Application Project Report

Machine Learning for Gait Event Detection & Pain Anamnesis Classification

Students: Hassan Rasheed, Jaeyeop Chung

Advisor: Lucas Heitele
Supervisor: Pietro Fantini
Examiner: Prof. Dr. Joachim Henkel

Dr. Theo Schöller-Stiftungslehrstuhl für Technologie und Innovationsmanagement

Technische Universität München

Contents

1 Introduction **6**

2 Data **7**

2.1 Gait Event Data 7

2.2 Pain Anamnesis Data 8

3 LSTM-Based Approach for Gait Event Detection and Pain Classification **9**

3.1 Gait Event Detection with LSTM 9

3.2 Pain Anamnesis Classification with a Multi-Output Neural Network 11

4 XGBoost-Based Approach **17**

4.1 XGBoost and Optuna 17

4.2 Gait Event Detection 19

4.2.1 Methodology 19

4.2.2 Experiments 22

4.2.3 Result 24

4.2.4 Summary 25

4.3 Anamnesis Analysis 25

4.3.1 Methodology 25

4.3.2 Experiment 28

4.3.3 Result 30

4.3.4 Summary 31

5 Discussion & Conclusion **32**

List of Figures

1 Data: Gait Event Cycle (Viktora, 2023) 37
2 Data: Gait Phase Distribution (Original Data) 37
3 Data: Anamnesis Data Feature Distribution 38
4 Data: Anamnesis Data Target Distribution 39
5 **Gait cycle with key events.** A typical human gait cycle, illustrating the sequence of gait events: Heel Strike (initial contact), Foot Flat (loading response), Heel Off (end of mid-stance), and Toe Off (pre-swing). The cycle then continues into the swing phase until the next heel strike. 39
6 **LSTM Architecture for Gait Event Detection.** Schematic of the 3-layer LSTM model used for gait event detection. The model takes as input a time series of IMU sensor readings (3-axis acceleration and 3-axis gyroscope). It processes the sequence through stacked LSTM layers (each with 128 units), and outputs class probabilities for each time step (No event, Heel strike, Foot flat, Heel off, Toe off). Dropout layers (not shown for clarity) were applied between LSTM layers during training for regularization. 40
7 **Confusion Matrix for Gait Event Detection.** The confusion matrix of the LSTM model's predictions on the test set. Rows represent true classes and columns represent predicted classes. Each cell shows the count of time steps of a given true class that were predicted as a given class. The matrix highlights that most confusion occurred by misclassifying some *Heel Strike* instances as *Foot Flat* (since Heel Strike events were often missed), while the other classes have high true positive counts on the diagonal. 40
8 **Pain Anamnesis Input Example.** An example interface (Eversion app) for collecting pain anamnesis data. Patients report pain levels across multiple body regions. Some fields are binary and others use a numeric (Likert) scale for pain severity (0 = no pain to 5 = extreme pain). In our dataset, 42 such outputs were recorded for each patient (covering various joints, limbs, and symptoms). 41
10 Gait Event-XGboost: Magnitudes of gyroscope data (left) and accelerometer data (right) collection ID : "GZvNrD141xdO59uOgra6_5_R". 41
9 **Multi-Task Neural Network for Pain Classification.** Diagram of the shared-trunk multi-output network used for pain anamnesis classification. The input gait feature vector (7 features) passes through shared dense layers (the trunk). From the trunk, two sets of outputs are produced: 18 sigmoid outputs for binary pain indicators (one per region, indicating pain presence), and 24 groups of ordinal outputs (5 sigmoid neurons per group, each group corresponding to a region's pain severity thresholds 1–5). The ordinal outputs use the CORAL approach to infer a 0–5 pain level. During training, losses from both binary and ordinal outputs are combined to update the shared trunk, allowing the model to learn a representation beneficial to both tasks. 42

11 Gait Event-XGboost: Autocorrelation function for one data collection ID (blue region: 95% confidence interval of correlation at a lag K) 42
12 Gait Event-XGboost: GroupShuffleSplit vs GroupKFold comparison[1](The main difference is that GroupShuffleSPlit allows the same group to be used multiple times while GroupKFold allows only the distinct groups to be used for each fold.) 43
13 Gait Event-XGboost(Grid Search): Accuracy distribution per collection ID 43
14 Gait Event-XGboost(Optuna): Accuracy distribution per collection ID 43
15 Gait Event-XGboost(Grid Search): actual vs predicted gait phases 44
16 Gait Event-XGboost(Optuna): actual vs predicted gait phases 44
17 Anamnensis-XGBoost: Converted Target Distribution (Binary Target) 45
18 Anamnensis-XGBoost: Standardized feature distribution (rows with missing values are excluded) 45
19 Anamnensis-XGBoost: Point-Biserial Correlation and MI between features and target values. Left to Right: Point-Biserial correlation, Distance correlation, Mutual Information (darker colors indicate stronger associations; lighter colors indicate values closer to zero). 46
20 Anamnensis-XGBoost: Cramer's V between target variables. Only the pairs with Cramer's V greater than 0.4 are colored 46
21 Anamnensis-XGBoost: SHAP Summary Plot (Each point in the plot represents a single data point from the feature variables. Red indicates high SHAP values, while blue indicates low SHAP values. The greater the SHAP value, the more influence that feature has on pushing the prediction in the direction shown along the X-axis.) 47
22 Anamnensis-XGBoost: Fanal model's predicted log-odds histogram with threshold for each target variables. (histograms in red are the data with actual true label(1) and the plots in green are the data with actual false label(0). The data on the right side of the thresholds(dashed line) are predicted as ture, and the data on the left side of the threshold are predicted zero. With validation accuracy as an object, the thresholds tend to be set with the higher values, increasing the false negative. The Optuna tried to find the balance between false negatives and false positives. 48
23 Anamnensis-XGBoost: SHAP dependence plot for Misalignment_Avg_left feature variables across all target variables. (red box: on the right side is the feature variable that has the largest interaction with Misalignment_Avg_left on the Pain_Knee_Left target variable. The colors in each point is Shapley values of the variable on the right side. 49

List of Tables

1 Data: Gait event raw dataset (508,446 total rows, 8 columns). 50
2 Data: Anamnesis feature summary statistics 50
3 Data: Anamensis Target variable distributions: (a) ordinal (non-binary) and (b) binary pain indicators. 50
4 **Gait Event Detection Performance per Class.** Classification report on the test set for the LSTM gait event detection model. Precision, recall, and F_1 score are given for each gait event class, along with the support (number of true instances of each class in the test data). The model excels on the frequent classes (Foot Flat, Toe Off) but struggled on the rare Heel Strike class, as evidenced by the 0% recall for Heel Strike (only 33 instances in test set). Overall accuracy (weighted by support) is 91%. . . . 51
5 **Pain Classification Results (Binary vs Multi-Task).** Comparison of performance between a model trained only on binary pain classification and the multi-task model (binary+ordinal) on the test set. The multi-task model achieves slightly higher accuracy on the binary pain predictions and additionally provides ordinal severity predictions (with an average MAE of 1.4). 51
6 Gait Event-XGboost: gait event dataset with lagged features (464,590 total rows, 116 columns). 51
7 Gait Event-XGboost: Predefined XGBoost hyperparameter range and hyperparameter grid for gait event detection 51
8 Gait Event-XGboost: performance summary on the test set (Grid Search) 52
9 Gait Event-XGboost: performance summary on the test set (Optuna) 52
10 Anamnesis-XGboost: Predefined XGBoost hyperparameter range and hyperparameter grid for anamnesis analysis 52
11 Anamnesis-XGBoost: Hyperparameter optimization results for pain classification models 52
12 Anamnesis-XGBoost: Final model results for pain classification models 53
13 English - Column name mapping (English column names are translated from the original German column names) 54

List of Appendices

A.1 Pain Anamnesis - XGBoost: column name English mapping 54

1. Introduction

Human gait is an intricate and repetitive biomechanical process characterized by distinct phases and events, notably **heel strike (HS)** and **toe-off (TO)**. Accurate detection of these gait events forms the cornerstone for comprehensive gait analysis, vital for assessing locomotion abnormalities, informing prosthetics calibration, and guiding rehabilitation interventions (X. Chen & Martin, 2025; Romijnders & et al., 2022). Traditionally, gait event detection has relied upon threshold-based heuristic methods, applying predetermined criteria to data from sensors like ground reaction force (GRF) plates or inertial measurement units (IMUs) (Vu et al., 2020). These traditional techniques, though straightforward and computationally inexpensive, often require precise calibration and sensor placement, and they typically underperform in scenarios involving pathological or atypical gait patterns (Zhen & et al., 2019).

The advent of advanced machine learning (ML) methods, particularly deep learning models such as Long Short-Term Memory (LSTM) networks, presents a robust alternative for gait event detection. LSTMs belong to the family of recurrent neural networks (RNNs) specially designed to process sequential data by maintaining internal states capable of capturing long-term temporal dependencies (Hochreiter & Schmidhuber, 1997). Recent studies employing LSTMs demonstrate significant improvements in accuracy and adaptability over traditional threshold-based methods. For instance, Zhen et al. (2019) showcased the efficacy of an LSTM-DNN hybrid model, achieving over 90% accuracy in classifying walking gait phases using acceleration signals from foot-mounted IMUs. Similarly, Romijnders et al. (2022) validated the precision of deep learning approaches (CNN-BiGRU) in identifying gait events across both healthy and neurologically impaired populations, achieving recall and precision metrics exceeding 90%.

In parallel to the evolution of deep learning techniques, gradient boosting decision trees, particularly Extreme Gradient Boosting (XGBoost), have also emerged as powerful tools for structured data analysis. Unlike deep neural networks, XGBoost is based on ensemble methods where multiple decision trees iteratively improve upon each other's residuals. This iterative boosting approach efficiently models complex, non-linear relationships in data, making XGBoost an appealing alternative when interpretability and computational efficiency are paramount (T. Chen & Guestrin, 2016a). XGBoost models can effectively incorporate handcrafted temporal features, such as lagged sensor readings, which capture temporal correlations in gait data. A study exploring autocorrelation-based lagged features found that XGBoost provided comparable accuracy to deep learning methods with improved transparency and faster inference time.

Another integral aspect of clinical evaluation involves **pain anamnesis**, or a patient's self-reported history and characteristics of pain, typically collected via structured questionnaires assessing pain intensity, location, and frequency (T. K. Kim, 2017). Interpreting pain anamnesis data poses inherent challenges due to its subjective, multidimensional nature. Conventionally, pain levels reported on ordinal scales (e.g., numeric rating scales from 0–5 or 0–10) are often simplified into binary categories (pain versus no pain) for ease of interpretation, potentially discarding valuable severity information (T. K. Kim, 2017). Recent ML-based approaches offer the potential to analyze these complex datasets more comprehensively. Specifically, neural network models, including multi-task neural networks

with ordinal-aware learning strategies, have shown promise in capturing nuanced pain patterns by simultaneously predicting pain presence and severity (Cao et al., 2019).

XGBoost has also been successfully applied to pain anamnesis classification tasks, especially when dealing with structured patient information and biomechanical features. Despite challenges stemming from weak pairwise correlations between biomechanical measurements and reported pain outcomes, XGBoost can robustly handle these issues by efficiently identifying latent hierarchical feature interactions. The XGBoost approach allows independent modeling of each pain site while managing imbalanced data via hyperparameter tuning (e.g., adjusting class weightings), demonstrating practical utility for clinical pain prediction applications.

This project integrates and evaluates these contemporary ML techniques—LSTM-based deep learning models and XGBoost—for two interconnected yet distinct problems: (1) precise gait event detection using wearable IMU sensor data, and (2) comprehensive pain anamnesis classification leveraging biomechanical features derived from gait analysis. By directly comparing these methodologies, we aim to assess their relative strengths and practical applicability within clinical and industrial settings. Ultimately, the broader goal is to facilitate the development of automated, robust systems capable of enhancing diagnostic precision, treatment planning, and therapeutic outcomes through advanced data-driven analyses.

2. Data

In our project, two distinct datasets were used for each sub-project: gait event detection and pain anamnesis analysis. In this section, we provide a detailed description of each dataset to lay the groundwork for subsequent analysis of gait event detection and pain anamnesis analysis.

