sensor Technology for Elite Swimming Performance Analysis: A Systematic Review. *Sensors*, 16(1), 18. <https://doi.org/10.3390/s16010018>
- Shperkin, M. (2025). *Multi-Task Deep Learning Approach for Segmenting and Classifying Competitive Swimming Activities Using a Single IMU*. [Magisterarb., University of South Carolina]. <https://scholarcommons.sc.edu/etd/8556>