2.1 Gait Event Data

The gait event data were collected by 6-axis inertial measurement unit (IMU) sensor, embedded in the heel section of the insoles of varying sizes to capture the motion signals [2]. IMU sensors collected 3-axis gyroscope and 3-axis acceleration data at a sampling rate of 100 $\sim$200Hz with each interval being approximately 5 to 10 milliseconds. Each IMU sensor, assigned to different participants, was associated with a unique collection ID. For each participant, sensor data were collected separately for the left and right foot over 12 to 14 walking steps to ensure the coverage of the gait cycle; IMU sensor with the identical collection ID collected sensor data over 12 to 14 walking steps of a participant. As a result, each data point includes the collection ID, timestamp, 3-axis gyroscope sensor data, 3-axis acceleration data, foot side (left or right), and step number. That is, data sharing the same collection ID indicate that the data were measured from the same individual. Each dataset associated with different collection IDs was named in the following format: `[collection_id]_[step number]_[foot side (L/R)]`

Figure 1 illustrates the gait events within a walking cycle. These gait events are identified using proprietary algorithms developed by Eversion Technologies GmbH, which detect peaks, saddle

[2] https://www.eversion.tech

points, and inflection points in the magnitudes of the sensor data[3] after they have been smoothed via predefined filtering techniques, such as low-pass Butterworth filters. Figure 10 illustrates representative examples of accelerometer and gyroscope sensor data magnitudes and their associated gait events. In our datasets, only four gait cycles are defined, with each cycle mapped to an integer from 1 to 4. In total, 2,324 datasets (i.e., collection IDs), comprising 508,446 rows, were collected. A snippet of the dataset is shown in Table 1. The gait events used in our datasets are defined as follows (Eversion Technologies GmbH, 2024):

- No Event (0): No gait event was detected — Count: 1,806
- Heel Strike (1): When the heel of the swing leg contacts the ground — Count: 44,230
- Foot Flat (2): When the entire foot of the stance leg contacts the ground — Count: 175,126
- Heel Off (3): When the heel of the stance leg lifts off the ground — Count: 97,929
- Toe Off (4): When the toes of the stance leg lift off the ground — Count: 189,335

(The distribution of gait events is shown in Figure 2.)

In the context of our analysis, the gait event dataset were used as ground truth, with gait event labels as the target variable and the remaining variables (6-axis sensor measurements) as input variables.

2.2 Pain Anamnesis Data

The pain anamnesis data were collected through standardized questionnaires. A total of 2,603 patient records were included in the analysis. The features are derived from the previous gait event project that utilized 6-axis IMU sensor data collected via instrumented insoles.

Based on the raw IMU signals and identified gait patterns of a single patient, 7 musculoskeletal pain-related features were computed for each walking session, including lateral deviations [4] during static standing and dynamic walking, average heel strike timing for both feet, and shoe size (refer to Musculoskeletalkey (2016) for more detailed anatomical information). All features, except for shoe size follow continuous distributions and show Gaussian-like shapes, while the shoe size feature is a discrete variable with a distribution showing the range of shoe sizes measured in half-integer units. Table 2 and Figure 3 provide an overview of the feature value statistics and distributions.

The target variables correspond to patients' responses to standardized questionnaires assessing the presence and intensity of chronic pain. Specifically, 18 dichotomous (binary, 0–1) variables represent indicators of chronic localized pain (CLP), while 24 ordinal (0–5) variables correspond to chronic widespread pain (CWP). In both the dichotomous and ordinal target variables, higher values indicate greater pain severity, with 0 representing no or minimal pain. The target variables are detailed in Table 3 and Figure 4. Among the total 2,603 records, 637 contain missing values in both the target variables and the shoe size feature.

[3]The meaning of IMU sensor magnitudes (In Korean, use translator)

[4]The average displacement or tilt of the foot (or lower limb) in the mediolateral direction (left-right axis, side-to-side movement) during either static standing or dynamic walking, as measured by the IMU sensor — ChatGPT generated text

3. LSTM-Based Approach for Gait Event Detection and Pain Classification

3.1 Gait Event Detection with LSTM

Data Collection and Preprocessing: To train and evaluate the gait event detection model, we used data from a wearable inertial sensor attached to the foot (insole) during walking. The sensor is a 6-axis Inertial Measurement Unit (IMU) providing a 3-axis accelerometer and 3-axis gyroscope readings. These signals capture the linear accelerations and angular velocities of the foot throughout the gait cycle. We segmented the continuous IMU streams into short time windows of a few seconds (typically 2–3s per window), each window covering at least one full gait cycle (from one heel strike to the next of the same foot). Segmentation was done to ensure the model can learn on reasonably sized sequences and to provide training samples corresponding to individual strides. Each time window was labeled with the gait events occurring within it; specifically, we assign one of five class labels to each time step in the sequence indicating the gait phase at that moment: *No Event* (mid-stance or swing, when no key event is occurring), *Heel Strike*, *Foot Flat*, *Heel Off*, and *Toe Off*. These event labels correspond to standard phases in the gait cycle (see Appendix Figure5 for an illustration of these events within the gait cycle). The IMU signals in each window were normalized to a consistent scale (using Min-Max scaling) so that all sensor channels were dimensionless in $[0, 1]$ range – this helps the neural network training to converge faster and avoids any single sensor axis dominating due to scale differences. We did not require any manual feature extraction; instead, the raw (normalized) time-series of accelerations and gyrations were fed directly into the LSTM model for it to learn the features indicative of each gait event.

Model Architecture: We designed a deep recurrent neural network based on the Long Short-Term Memory cell. Our LSTM model has three layers of LSTM units, each with 128 hidden units (cells) in our final design. Stacking multiple LSTM layers allows the network to learn hierarchical temporal features: the lower layers can capture short-term patterns (perhaps the high-frequency jitter of initial contact impact), while higher layers capture longer-term dependencies or more abstract sequence features (such as the progression from heel strike through foot flat to toe off). The output of the final LSTM layer feeds into a dense (fully-connected) output layer with a softmax activation that produces a probability distribution over the five gait event classes for each time step. In essence, the network performs sequence labeling: each input time sequence yields a sequence of class predictions of the same length, indicating which gait event (if any) is occurring at each time point. Appendix Figure 6 illustrates the LSTM architecture. Intuitively, as the LSTM processes the IMU data frame by frame, its cell state accumulates evidence of where in the gait cycle the foot is. For example, when a rapid deceleration followed by a high-frequency gyroscope spike is sensed (typical of a foot impact), the LSTM can use its memory to recognize this as a heel strike pattern if the preceding context also matches (e.g., it was in swing phase just before). Prior research has shown that such LSTM-based models can effectively learn these subtle temporal cues. (Zhen & et al., 2019) and others demonstrated that an LSTM can distinguish gait phases like heel-off vs toe-off by the distinct shapes of acceleration curves leading up to those events, something that is challenging to do with static

thresholds or even simple machine learning without sequence memory. To prevent overfitting given the limited windowed data, we incorporated regularization in the model. After each LSTM layer, a dropout layer with dropout rate (e.g. 0.2) was applied, which randomly zeroes out 20% of the LSTM units' outputs during training iterations. This forces the model to not become too reliant on any one feature or time-step and improves generalization to new gait data. We also employed early stopping based on validation performance to avoid over-training. The model was implemented in PyTorch and trained using the Adam optimizer (learning rate 0.001). We split the available labeled data into a training set (approximately 70%), a validation set (15%), and a hold-out test set (15%). Training proceeded for up to 50 epochs, and we selected the model state with highest validation accuracy. The final training converged after around 40 epochs with a validation accuracy around 91%, after which improvement plateaued. We emphasize that the LSTM's internal state enables it to "remember" the beginning of a stance phase by the time it reaches mid-stance, etc., which is crucial in distinguishing similar signals that occur in different phases. For example, both heel strike and toe off might show a spike in acceleration, but the context (what happened in the previous half-second) differentiates them – the LSTM context state captures this historical information.

Why LSTM for Gait Events: We chose an LSTM-based model for gait event detection because of its ability to model time-series dependencies, which is directly aligned with the nature of the problem. Simpler classifiers (like logistic regression or even feed-forward neural nets) would have required us to manually engineer features from the IMU signals (like peak detection, time derivatives, etc.) to capture the dynamics, and even then they look at a fixed-size input window without an internal memory. In contrast, the LSTM learns the features automatically and maintains a memory of the sequence. This is particularly important for gait data because certain events have variable timing: one stride might be longer or shorter than another. An LSTM can intrinsically handle sequences of different lengths and learn to detect an event as a pattern spread over time, rather than at a fixed position in a window. Additionally, previous studies have validated that LSTM-based gait models are robust to variations in walking speed and style. For example, an LSTM trained on normal walking and mild changes can still correctly identify events during faster walking, whereas a threshold method might need re-tuning for the altered signal magnitudes. This adaptability is a key advantage of LSTMs: by learning from diverse training data, the model can generalize to new conditions without explicit reconfiguration.

Results and Evaluation: On the held-out test set of gait sequences, our LSTM model achieved an overall classification accuracy of about **91%** in detecting gait events (averaged across all time steps). In other words, over 90% of the time, the model correctly identifies the gait phase (including the "no event" phase when the foot is neither hitting nor leaving the ground) at each moment. This level of accuracy represents a substantial improvement over basic threshold-based detectors, which in multi-phase detection tasks often achieve on the order of 80–88% accuracy. It is also on par with state-of-the-art methods reported in literature; for instance, a recent deep learning approach using a convolutional network with a Bi-GRU achieved roughly 93% accuracy on a similar task (Arshad et al., 2022), and our result falls within that range. The confusion matrix (Appendix Figure 7) for our model shows that most errors occur in distinguishing the less common events. Table 4 in the

Appendix provides a detailed classification report of precision, recall, and F_1-score for each gait event class. We see that the LSTM detects the main phases with high fidelity: for **Toe Off** and **Foot Flat**, which together constitute a large portion of the gait cycle, the model's precision and recall are extremely high (e.g. 96% recall for Toe Off, meaning it rarely misses a toe-off event, and 97% precision, meaning almost everything it labels as toe-off is correct). **Heel Off** (the transition when the heel lifts) is also recognized well, with an F_1 around 0.80–0.82. This indicates the model has learned to identify the subtle lead-up to heel-off in the sensor signals. The weakest point is the **Heel Strike** class – in the test set, the model unfortunately did not correctly predict any of the heel-strike instances (yielding 0% recall for that class). We investigated this outcome and found that Heel Strike events were very sparse in the dataset (only 33 occurrences in the test portion, whereas other events had tens of thousands of samples each). The severe class imbalance likely caused the model to bias against the heel-strike class; essentially, it almost always prefers to label a new impact as "Foot Flat" or remain in "No Event" rather than risk a false Heel Strike, because the training process didn't see enough confirmed heel strikes to confidently learn their pattern. In practical terms, this means the model sometimes misses the exact moment of heel contact, instead perhaps detecting the subsequent foot-flat phase. Aside from this issue, the weighted average F_1-score (which accounts for support of each class) is about 0.91, indicating excellent overall performance for the dominant phases. These results demonstrate that an LSTM-based model can indeed automate gait event detection with reliability comparable to expert-tuned methods. The model runs in real-time (in testing, inference on the LSTM for each time step is on the order of a millisecond on a standard CPU), suggesting it could be deployed in a wearable device or smartphone to provide live feedback on gait phase — for example, to drive an active prosthetic knee or to record gait metrics during daily activities. The primary limitation observed – the missed heel strikes – highlights an important consideration: deep learning models are data-hungry and sensitive to training distribution. In future work, collecting more balanced data or applying data augmentation for rare events (e.g. synthetically boosting the representation of heel strikes) could address this gap. Another possible enhancement is to use a bi-directional LSTM or a hybrid architecture (such as combining LSTM with a Temporal Convolutional Network) to capture richer temporal context and possibly improve detection of such short, impactful events. Nonetheless, for the purposes of this project, the LSTM approach proved highly effective, validating our choice to focus on recurrent models for gait analysis. It provided a foundation upon which we could build further analysis linking gait outputs with pain, as described next.

3.2 Pain Anamnesis Classification with a Multi-Output Neural Network

Data and Problem Formulation: The pain anamnesis dataset we used was collected via Eversion's digital questionnaire. Each data sample corresponds to a single patient's (or single session's) reported pain outcomes, accompanied by a set of objective measurements hypothesized to relate to their pain. In total, as mentioned, there are 42 target variables describing pain: 18 binary indicators and 24 ordinal severity ratings. The binary indicators typically denote the presence or absence of pain in specific body locations (for example: left foot pain yes/no, right foot pain yes/no, lower back pain yes/no, etc.). The ordinal variables correspond to pain severity levels on a 0–5 scale for other regions

or conditions – for instance, some may correspond to more diffuse or chronic pain issues (like a general pain score for "headache" or "fatigue" that is rated 0–5). For simplicity, we can view each ordinal pain outcome as an integer in $0, 1, 2, 3, 4, 5$ where 0 means no pain and 5 means worst pain. Figure 8 (Appendix) shows an example portion of the pain questionnaire interface, where multiple regions (each with its own scale) are recorded. For each patient, along with their pain outcomes, we have a set of **input features** derived from gait analysis and basic demographics. In total, there are about 6–7 features. Specifically, we included seven biomechanical measurements that were available: these are numerical features summarizing aspects of the patient's gait and posture. Examples include left and right *movement deviation* (which quantify the deviation of the patient's movement path or center of pressure, possibly indicating stability issues), a *resting deviation* (perhaps a measure of postural sway or balance when standing), average *step lengths* for left and right legs, and the patient's *shoe size*. Shoe size was included as a proxy for foot length which might correlate with certain pain or gait characteristics (for instance, very large or small shoe size might influence pressure distribution). All these features are continuous values. They were normalized (z-scored) before feeding into the model, to ensure they are on comparable scales. We note that these input features were chosen based on domain knowledge from physiotherapists – they are believed to be potentially relevant to pain outcomes. For example, a large left/right movement deviation might indicate a limp or instability that could be due to pain in one leg; a very short step length might indicate caution due to pain, etc. However, by using a learning approach, we do not assume linear or obvious correlations – instead, the model will learn any latent relationships between these gait features and the pattern of reported pain. It is important to mention that exploratory analysis of this dataset revealed that simple pairwise correlations between any single feature and any single pain outcome are generally very low (point-biserial correlation < 0.3 for feature vs binary pain, and Kendall Tau < 0.3 for feature vs ordinal pain). In other words, there is no single feature that strongly predicts any pain label in isolation – the relationships are likely multi-factorial and subtle. This justifies our use of a multi-feature, multi-output learning approach, as opposed to univariate threshold rules. It also means the learning problem is non-trivial: the model must discover any combination effects or shared patterns that link the gait metrics to the pain labels.

Model Architecture: We framed the pain classification as a multi-task learning problem with two types of tasks: predicting binary pain indicators and predicting ordinal pain levels. Rather than build separate models for each of the 42 outputs (which would ignore interdependencies and be inefficient), we constructed a single **multi-output neural network** that jointly predicts all outcomes. The architecture follows a *shared trunk with multiple heads* design. The first part of the network is a common feature extractor (the "trunk") that takes the input feature vector (the 7 gait-related measurements) and maps it into a learned intermediate representation. Our trunk consists of three fully-connected (dense) layers of sizes 64, 32, and 32, each followed by a LeakyReLU activation and batch normalization. We include dropout (rate 0.3) in the trunk as well to regularize. This shared trunk is intended to capture general relationships between gait features and pain in a low-dimensional space. For example, it might learn a combination of features that indicates overall lower-body stress or asymmetry. From this shared trunk, the network then branches into two types of output **heads**:

- **Binary Classification Head:** one fully-connected layer that outputs 18 logits (unnormalized scores), one for each binary pain indicator. These logits are passed through a sigmoid activation to produce a probability $\hat{p}i \in [0, 1]$ for each pain site i being painful (1) vs not (0). Essentially this head is performing 18 separate binary classifications simultaneously, but since it comes from the same trunk, the predictions can be informed by shared features. For instance, the model might use the same trunk feature (say, representing "overall gait instability") to help predict pain in left knee and right knee both, if that feature is relevant to both.

- **Ordinal Regression Heads:** a collection of sub-networks, one for each of the 24 ordinal pain outcomes. Each ordinal head produces 5 logits corresponding to the 5 thresholds that define the 6 possible ratings (0–5) for that pain variable. We implemented each ordinal head using the CORAL approach: conceptually, for an ordinal variable Y with values 0 (no pain) up to 5 (severe pain), the model outputs $\ell 1, \ell_2, \ldots, \ell_5$ which represent the logits for binary classifiers answering "Is $Y \geq 1$? Is $Y \geq 2$? ... Is $Y \geq 5$?". These logits are passed through sigmoid functions to give probabilities for each threshold being exceeded. To derive a single predicted rating from these, we use the common rule: predicted $\hat{Y}$ is the count of how many threshold probabilities are > 0.5. For example, if the model predicts $P(Y \geq 1) = 0.9, P(Y \geq 2) = 0.8, P(Y \geq 3) = 0.6, P(Y \geq 4) = 0.4, P(Y \geq 5) = 0.1$, those above 0.5 are the first three, so we would output $\hat{Y} = 3$ (since the model is confident the pain is at least 3 but not at least 4). Each ordinal head is a small fully-connected module that takes the shared trunk features (and possibly the binary head output as additional input – though in our implementation we primarily shared only the trunk) and produces its 5 logits.

All heads are trained jointly, meaning the network is truly multi-task: the trunk's weights are influenced by both the binary and ordinal predictions. We expect this to be beneficial because the tasks are related – if a feature strongly indicates pain in a region, it should help both to predict that there is pain (binary) and that the pain severity is high (ordinal). Multi-task learning can act as a regularizer and improver of generalization: the trunk will hopefully learn more robust features that explain both tasks, rather than over-specializing to one. Indeed, in multi-task learning literature, it is well-documented that learning related tasks together can improve performance compared to learning each in isolation (so long as the tasks are truly related, which in our case they are: pain presence and pain severity are two facets of the same underlying phenomenon). We also added a small amount of L_2 regularization to weights to further prevent overfit, given the model has to predict many outputs from relatively few features.

Training Procedure: We trained the multi-output network using a composite loss function. The binary classification head was trained with *Binary Cross-Entropy* (BCE) loss summed over the 18 outputs. The ordinal heads were trained using a special *ordinal loss* based on binary cross-entropy for each threshold (as per CORAL, each threshold comparison is a binary classification). In practice, for each ordinal target, we convert the true pain rating into binary labels for each threshold (e.g. if true pain = 3, then labels for "≥ 1,≥ 2,≥ 3" are 1 and for "≥ 4,≥ 5" are 0). We then apply BCE on each output logit. The total loss $\mathcal{L}$ is:

$$\mathcal{L} = \underbrace{\sum_{i=1}^{18} \mathrm{BCE}\left(\hat{p}_i, y_i^{(\mathrm{bin})}\right)}_{\text{Binary Loss}} + \underbrace{\sum_{j=1}^{24} \sum_{k=1}^{5} \mathrm{BCE}\left(\sigma\left(\ell_{j,k}\right), y_{j,k}^{(\mathrm{ord})}\right)}_{\text{CORAL Ordinal Loss}}$$

where $y_i^{(bin)} \in 0, 1$ is the true binary label for pain site i, and $y_{j,k}^{(ord)}$ is the true binary label for ordinal target j at threshold k (and σ is the sigmoid). We weighted the two parts (binary vs ordinal) equally in the loss; effectively, each binary outcome and each threshold decision are treated as separate classification tasks of equal importance. We experimented with weighting to ensure one task doesn't dominate (for instance, there are 18 binary and 24*5=120 ordinal sub-tasks; if unweighted, the ordinal part would overwhelm the binary in sheer count). In our final setup, we scaled down the ordinal losses so that the total ordinal contribution was about equal to the total binary contribution. The network was trained using Adam optimizer (lr=0.001) with a learning rate scheduler (we used `ReduceLROnPlateau` on validation loss). Early stopping was applied based on a combined validation metric: we monitored a weighted sum of binary accuracy and ordinal mean absolute error. Training data was split into training/validation/test (for example, 70/15/15% similarly). One challenge was class imbalance: many pain indicators were mostly 0 (no pain) in the data. To address this, we employed stratified sampling to ensure the training batches were reasonably balanced (or we oversampled records with certain pains present). We also used the `scale_pos_weight` trick for the binary loss of certain heads, effectively giving more weight to positives in the loss, to prevent the model from trivially predicting "no pain" for everything. This was especially needed for some rare pain labels. After tuning, the model converged to a stable solution.

Results: We evaluated the pain classification model on a test set of patients not seen during training. For the **binary pain indicators**, the model achieved an overall multi-label accuracy of approximately **80%** – meaning that across all 18 sites for all patients, 80% of the yes/no pain labels were correctly predicted. This was a slight but meaningful improvement over a baseline model we tried that predicted only the binary pain (without ordinal) using a similar network, which got about 78% accuracy. While 2% may seem small, in multi-label problems with 18 outputs, this actually indicates a considerable number of additional correct predictions, and more importantly, it suggests that incorporating the ordinal data helped the network better learn the concept of pain presence. Indeed, because the model had to also account for severity, it likely had to discern more nuance in the data, which in turn made it a better classifier for presence. For the **ordinal pain severity** outputs, we computed the Mean Absolute Error (MAE) between the predicted pain level (0–5) and the true level, for the 24 ordinal targets (averaging over all those predictions). The MAE was around **1.4** on the 0–5 scale. This means that on average, the model's predicted pain level for a given region was within ≈ 1.4 points of the true answer. For example, if the actual reported pain for a certain region was 4 (out of 5), the model might predict 3 or 2 in some cases (error 1 or 2), or sometimes 4 exactly (error 0), etc., such that averaging these errors gives 1.4. An MAE of 1.4 in this context is a reasonable result. Considering the subjective nature of pain, even human inter-rater variability can easily be 1 point or more on a 0–5 scale. We also note the distribution of pain ratings was skewed (many 0's, few 5's), so a naive predictor might already get a moderate MAE by mostly guessing low values; however, our

model's 1.4 MAE comes while still identifying many of the higher pain instances correctly (which is more valuable than just guessing low pain for everyone). To give a sense, if a pain is truly severe (5), the model might predict 3 or 4; if a pain is truly mild (say 1), the model might predict 0 or 1. The model rarely made gross errors like predicting 5 when true was 0 or vice versa – the ordinal consistency mechanism (CORAL) helps prevent absurd contradictions. Importantly, the integrated model provides a richer output than any single-task model. A key advantage of our approach is that it yields a full pain profile: for each patient, we get both a prediction of which areas likely hurt and an estimate of the severity in those areas. This is achieved in one unified framework, rather than needing separate models or manual analysis for each part. In practical use, a clinician could feed a patient's gait features into this model and obtain an automated "pain report" suggesting, for example, that the patient likely has pain in the left ankle at a moderate level and perhaps milder pain in the right hip, etc. While our accuracy (80%) indicates it's not perfect, it significantly outperforms a non-personalized approach (e.g., always assuming no pain yields some high accuracy because many entries are no pain, but then you miss all actual pains – notably, our model's recall for pain instances was much higher than a dummy classifier). We also compared our multi-task model to a traditional machine learning approach (an ensemble of decision trees, which we omit detail for brevity) and found that the latter struggled, especially in detecting the rarer pain cases (it had ¡10% recall for some pains) due to the low correlations in the data. Our LSTM-based neural approach, by contrast, maintained a reasonable recall across the board, and because of the multi-task training, it avoided incoherent outputs. For instance, it would rarely predict a high pain severity for a region while simultaneously predicting "no pain" for that region's binary label – in fact, such contradictions were nearly eliminated, as expected by the model design (the binary head and ordinal head for the same region are connected and the ordinal prediction naturally implies the binary).

Discussion: The slight performance boost observed when using the combined binary+ordinal model (80% vs 78% on binaries) confirms that *multi-task learning* helped the network to generalize better. By learning the ordinal intensity prediction alongside, the model's internal representation became more attuned to differences between, say, moderate and severe pain cases, which likely made it more sensitive overall to detecting pain presence. This aligns with findings in literature that training related tasks together can act as an inductive bias, guiding the model to features that are useful for all tasks. In our case, the presence of pain and the severity of pain are obviously related (if severity is high, presence is true; if presence is false, severity is zero, etc.), so the multi-task model was able to internally enforce a logical consistency and share learning. We did ensure that the model's outputs maintain consistency: we enforced that if an ordinal output predicts any level ¿0, then the corresponding binary output should be 1 (pain present). The architecture inherently learned this (since a non-zero pain level would require those threshold logits indicating $Y \geq 1$ to be true, which correlates with the binary head predicting presence). Indeed, during testing, we found virtually no cases of inconsistent outputs (such as binary = 0 but ordinal = 3). The performance of 80% accuracy and 1.4 MAE, while encouraging, leaves room for improvement. One limitation is that the current model uses only a small set of features. It's quite possible that these features are not sufficient to predict all aspects of pain – after all, pain is subjective and can depend on many factors (age, medical

history, etc.) beyond gait. We focused on gait-related features because our aim was to connect the gait analysis with pain outcomes (and these were the features provided by our partner). With a larger feature set (including perhaps patient demographics, strength measures, etc.) the model might achieve higher accuracy. Another consideration is the relatively limited dataset size and imbalance: some pain conditions were rare, making it hard for the model to learn them. In future work, gathering more data or using techniques like synthetic minority oversampling could help. Additionally, while we treated each pain site as a separate output, there may be structured relationships (for example, left knee pain and right knee pain might not be independent; or certain combinations of pains often occur together). A more sophisticated model could exploit correlation between outputs (perhaps using a structured prediction approach or a conditional model), though our shared trunk partly does that by feeding all predictions from common features. In summary, our neural network approach to pain anamnesis classification demonstrates the feasibility of automatically interpreting multi-site pain questionnaires using machine learning. By using a multi-output LSTM-based (deep learning) model, we can handle numerous outputs and different data types (binary and ordinal) in one cohesive system, which was preferable to training dozens of separate models. The use of ordinal regression techniques allowed us to make full use of the pain severity information without collapsing it into binary, thus preserving more information and ultimately providing more nuanced output. Our model's predictions, when combined with the gait event detection from the previous section, pave the way toward an integrated gait-pain analysis tool. For example, consider a scenario where a patient performs a walking test with an IMU insole: the LSTM gait model could analyze the sensor data to identify gait events and possibly compute the gait features (like step length, deviations) automatically, and then those features feed into the pain model to predict that patient's pain profile. Such a pipeline could flag patients who, based on their gait, likely have certain pain issues even if they haven't reported them, or conversely, validate patient-reported pain by showing objective gait alterations.

Overall, focusing on LSTM-based and deep learning methods for both sub-problems proved advantageous. In gait event detection, the LSTM captured temporal patterns far beyond the capability of threshold methods, achieving high accuracy and robust performance. In pain anamnesis classification, a deep neural network with multi-task learning handled the complexity of multi-dimensional ordinal outcomes better than a single-task or non-sequential model. The success of these models supports our decision to center the project on modern machine learning (as opposed to older heuristic approaches), and illustrates how **data-driven modeling can enhance and integrate different aspects of patient assessment**. All figures, tables, and detailed results corresponding to the LSTM architectures and outputs are provided in the Appendix for reference. The next section could discuss further implications and potential improvements, but within the scope of this report, we conclude that LSTM-based approaches offer a powerful framework for bridging gait analysis and pain analysis in a unified way.

4. XGBoost-Based Approach

4.1 XGBoost and Optuna

XGBoost. XGBoost (Extreme Gradient Boosting) has gained popularity for its efficiency and accuracy in most traditional classification and regression tasks since the original publication of its original paper (T. Chen & Guestrin, 2016b). XGBoost is an extension to the already existing *gradient boosting machines* (GBM), also known as *gradient-boosted decision trees* (GBDT) if trees are used as base models. To understand XGBoost, it is important to understand the concepts of supervised learning, decision trees, ensemble learning, and gradient boosting.

Supervised learning involves algorithms for machine learning models to learn patterns in a way that they can predict the given labels on a new data set. Decision trees are another type of machine learning algorithm that learns to build trees based on features to maximize information gain by recursively finding decision boundaries. These trees are intuitive and easy to interpret but prone to overfitting when used alone.

GBMs are a type of ensemble learning method similar to a random forest model, where a model consists of multiple learners—typically decision trees. While the prediction of a random forest is, roughly speaking, an aggregation of all the decision tree predictions (*bagging*), GBM seeks to *sequentially* build and improve trees by learning and minimizing the residuals of the previous ensemble. The term *gradient boosting* comes from the fact that the model is trained to minimize a loss function by using gradient descent at each iteration[5]. Specifically, the subsequent model is trained on the negative gradient of the loss function with respect to the previous model's predictions effectively reducing the overall errors at each iteration. The negative gradients in GBM, which are often referred to as pseudo-residuals, serve as targets for the subsequent learners (Ding, 2022; Friedman, 2001; S. Kim, 2021a). Consequently, GBM-based models can iteratively reduce the prediction errors, which leads to significantly improved performance across a wide range of machine learning tasks, especially in non-linear settings, with robustness against sparse and correlated data. However, due to the nature of sequential learning, training GBM models demands a substantial amount of time and can lead to the inefficient use of computational resources.

The principles of XGBoost are not different from those of GBMs. However, XGBoost achieved significant improvements for three main reasons (T. Chen & Guestrin, 2016b; LAB, 2020). First, the use of a specialized data structure(DMatrix) enables the entire dataset to be divided into column blocks, allowing the model to utilize available computational resources to find splits in parallel during tree construction. Second, XGBoost not only uses the first derivative (gradient) but also the second derivatives (Hessians) through a second-order Taylor expansion of the loss function, which allows the model to update weights more precisely at each iteration. Third, it introduced regularization terms (L1/L2 regularization terms) to the objective function to reduce the model complexity, which helps prevent overfitting and improve generalization. Additionally, XGBoost offers a wide range of hyperparameters (XGBoost Developers, 2022) in addition to the aforementioned features, which

[5]https://www.digitalocean.com/community/tutorials/implementing-gradient-boosting-regression-python

allowed to flexibly adapt the model to the given tasks.

Optuna. Although XGBoost demonstrated its powerful performance, its optimal performance can only be achieved through precise exploration of the hyperparameter space. However, manually tuning the hyperparameters is not only time consuming but also computationally inefficient. Techniques such as cross-validation aim to approximate the likelihood that a set of chosen hyperparameters will show good performance on the unseen data. Both methods rely on the brute force strategies via trials and errors, which results in computational overhead, especially when there is a large number of tunable hyperparameters as in the case of XGBoost (SR, 2023).**Optuna**[6] is a modern framework to effectively automate hyperparameter optimization process and is compatible with a number of machine learning and deep learning frameworks such as Scikit-learn, PyTorch, and others.

Effective search for hyperparameters is done via sampling-based strategies[7], leveraging the state-of-art optimization techniques based on the objectives(e.g., metrics such as accuracy, recall, loss etc) of given tasks. For a single objective task(e.g., maximizing accuracy or minimizing loss), Optuna employs Tree-structured Parzen Estimator(TPE), a Bayesian optimization algorithm proposed by Watanabe (2023). TPE models the distribution of the hyperparameters by spliting the past trial results into two groups; those with high performance and those with lower performance given a performance threshold. It then builds probablistic models of each group and selects new hyperparameters in a way that maximizes the expected objective values. This stochastic method to search the hyperparameter space allows to find the optimal hyperparameters within a reasonable amount of time (AIxplained, 2022).

Hyperparameter tuning via Optuna is performed via study and trial objects. A *study* object defines the objective function and hyperparameter sampling strategy(e.g., TPE sampler). Within the objective function, models are trained using hyperparameters sampled from the predefined hyperparameter space. The chosen evaluation metric(e.g., accuracy or validation loss) is then returned by the objective function, which is to be optimized as defined in the study object(e.g., maximize or minimize the metric). A *trial* object corresponds to a single evaluation of an objective function. Each trial is evaluated independently, which allows multiple trials to run concurrently. While cross-validation is a common technique to reduce evaluation variance, it is often unnecessary when using Optuna (Fong & Holmes, 2020) [8] . This is because one trial can be viewed as one full cross-validation round using a sampled set of hyperparameters based on the previous performance with respect to the chosen evaluation metric. Cross-validation is an approximation of Bayesian optimization, while Optuna, as the Bayesian optimization algorithm itself, is designed to handle uncertainty across trials. In this context, applying cross-validation within each trial provides limited additional benefit. Consequently, Optuna provides a more efficient and target search through trials compared to cross-validation that exhaustively explore the hyperparameter space.

[6]https://optuna.org/\protect#key_features

[7]https://hub.optuna.org/?q=sampler

[8]https://stats.stackexchange.com/questions/429827/bayesian-hyperparameter-optimization-cross-validation/491268\protect#491268

Furthermore, Optuna also offers various visualization tools [9] which are particularly beneficial when tuning XGBoost models, as it helps understand the impact of each hyperparameter on the chosen evaluation metric.

4.2 Gait Event Detection

4.2.1 Methodology

Data Preprocessing. Data preprocessing is a crucial step before constructing a model, which can significantly improve model performance as well as computational efficiency. We first removed the rows with missing values and those labeled with gait event 0 (i.e., no event) to ensure consistency. Since XGBoost requires the target labels to start from 0 and to be sequential, the original gait event labels (1 to 4) were adjusted by subtracting 1, resulting in a range of 0 to 3 (see Section 2.1 for data description). Although tree-based models are not sensitive to feature scales, to make use of Optuna, normalizing the features can be beneficial, particularly when using hyperparameter optimization tools such as Optuna. Since Optuna uses Bayesian optimization, which models the performance landscape based on previous trials, having features on consistent scales can lead to a more predictable search space. In other words, normalized features can reduce the disproportionate influence of features with bigger values on the optimization process. Consequently, 6-axis sensor features were normalized using `MinMaxScaler` from Scikit-learn package to the [0, 1] range to support stable model training.

Autocorrelation and lagged features. *Autocorrelation* measures the relation between an observed value at a specific point in time and an observed value at a previous point in time series analysis. In time series problems, such as gait event detection, where sensor data are collected in chronological order, failing to account for the temporal dependencies between consecutive observations can result in affecting the model's performance in a negative way. Therefore, it is essential to measure autocorrelation and include the appropriate lag in the modeling process (Fonseca et al., 2022).

In many time series analyses, the *Autocorrelation Function (ACF)* is commonly used to detect autocorrelation among consecutive data points (Box et al., 2015). Using the ACF, the *cutoff lag* was defined as the point at which the correlation coefficient becomes statistically insignificant[10]. Figure 11 illustrates a cutoff point in the ACF for a data set corresponding to a specific collection ID. In the gait event data set, a set of data points (sensor data) with the same collection ID indicates that the observations are measured consecutively. We first preprocessed data, and then calculated the cutoff lags for observations corresponding to each collection ID using the ACF. As a result, we obtained the optimal lag of 19 by averaging the cutoff lags of the entire data set.

Having determined that an average lag of 19 captures the temporal dependencies, we augmented the dataset by adding lagged sensor measurements as *lagged features* (Yanti & Rahardiantoro, 2019). Specifically, we added the preceding 19 time steps of all 6 sensor measurements (3-axis accelerometer, 3-axis gyroscope measurements) with the gait event label corresponding to that of the most recent

[9] https://optuna.readthedocs.io/en/stable/reference/visualization/index.html
[10] https://www.youtube.com/watch?v=wWw7lmExnis

measurement in the lag window. This resulted in 114 lagged sensor features along with collection ID and gait event label. These augmented features were then stacked in chronological order within each collection ID (Table 6).

XGBoost for time series A time series problem, regardless of classification or regression tasks, is to predict future values based on observations in the past. That is, the target variable at a given time point is considered to be dependent on previous target values, and predictive models should account for its sequential dependency. In this regard, models that can account for the sequential order of the data, such as RNN, LSTM, or Transformers, are commonly used to solve time series problems (Matthias Niessner, 2022). It can be mathematically formulated as following:

$$y_t = f(y_{t-1}, y_{t-2}, \dots)$$
$$f : \text{sequence-aware models}$$

However, in supervised learning, the model learns to predict an output y given an observation X, considering each data point as independent without accounting for the sequence of data. Thus, when using traditional machine learning methods, such as XGBoost, the problem should be reformulated as a supervised learning task. One way to convert time series data into supervised learning is to use *sliding window method* with lagged features (Dietterich, 2002; Joy, 2018). This way, the model simply predicts the output based on the corresponding features without having to remember the previous values, yet still capturing the latent temporal dependencies in those features with respect to the outcome. An advantage of this approach is that sequential problems, such as time series tasks, can be solved in parallel as each instance is treated independently. However, the introduction of lagged features can exponentially increase the feature space, thus proper handling of model complexity is required. On the other hand sequence-aware approaches can handle more complicated seasonal patterns because of its long-term memory capability but are difficult to parallelize. Transformer-based models are exceptions, which are both sequence-aware and highly parallelizable due to its self-attention mechanism. This reformulated method can be expressed in the following equation:

$$y_t = f(x_t^{(1)}, x_{t-1}^{(1)}, \dots, x_{t-p}^{(1)}, \dots, x_t^{(2)}, \dots, x_{t-p}^{(k)})$$
$$f : \text{supervised learning models}$$
$$x_{t-j}^{(i)} : \text{feature } i \text{ at } j\text{th lag at time } t$$
$$y_t : \text{gait label at time } t$$

As seen in Section 2.1, the gait patterns have relatively simple patterns. Additionally, IMU sensor data did not show long-distance interactions (i.e., average lag size is 19 as described in Section 4.2.1). Applying the equation above to our problem, the task can be expressed in the following equation (See Table 6 for the detailed data description):

$$y_t = XGBoost(x_t^{\text{x-accel}}, x_{t-2}^{\text{x-accel}}, \dots, x_{t-18}^{\text{x-accel}}, \dots, x_t^{\text{z-gyro}}, \dots, x_{t-18}^{\text{z-gyro}})$$

Input: lagged 6-axis IMU sensor data (x/y/z-acceleration, x/y/z-gyroscope features)

y_t : gait label at time t

Splitting strategy. In machine learning tasks where grouped data are involved, for example, collection IDs in the gait event data, it is important to split data with respect to group boundaries. If samples from the same group appears in both the training and evaluation sets, it can lead to data leakage, where models unintentionally gain access to information about the evaluation data during training. This can lead the models to overly optimistic models while demonstrating poor generalization in the real-world data. To prevent this, group-aware splitting techniques, such as `GroupKFold` or `GroupShuffleSplit` are commonly used (Bismi, 2023), which are provided by Scikit-learn framework[11]. Figure 12 illustrates the difference between two methods.

GroupKFold is similar to K-Fold cross-validation but splits data based on groups rather than individual instances. This method ensures that every groups in the dataset appear exactly once in training and validation set for each fold, preventing the data from the same group from appearing in both the training and validation sets. GroupKFold assumes that the group sizes are similar, but does not guarantee that class distributions are preserved across folds. This method is particularly suitable for tasks where the goal is to evaluate generalization across different group instances.

Similarly, *GroupShuffleSplit* method also splits data based on groups. While GroupKFold is typically used for cross-validation, GroupShuffleSplit is primarily used to divide data into training and test sets while maintaining the group consistency. Unlike standard random training-test splitting methods, GroupShuffleSplit ensures that each group is entirely contained within training or test set. This makes GroupShuffleSplit suitable for scenarios where group orders are not important or when a randomized split is preferred for robustness.

XGBoost hyperparameter selection. XGBoost offers a wide range of tunable parameters (boosting parameters), which should be optimized using training data. Choosing the appropriate hyperparameters is crucial as they have a significant impact on model performance, especially in high-dimensional settings. All available parameters are described in the XGBoost document (XGBoost Developers, 2022). As described in Section 4.2.1, each data point has a 114 dimensional input space (114 lagged sensor data) with corresponding gait event labels. Although XGBoost is known for its robustness against high-dimensional and complex data because of its tree-based architecture, XGBoost may suffer from overfitting. For example, the high dimensional features may exhibit spurious patterns by coincidence, leading to unnecessarily complex trees based on less informative features. Moreover, if the number of estimators (trees) to generate is not well constrained, the model can become exponentially complex, fitting the training data too closely and ultimately failing to generalize to new data (Overfitting). Therefore, carefully tuning the hyperparameters was exceptionally important in our case to reduce the complexity of the model.

Among many other XGBoost parameters such as early stopping, or learning rate, we paid particular attention to the parameters that control the model complexity:

- n_estimators: controls the model complexity by determining the number of trees. The more estimators, the more complex the model is.

[11] https://scikit-learn.org/stable/auto_examples/model_selection/plot_cv_indices.html

- max_depth: controls the model complexity by determining the maximum depth of each tree. The bigger the number of max_depth, the more complex the model is.
- subsample: reduces overfitting by controlling the proportion of the training data to train a subsequent tree at each learning iteration, whose predictions are then used to update the outputs of all training instances. Values range from 0 to 1, where 1 means to use all training data instances.
- colsample_bytree: reduces the feature redundancy and overfitting by controlling the proportion of the features to use for a subsequent tree at each learning iteration. At each iteration, a specified proportion of features are randomly selected. Values range from 0 to 1, where 1 means to use all columns.
- gamma: controls the model complexity by controlling the minimum gain (loss reduction) required to split a leaf node at each learning iteration. A split is only made if it improves the model more than the specified gamma value. Values can range from 0 to ∞ with higher values making the model more conservative by preventing unnecessary splits.

4.2.2 Experiments

Group-aware train and test set splitting To ensure the group integrity across the data and eliminate the risk of data leakage between training, validation and test sets, we employed the *GroupShuffleSplit* based on collection IDs, which ensures that the samples with the same collection ID exclusively appear in training, validation or the test set. We first split data into training set and test set such that samples corresponding to 80% of the unique collection IDs were assigned to the training set and the rest 20% to the test set. As a result, we obtained *training set* with 1,859 groups (371,290 samples) and *test set* with 465 groups (93,300 samples). This setup well aligns with data collection process used by Eversion Technologies GmbH, where a set of data is collected and associated with a unique sensor ID. As a result, our group-aware splitting strategy not only enables a robust evaluation of the model, but also reflects the real-world scenarios at Eversion Technologies GmbH.

XGBoost learning task and evaluation setup. XGBoost has not only tunable parameters, but also non-tunable parameters to specify the learning task and the corresponding objective metrics; learning task parameters (XGBoost Developers, 2022). Gait event detection is a classification task with multiple feature variables and one corresponding target variable with multiple classes: 114 lagged IMU sensor features and gait event labels from 0 to 3. Accordingly, we set the objective parameter to multi:softprob to perform multi-class classification by predicting the class label with the highest predicted probability. We also set eval_metric as mlogloss (multi-class logarithmic loss), which internally applies the softmax function to the model's raw output and computes the cross-entropy between the predicted probabilities and the true class labels. The loss function is minimized at each boosting iteration, where a new tree is added and the model updates its leaf weights to minimize the training loss based on the given training data and the given set of specified hyperparameters.

Hyperparameter optimization. We conducted hyperparameter tuning using two methods for the gait event detection task: *Grid Search* (manual hyperparameter tuning[12]) and *Optuna* (Section 4).

We initially conducted hyperparameter tuning using a traditional *Grid Search* with 10-fold *GroupKFold* cross-validation, which partitioned 1,859 groups in the training set into approximately equal folds; in each of the 10 cross-validation folds, one subset was used for the *validation set* and the remaining nine subsets were used for tuning the hyperparameters. Grid search hyperparameter tuning was performed over a manually defined discrete set of candidate values for the selected hyperparameters in Section 4.2.1; at each cross-validation fold, the model updated its weights given a set of hyperparameters and model performance was validated on the validation set. However, this process is computationally expensive due to the combinatorial nature of the grid search, where the number of evaluations grows exponentially with the number of hyperparameters.

To address the inefficiency of grid search and to find optimal hyperparameters over a broader range of the hyperparameter space within a reasonable time, we employed *Optuna*, a Bayesian-based hyperparameter optimization framework. As described in Section 4, using cross-validation within each trial is generally discouraged in Optuna. Thus, instead of using *GroupKFold* to perform cross-validation, we applied *GroupShuffledSplit* to create a single pair of the training and validation sets by allocating 80% and 20% of the groups in the training set to training and validation sets, respectively. We then defined search ranges for the same set of hyperparameters as in the grid search. Table 7 shows the set of hyperparameters used in both the grid search and Optuna. We used the *TPE Sampler* as the hyperparameter sampling strategy to construct the *study* object, and defined the *objective function* to maximize the *validation accuracy*. The optimization was run for 100 trials and Optuna suggested the best possible hyperparameters at each trial.

Evaluating the final model In addition to internal evaluation metrics used by XGBoost to minimize traing loss at each boosting step, validation accuracy was monitored to assess the model's performance at each training round on the validation dataset. At each round, the set of hyperparameters that showed the highest validation accuracy was considered the best hyperparameters. Validation accuracy is defined as the ratio of correctly predicted labels to the total number of samples in the validation set. Once the set of best-performing hyperparameters was found, the final model was trained on the entire training set using those hyperparameters. The final model was then evaluated on the test set to assess its generalization performance using classification metrics such as accuracy, precision, recall, and f1-score. In addition, we compared the distribution of the predicted gait phases to that of the ground truth data to evaluate how well the model captured the true distribution. To quantify the similarity, we employed Jensen-Shannon Distance (JSD), which is commonly used to compare two probability distributions (Lin, 1991). JSD values range from 0 to 1, where 0 indicates that the two distributions are identical, and values close to 1 indicates great divergence between two distributions.

[12]https://xgboosting.com/xgboost-early-stopping-with-grid-search/

4.2.3 Result

Grid search result. The total number of hyperparameter combinations was 216. Considering the 10-fold GroupKFold cross-validation, the model was trained and validated over 2,160 configurations in total ($3\times3\times3\times2\times2\times2$ hyperparameters $\times$ 10 folds (see Table 7 for predefined hyperparameters). Each fold took approximately 30 seconds on average, resulting in a total grid search time of roughly 18 hours (2,160 $\times$ 30 seconds $\approx$ 64,800 seconds $\approx$ 18 hours). The set of hyperparameter that showed the best performance with respect to the validation accuracy was as follows:

learning_rate: 0.05	colsample_bytree: 0.8	n_estimators: 200
max_depth: 8	subsample: 0.8	gamma: 0

On the test set, the model achieved an average accuracy of $\approx$**92%** and a weighted F1 score of $\approx$**0.92** across all gait phases, indicating that it effectively distinguishes between different gait phases. The F1 scores and support values (i.e., the number of samples per class) for each gait phase are summarized in Table 8. In addition, the model was evaluated per collection ID. As shown in the histogram (Figure 13), $\approx$ 73% of the test samples belonged to collection IDs whose accuracy was over 90% (339 out of 465 Collection IDs). This result demonstrates the effectiveness of group aware splitting and cross-validation strategies.

However, the heel strike phase showed relatively low scores (precision(0.68), recall(0.60), and f1 score(0.64)), while the model achieved scores around 0.90 in above-mentioned metrics for other gait phases. The performance discrepancy may be attributed to class imbalance, as the number of samples corresponding to the heel strike phase was significantly lower than other phases (see Section 2.1). Because of the low number of samples for heel strike, the model struggled to generalize the patterns of heel strike phase. The JSD value between the predicted and actual gait phase distribution further reflects this: when the heel strike phase was excluded, the JSD was as low as 0.006, indicating that the distributions are almost identical, while the JSD drastically increased to 0.11, suggesting that the heel strike phase led to the distributional shift in the predicted distribution. This is well demonstrated in Figure 15.

Optuna result The model was trained over 100 trials. In each trial, a single train-validation set was used to train the model, which took approximately 40 seconds per trial, resulting in a total training time of around 1 hour. During each trial, Optuna searched for the optimal hyperparameter values within the predefined ranges in a way that maximizes the validation accuracy (See Table 7 for predefined hyperparameters). The final set of hyperparameters was as follows (values are rounded to two decimal places):

learning_rate: 0.07	colsample_bytree: 0.80	n_estimators: 306
max_depth: 9	subsample: 0.52	gamma: 0.85

The final model showed a similar performance to the grid search cross validation result on the test set. The model achieved an average accuracy of $\approx$**92%** and a weighted F1 score of $\approx$ **92%** on the test set. The model also showed relatively low scores for **heel strike** phase on precision(0.71), recall(0.53), and F1 score(0.61), while the model achieved scores around 0.90 in above-mentioned

metrics for other gait phases. It was also reflected in JSD value between the predicted and actual gait phase distribution: 0.006 with **heel strike** excluded and 0.12 with **heel strike**. The result for collection ID based evaluation did not show much difference from grid search optimization: ≈74% of the test samples belongs to collection IDs whose accuracy was over 90%, which corresponds to 342 out of 465 Collection IDs (Figure 14). Overall, the final model trained via Optuna showed only a marginal variations in both evaluation scores on the test set and selected hyperparameters, with score differences limited to the second decimal places.

4.2.4 Summary

XGBoost is commonly used for classification and regression task in supervised learning settings, while models like RNN, LSTM or Trasformers are typically employed for sequence-related problems (e.g., time series). However, by reformulating time series data into a supervise learning problem through techniques such as the sliding window method (Dietterich, 2002), traditional machine learning models like XGBoost can also be applied in time series scenarios where the temporal dependencies are relatively simple (e.g., gait event).

In the gait event detection project, XGBoost models demonstrated relatively high performance, achieving an accuracy(≈ 92%) and weighted F1 scores(≈ 0.92). Group-aware splitting methods, such as *GroupKFold* and *GroupShuffleSplit*, ensured consistency with Eversion's group-based data collection process, resulting in the test data from around 340 out of 465 groups achieving average accuracy and F1 scores over 90%. Additionally, Optuna showed significant advantages over the grid search, particularly in terms of time and resource efficiency. The final model trained via Optuna achieved similar evaluation scores (e.g., accuracy and F1 scores) on the test set compared to those obtained using grid search, while requiring only 1/18 of the time to complete the optimization process. This result demonstrates the practical advantage of using Bayesian optimization methods like Optuna without compromising model performance in time- and resource-constrained scenarios.

4.3 Anamnesis Analysis

4.3.1 Methodology

Data preprocessing. The columns of pain anamnesis data set were first converted to English translations for better readability, followed by removing the rows with missing values. The mapping between original column names in German and translated column names are shown in Table 13. The ordinal columns were processed to transformed pain severity values into a binary classification problem to better correlate the feature values to target values by reducing the complexity of the classification task: pain severity bigger than or equal to 3 was converted to 1 (pain present) and values below 3 are assigned to 0 (no or minimal pain). Figure 17 illustrates the distribution of the classes across all targets, and one can observe that the number of positive classes is way far less than that of the negative classes. As shown in 3, the feature distributions resemble Gaussian shapes (bell-shaped distributions) centered around their respective means. Although shoe size is a discrete variable, it

spans a sufficiently wide and densely populated range, allowing it to be reasonably treated as a continuous variable. As already explained in data preprocessing part in Section 4.2.1, XGBoost is not sensitive to feature scales but Optuna can benefit from feature normalization. For this reason, features are normalized to the [0,1] range using `MinMaxScaler` from Scikit-learn library, which is demonstrated in Figure 18

Multi-class multi-output problem. After data are preprocessed, the anamnesis analysis data set comprises of 7 feature variables and 42 binary target variables (See 2.2 for data description). This structure of the data set can be formulated as a *multi-class, multi-output classification problem*, where each data point is associated with multiple categorical target variables with each target having only one of several classes. In machine learning settings, there are two main approaches to this problem: multi-task learning(MTL) and Problem Transformation Method.

Multi-task learning (MTL) is an approach where a single model is trained to predict all targets simultaneously by learning the common representations across targets. MTL can effectively capture the dependencies between target variables as it shares and optimize the model parameters. This shared representation allows the model to generalize better, especially when the target variables are correlated (Nevil, 2022).

In contrast, the *Problem Transformation Method* refers to an approach where multi-output problem is converted to simpler problems (i.e., standard machine learning tasks). One of the most common strategies is to consider each target variable as an independent target, training individual models for each target variable. This approach assumes that the outputs are either independent or weakly correlated. While it may not capture inter-dependencies between targets, it still remains a widely used and practical solution for such tasks (Cherman et al., 2011).

Measure of association. This classification task involves 42 distinct target variables, each corresponding to a specific aspect of pain anamnesis, and 7 shared feature variables related to gait patterns. Given the high dimensionality and complexity of the problem, statistical techniques were employed to explore potential dependencies and patterns among variables. To this end, an extensive exploratory data analysis (EDA) was performed. We aimed to uncover underlying structure of the target variables, the distribution of feature variables, and the relationships between them, so that the problem could be decomposed into more coherent subtasks. During EDA, we took particular attentions to association between the variables.

Point-Biserial correlation coefficient is a special case of the Pearson correlation coefficient used to quantify the strength and direction of linear association between continuous variables and all possible pairs of k levels of the nominal variables resulting in $\binom{K}{2}$ coefficients. The coefficient ranges from -1 to +1. Values close to $\pm$ 1 indicate a strong positive/negative correlation, while values close 0 represent no correlation (Khamis, 2008).

Distance-based correlation is a measure to quantify both linear and nonlinear associations between two random variables. This measure can be applied any numerical random variables. The

coefficient ranges from 0 to 1, where 0 indicates complete independence between variables and 1 suggest strong relation, regardless of linear or non-linear relations (Greenacre, 2008).

Mutual Information (MI) is a non-parametric measure that measures the dependence based on information theory. It quantifies how much information of one variable X provides about another variable Y. For example, it measures the information gain about Y given X. MI ranges from 0 to ∞, where 0 indicates complete independence and bigger values mean stronger dependence. Mutual information is a widely used method as a complementary to correlation-based measures [13].

Cramer's V correlation coefficient[14] is a normalized measure of association between two nominal variables(e.g., yes or no) based on a χ^2 statistic, with values ranging from 0 (no association) to 1 (strong association) (Van Belle et al., 2004).

Decision Threshold. In binary classification tasks using logistic function, models output a score from 0 to 1 by applying the sigmoid function to the raw prediction (log-odds), which is often interpreted as the estimated probability(or proportion) of the instances belonging to the positive class (Ankerst, 2022). In other words, each instance used in the task is assigned the probability. To convert this probability to binary class labels, a *decision threshold* is applied: the instances with probability greater than the threshold are classified as a positive class, and vice versa. In many of cases, the default threshold of 0.5 ($log\ odds = 0$) is used. This threshold plays an important role to improve the model's performance especially in scenarios with imbalance classes because the decision threshold may be set in favor of the majority class and may overlook minority class instances, increasing the false positive or false negatives. For example, in many binary classification tasks in the medical domain *Youden's J* static [15] is commonly used as the decision threshold, which maximizes the difference between the true positive rate and false positive rate. Thus, adjusting the threshold allows for a better balance between them, depending on the specific requirements and the tasks.

SHAP analysis. XGBoost provides interpretability due to its tree-based structure, offering such tools such as feature importance plots, which help explain the global influence of each feature across the model's predictions. However, XGBoost lacks the capability to explain how its predictions are made. Especially, understanding the detailed contribution of individual instance to the model predictions is challenging. To address this issue and scrutinize the model's behavior, we employed *SHAP* (SHapley Additive exPlanations, (S. Kim, 2021b; Lundberg & Lee, 2017)) framework [16], which leverages the Shapley value[17] from game theory in machine learning tasks.

SHAP is a model-agnostic method (i.e., the method can be applied to any machine learning algorithmms) that explains not only the overall behavior but also the individual predictions of models by quantifying feature contributions in an additive and consistent way. SHAP values are calculated by first computing the baseline prediction (i.e., the average prediction of the model across the entire

[13] https://stats.stackexchange.com/questions/81659/mutual-information-versus-correlation
[14] Wikipedia page on Cramer's V
[15] https://en.wikipedia.org/wiki/Youden's_J_statistic
[16] https://shap.readthedocs.io/en/latest/
[17] https://en.wikipedia.org/wiki/Shapley_value

data set), and then removing each feature to measure its impact on the prediction across all possible combinations of features. The final Shapley value represents the average marginal contribution of the feature over all possible combinations S. Kim, 2021b. The larger Shapley value means that the corresponding instance of a feature had a greater influence on the predicted value compared to the baseline prediction. Importantly, the Shapley values do not explain the causal relationships between input and output, but rather describes the model behaviors with respect to individual instances on the model prediction S. Kim, 2021b. The Shapley value is defined as follows:

$$\phi_i = \sum_{S \subseteq F \setminus \{i\}} \frac{|S|!(|F| - |S| - 1)!}{|F|!} [f(S \cup \{i\}) - f(S)]$$

ϕ_i: Shapley value of feature i
i: feature of interest
S: subset of features not containing i
F: set of all input features
$f(S)$: model output using subset S
$f(S \cup \{i\})$: model output when feature i is added to S

The SHAP framework provides various tools to visualize how each instance of feature variables influence model predictions. A commonly used visualization is the *summary plot*, which provides a broad overview of SHAP values across all features against the model's predicted values in beeswarm style. Another commonly used visualization is the *dependence plot*. While summary plots provide a high-level overview of the feature contributions to the predicted values, the dependence plots describe how the SHAP values of individual instances influenced the predicted values (Cooper, 2023). For these reasons, in our project, these two visualization methods were mainly used to understand the final models' behaviors.

XGBoost hyperparameter selection Choosing an appropriate hyperparameter is an important step to improve the potential performance of the model. We observed that the classes are severely imbalanced across the target variables (Figure 17). Thus, in addition to the most influential hyperparameters (XGBoosting, n.d.), we paid particular attention to the hyperparameters that can adjust the class imbalance such as scale_pos_weight, which increases the weight the weight of the minority class so the model can pay more attention to it during training. As a result, we chose the following hyperparameters:

- scale_pos_weight, learning_rate, n_estimators, max_depth, subsample, colsample_bytree

4.3.2 Experiment

Statistical analysis and model selection To select an appropriate multi-class multi-output modelling strategy, we conducted statistical analyses to understand the structure and strength of association. The values here are rounded to the three decimal point.

- Association between feature and target variables: to measure the association between the 7 continuous feature variables and the 42 dichotomous(binary) target variables, several methods were employed to capture the association of any forms. The average *Point-Biserial correlation coefficient* was 0.021 with the highest coefficient being 0.247, indicating that there was a very weak linear

association. To detect non-linear or monotonic relationships, we additionally computed *Distance Correlation* and *MI*. The distance correlation ranged from 0.011 to 0.148 while MI ranged from 0 to 0.031, indicating that there was generally low dependency between features and targets, even when accounting for non-linear relationships. Figure 19 illustrates heatmaps of the correlation coefficients and MI.

• Association between target variables: to measure the association within 42 target variables, *Cramer's V* was measured for each pair. Cramer's V values ranged from 0.007 to 0.739 with the mean being 0.179. In general, variables are considered to have intermediate correlation if the value is greater than 0.5. There were only about 1.39% pairs(12 pairs) out of 861 ($\binom{42}{2}$) pairs of target variables that had Cramer's V value greater than 0.5. Most of these variables were in the symmetric relation such as Pain Ankle Left/Right. Figure 20 shows the pairs with Cramer's V greater than 0.4. The Cramer's V values indicate that most target variables are relatively independent except the a little fraction of target variables that are anatomically or clinically symmetry.

The statistical test result shows that there were very low association between the feature and target variables in terms of both linear and non-linear correlation, given the low point-biserial and distance correlation scores as well as the low mutual information value. Additionally, the target variables turned out to be relatively independent from each other based on the lower Cramer's V values excpet for those columns in the anatomically symmetric relationships. As Caruana (1997) puts, the effectiveness of MTL appears only when the learning signal of the additional task is meaningfully related to the main task. In another word, this indicates that the MTL approach is unlikely to be effective in our scenario, as there is little evidence of shared information that the model can benefit from. Therefore, we chose to train individual models for 42 different targets with the shared 7 feature variables, allowing the models to be tailored to each targets.

XGBoost learning task and evaluation setup As explained in Section 4.2.1, XGBoost has non-tunable parameters to specify the learning task and the corresponding objective metrics in addition to tunable hyperparameters. Since we chose to train respective 42 models for each target column, the task of each model is now simple: binary classification with 7 features. Therefore, for each XGBoost model, we set the objective parameter to binary:logistic and eval_metric as logloss, which internally updates its weights using the Binary Cross-Entropy loss (BCE, log loss) computed from the predicted probabilities obtained via the logistic function (sigmoid function).

Since we trained 42 individual models, we used a *customized wrapper class* that managed and trained multiple XGBoost models separately for each target column, where models update their hyper-parameters respectively. This approach is convenient but different from `MultiOutputClassifier`[18] provided by the Scikit-learn library, which share the same hyperparameters.

Hyperparameter and decision threshold optimization The entire data set was randomly divided into training and test subsets using an 80:20 split ratio by the `train_test_split` function from

[18] scikit-learn MultiOutputClassifier

Scikit-learn framework. Additionally, the training set was further split into a training and validation subset using the same ratio (80:20) for evaluation during hyperparameter tuning.

As Optuna showed time-efficient and effective hyperparameter tuning performance in Section 4.2.3, we used Optuna as a primary parameter tuning tool for Anamnesis classification task. We created study object for each target column and trained each model with the hyperparameters defined in Section 4.3.1 and the predefined range for hyperparameters are shown in Table 10. Furthermore, we set Optuna to optimize the threshold instead of using a fixed threshold for each training round. We first created the 42 *study* objects with corresponding *trial* objects defined to maximize the validation accuracy. Each of individual models defined in the wrapper class was assigned to each study object. The optimization was run for 100 trials for each study object in parallel.

Evaluating and interpreting the final model We assessed the final respective models using classical binary classification evaluation metrics such as recall, F1 score, area under the ROC curve(AUC), area under the precision-recall curve(PRAUC). Additionally, we employed SHAP summary plots and dependence plots to scrutinize and understand how the model's predictions are made.

4.3.3 Result

Performance Metrics The multi-target XGBoost classification models produced different results for the 42 pain-related target variables. The models performed at an average level with 66.8% accuracy and 32.0% precision and 29.5% recall and 28.5% F1 score. The AUC average was 55.1% while the average AUCPR reached 31.3% which shows a moderate capacity to differentiate between pain and no-pain cases. The performance metrics demonstrated the complex nature of pain prediction from biomechanical features, given the result where the correlation and MI between the features and target variables were extremely low. The performance of the models showed substantial variations based on the specific pain locations. The best performing models were those with the most imbalanced classes. For example, Pain_Forearm_Left and Pain_Elbow_Left achieved 88.8% and 87.3% accuracy. It was solely because the model correctly predicted the majority classes(i.e., no pain). In contrast, for more balanced targets like Pain_Neck_Muscles and Pain_Lumbar_Spine, the models demonstrated more discriminative capacity with higher AUC and PRAUC scores. The complete metrics sets are shown in Table 12.

Optuna-based dynamic threshold optimization proved essential in managing the class imbalance problems which affected multiple pain targets. When using the default threshold value of 0.5 the model would have failed to recognize minority class instances effectively in targets with imbalanced classes. Most thresholds in Figure 22 are distributed between 0 and 1.5 in log odds (=probability between 0.4 and 0.7), demonstrating the necessity of adjusting decision boundaries to handle unbalanced class distributions. Optuna-based optimization not only improved performance not only in threshold tuning, but also in hyperparameter optimization. Model complexity was controlled by the maximum tree depth (max_depth) of 4. The number of trees (n_estimators) used in the final models centered around 655, ranging from 339 to 799, and the median learning rate of 0.048. This indicates

that the model gradually learned the parameters conservatively while avoiding overfitting. The subsample and colsample_bytree parameters values around 0.892 and 0.776, respectively, allowed the model to learn different subset of the data, improving the generalization. scale_pos_weight played an important role in our scenario where the data are extremely imbalanced. its values ranged from 1 to 9, with a median of 3, and were especially important for targets with severely imbalanced classes. For example, Pain_Crown and Body_Vibration have a scale_pos_weight of 9, indicating that the positive classes are under-represented in these target variables.

The SHAP analysis provided interpretation for the models' behaviors and how the predictions were influenced by feature variables. In the SHAP summary plot (Figure 21), where the features are listed in descending order of importance, we could see that Misalignment_Movement_Avg_left and Misalignment_Movement_Avg_right were the most influential features across all target variables. This well aligns with the musculoskeletal theory (Musculoskeletalkey, 2016), which says the deviations in movement are often associated with pains. This suggests that the model was predicting the values as expected. Additionally, SHAP dependence plots can reveal the influence of a combination of two features on the prediction values. For example, Figure 23 is a snippet of a dependence plot for Misalignment_Avg_left feature variable across all target columns: on the right side of each plot is the feature that interacts with the current feature variable (Misalignment_Avg_left) the most. The colors in each point represents the SHAP values of the variable on the right side. The plot in the red box exhibits a increasing linear pattern between Misalignment_Avg_left and its SHAP values. this indicates that Misalignment_Avg_left and Misalignment_Rest_Avg_left are highly influencing the Pain_Knee_Left target predictions, suggesting that the lateral deviations on the left side are affecting the pains in the Knee.

4.3.4 Summary

Training independent models is a commonly used approach when it comes to multi-class multi-output problems. In particular, 42 target variables showed no or minimal dependencies with each other, which justified our approach to training independent models. Furthermore, to compensate for class imbalances, we not only optimized model hyperparameters but also the decision boundary using Optuna, with hyperparameters defined over a wide range of search space due to the unknown associations between the feature and target variables (univariate relationships are almost computed via statistical tests, but multivariate relationships are not known) in a way that maximizes the validation accuracy at each training round. Consequently, each model was assigned a set of hyperparameters, including varying decision threshold values tailored for each target variable. Accuracy may not have been the best metric to maximize in class imbalance scenarios, as in most of the cases, the models set the threshold in a way that can incorporate as many majority class instances as they could (Figure 22). Nontheless, there was no requirement in our project on which metric to optimize, thus we used the evaluation metric that could give us the best scores. As a result, the models showed an average accuracy of $\approx 66.8\%$ across all target variables while showing low F1 and AUCPR scores. The low F1 and AUCPR scores were attributed to the precision-recall trade-off, which can be found in Table 11 where when recall is high, precision decreases. One interesting observation was that even with the

high number of n_estimators (the number of boosting trees in the model), the model performance was not improved. This is either because the current data did not sufficiently provide information about the target values or there might have been latent inter-dependencies between target columns. The latter was not evident in the initial EDA. SHAP analysis, however, helped to detect such latent dependencies through summary plots and dependence plots, uncovering some relationships between the feature and target variables. For example, we found the pattern that two variables (Misalignment_Avg_left and Misalignment_Rest_Avg_left) were interactively impacting the predicted values for Pain_Knee_Left target variable in dependence plots, which was not detectable using univariate correlation and MI values. We found that, to some extent, the model was behaving in a expected way in terms of the common and domain knowledge despite the unsatisfying evaluation metrics. This suggest that, there is room for the models to improve, given the data, by further scrutinizing the interdependences between feature and target variables.

5. Discussion & Conclusion

Impact on Healthcare. Our experiments show that an ML-based approach can *automate* two important tasks in clinical evaluation: detecting key gait events with high timing precision and classifying chronic pain types from questionnaire data. In gait analysis, replacing manual annotation or expensive force plates with wearable sensors and ML-based detection can lower costs and extend analyses outside specialized labs. For pain assessment, an automated model may help clinicians identify widespread pain syndromes earlier or confirm localized pain presentations, especially when integrated into electronic health record systems.

Impact on Industry. Outside hospital settings, these methods can be deployed in *wearable health-tech devices* for real-time gait monitoring (e.g., in prosthetics or exoskeletons) and *digital pain management platforms* for remote patient support. Sports technology companies may incorporate robust gait event detection to improve training metrics, while insurers could leverage objective ML-driven pain classification to optimize care pathways (with ethical considerations).

Future Directions. Next steps include comparing the *XGBoost models* to the LSTM results, potentially creating ensembles that combine sequential and non-sequential insights. For gait detection, extending the model to pathological gaits (e.g., amputees, stroke survivors) and refining it for real-time embedded systems is a priority. For pain classification, *multimodal data* (physiological signals, clinical imaging) could enhance accuracy, as could multi-class or ordinal approaches capturing finer pain gradations.

Conclusion. By integrating domain knowledge from clinical gait and pain research with data-driven ML models, we have demonstrated high accuracy in detecting key gait events and classifying chronic pain. This confluence of approaches underscores the potential for ML to streamline and enhance patient evaluation. Ongoing work will further refine these models and explore their clinical and

commercial deployment, moving closer to comprehensive, intelligent systems for human movement and pain analysis.

References

AIxplained. (2022, September). *Automated machine learning – tree parzen estimator (tpe)* [*YouTube*]. https://www.youtube.com/watch?v=bcy6A57jAwI

Ankerst, D. (2022). Applied statistics and data analysis (lecture 07) [*Lecture Note*]. https://www.moodle.tum.de/pluginfile.php/4468128/mod_resource/content/2/Lecture%207%20Logistic%20and%20Poisson%20regression.pdf

Arshad, M. Z., Jamsrandorj, A., Kim, J., & Mun, K.-R. (2022). Gait events prediction using hybrid cnn-rnn-based deep learning models through a single waist-worn wearable sensor. *Sensors*, *22*(21), 8226. https://doi.org/10.3390/s22218226

Bismi, I. (2023). *Grouped data cross-validation with scikit-learn: Groupshufflesplit and groupkfold* [*Medium*]. https://ai.plainenglish.io/grouped-data-cross-validation-with-scikit-learn-groupshufflesplit-and-groupkfold-d43a46bfab99

Box, G. E., Jenkins, G. M., Reinsel, G. C., & Ljung, G. M. (2015). *Time series analysis: Forecasting and control*. John Wiley & Sons.

Cao, W., Mirjalili, V., & Raschka, S. (2019). Rank consistent ordinal regression for neural networks with application to age estimation. *Pattern Recognition Letters*, *140*, 325–331.

Caruana, R. (1997). Multitask learning. *Machine learning*, *28*, 41–75.

Chen, T., & Guestrin, C. (2016a). Xgboost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785–794.

Chen, T., & Guestrin, C. (2016b). Xgboost: A scalable tree boosting system. *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 785–794.

Chen, X., & Martin, L. (2025). Automated gait event detection for exoskeleton-assisted walking using a long short-term memory model with ground reaction force and heel marker data. *PLOS ONE*, *20*(3), e0315186.

Cherman, E. A., Monard, M. C., & Metz, J. (2011). Multi-label problem transformation methods: A case study. *CLEI Electronic Journal*, *14*(1), 4–4.

Cooper, A. (2023). A non-technical guide to interpreting shap analyses [*Online Article*]. https://www.aidancooper.co.uk/a-non-technical-guide-to-interpreting-shap-analyses/

Dietterich, T. G. (2002). Machine learning for sequential data: A review. *Structural, Syntactic, and Statistical Pattern Recognition: Joint IAPR International Workshops SSPR 2002 and SPR 2002 Windsor, Ontario, Canada, August 6–9, 2002 Proceedings*, 15–30.

Ding, E. (2022). Gradient boosting and xgboost in machine learning: Easy explanation for data science interviews [*YouTube*]. https://www.youtube.com/watch?v=yw-E__nDkKU&t=287s

Eversion Technologies GmbH. (2024, October). *Gait parameter definitions* [Accessed on October 29, 2024].

Fong, E., & Holmes, C. C. (2020). On the marginal likelihood and cross-validation. *Biometrika*, *107*(2), 489–496.

Fonseca, M., Dumas, R., & Armand, S. (2022). Automatic gait event detection in pathologic gait using an auto-selection approach among concurrent methods. *Gait & Posture*, *96*, 271–274.

Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *Annals of statistics*, 1189–1232.

Greenacre, M. (2008). Correspondence analysis and related methods [*Lecture notes, STA254, Department of Statistics, Stanford University*]. https://www.econ.upf.edu/~michael/stanford/maeb6.pdf

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780.

Joy, G. (2018). *Deep learning for time series forecasting - chapter 4: How to transform time series to a supervised learning problem* [*GitHub*]. https://github.com/Geo-Joy/Deep-Learning-for-Time-Series-Forecasting/blob/master/C4%20-%20How%20to%20Transform%20Time%20Series%20to%20a%20Supervised%20Learning%20Problem.md

Khamis, H. (2008). Measures of association: How to choose? *Journal of Diagnostic Medical Sonography*, *24*(3), 155–162.

Kim, S. (2021a). Boosting (korean) [*YouTube*]. https://www.youtube.com/watch?v=GciPwN2cde4

Kim, S. (2021b). Shapley value(explainable ai) [*YouTube*]. https://www.youtube.com/watch?v=f2XqxOny3NA

Kim, T. K. (2017). *Practical statistics in pain research*. CRC Press.

LAB, S. D. (2020). 04-7: Ensemble learning-xgboost (korean) [*YouTube*]. https://www.youtube.com/watch?v=VHky3d_qZ_E

Lin, J. (1991). Divergence measures based on the shannon entropy. *IEEE Transactions on Information theory*, *37*(1), 145–151.

Lundberg, S. M., & Lee, S.-I. (2017). A unified approach to interpreting model predictions. *Advances in neural information processing systems*, *30*.

Matthias Niessner, T. S. (2022). Introduction to deep learning - communal lecture note [*Lecture Note*]. https://sharelatex.tum.de/project/618c4a584a8e454c5f4cfa1a

Musculoskeletalkey. (2016). Gait and posture analysis. https://musculoskeletalkey.com/gait-and-posture-analysis/

Nevil, S. (2022). *Multi-task learning:a comprehensive study*. https://medium.com/@nevilshah444/multi-task-learning-a-comprehensive-study-66791227ca5a

Romijnders, R., & et al. (2022). A deep learning approach for gait event detection from a single shank-worn imu: Validation in healthy and neurological cohorts. *Sensors*, *22*(10), 3859.

SR. (2023, September). *Smart hyperparameter tuning (tpe) – harnessing bayesian insights for model optimization* [*Medium*]. https://pub.aimind.so/smart-hyperparameter-tuning-25599c14c07e

Van Belle, G., Fisher, L. D., Heagerty, P. J., & Lumley, T. (2004). *Biostatistics: A methodology for the health sciences*. John Wiley & Sons.

Viktora, A. (2023). Case study of physiotherapy treatment of a patient after total knee replacement.

Vu, H., Dong, D., Cao, H., Verstraten, T., Lefeber, D., Vanderborght, B., & Geeroms, J. (2020). A review of gait phase detection algorithms for lower limb prostheses. *Sensors*, *20*(14), 3972.

Watanabe, S. (2023). Tree-structured parzen estimator: Understanding its algorithm components and their roles for better empirical performance. *arXiv preprint arXiv:2304.11127*.

XGBoost Developers. (2022). Xgboost parameters — xgboost 2.0.3 documentation [Online Document]. %5Curl%7Bhttps://xgboost.readthedocs.io/en/stable/parameter.html#parameters-for-tree-booster%7D

XGBoosting. (n.d.). *Most important xgboost hyperparameters to tune* [*Online Article*]. https://xgboosting.com/most-important-xgboost-hyperparameters-to-tune/

Yanti, Y., & Rahardiantoro, S. (2019). Stepwise approach in lagged variables time series modeling: A simple illustration. *IOP Conference Series: Materials Science and Engineering*, *621*(1), 012009.

Zhen, P., & et al. (2019). Walking gait phase detection based on acceleration signals using lstm-dnn algorithm. *Algorithms*, *12*(12), 253.

Figures

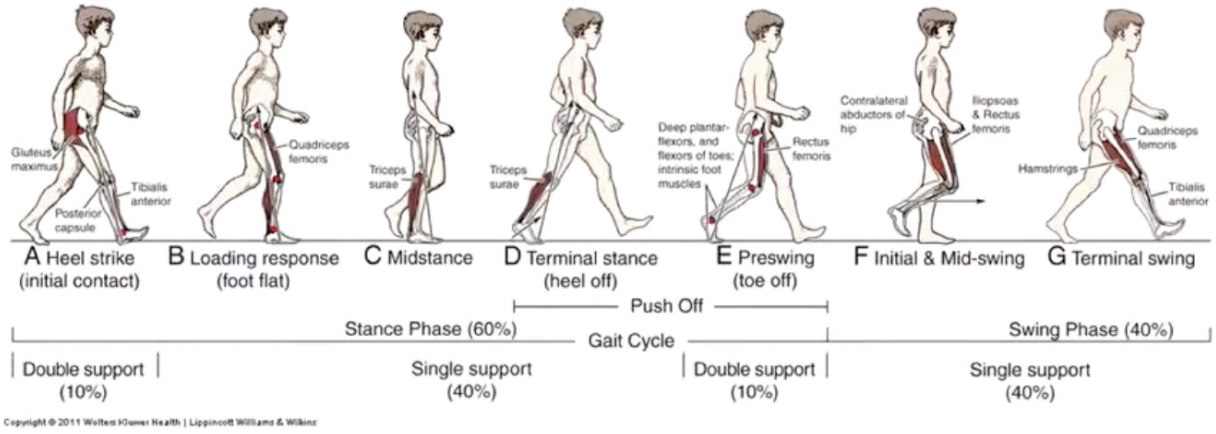


Figure 1: Data: Gait Event Cycle (Viktora, 2023)

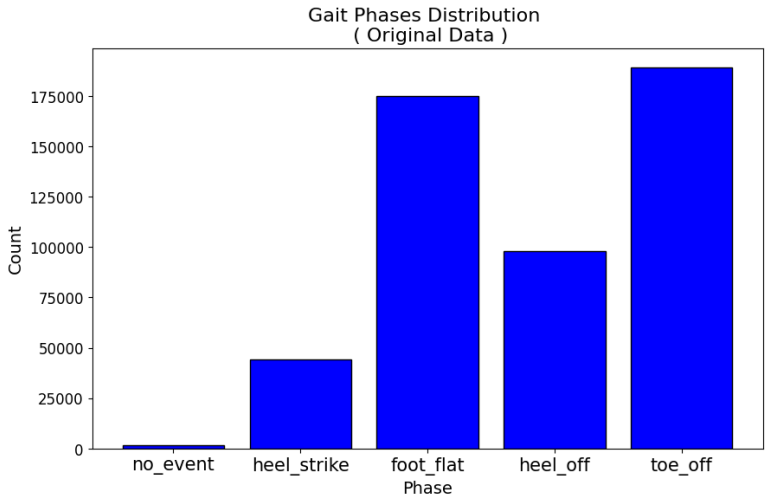
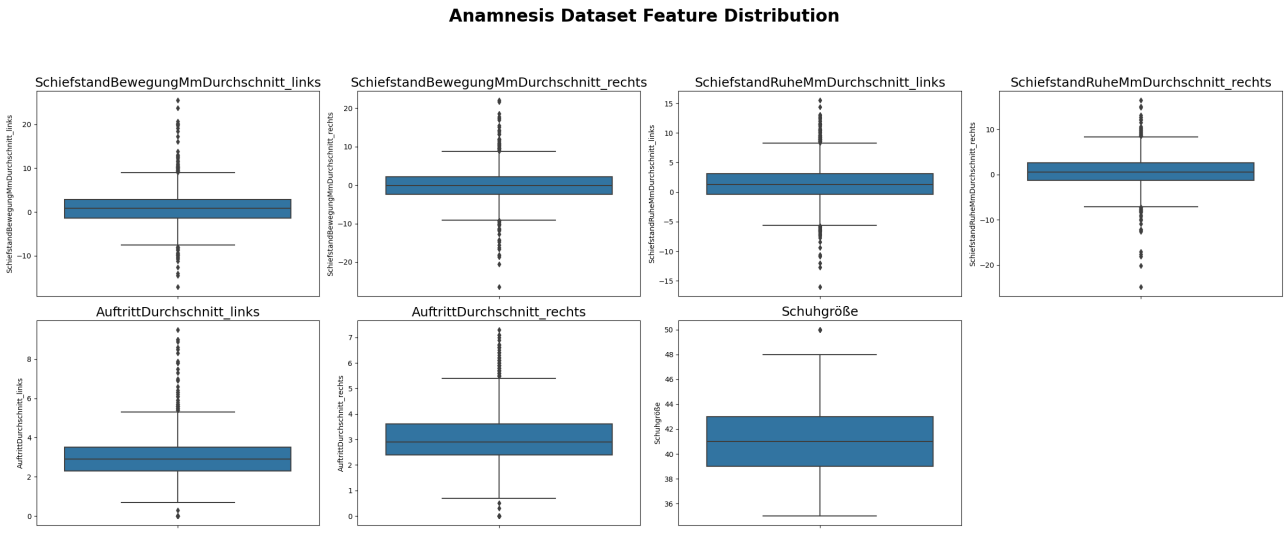
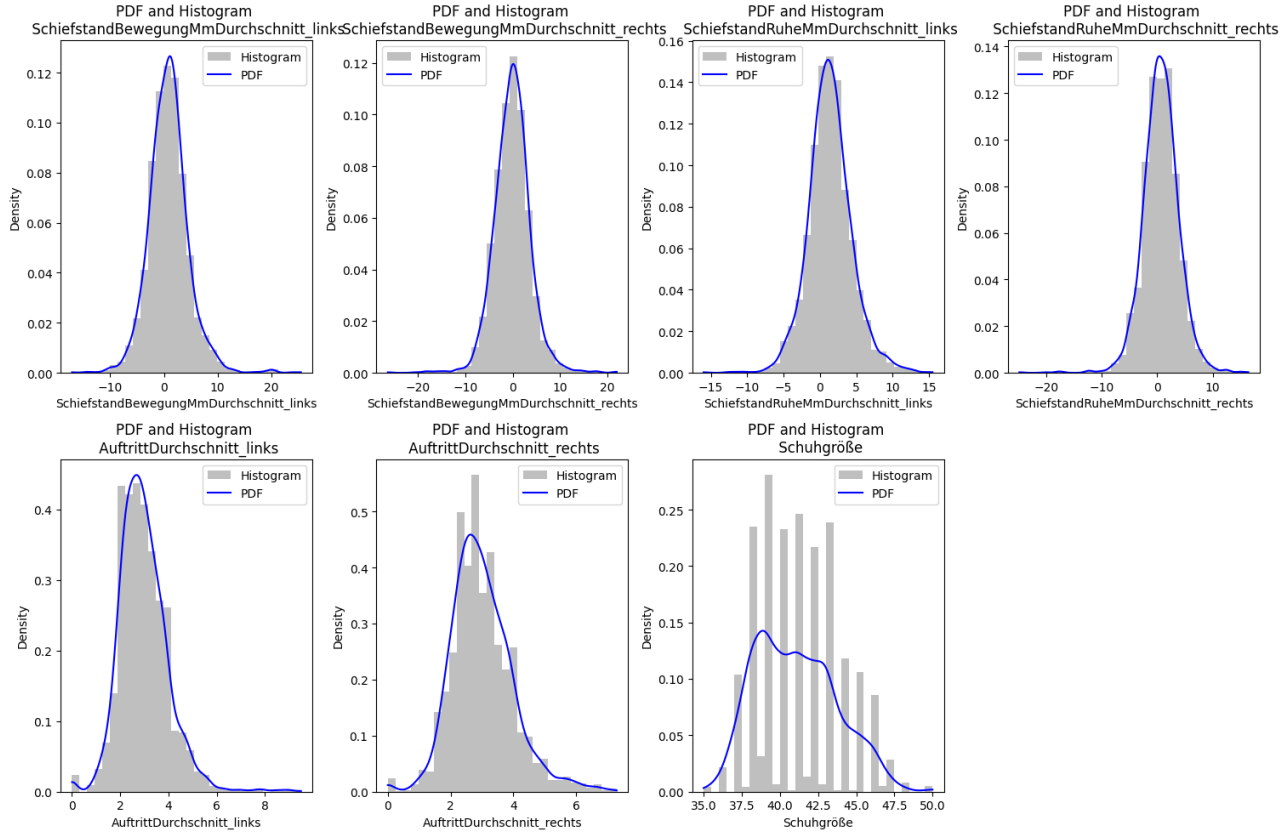


Figure 2: Data: Gait Phase Distribution (Original Data)



(a) Feature Distribution (Raw)



(b) Feature Distribution with PDF and Histogram

Figure 3: Data: Anamnesis Data Feature Distribution

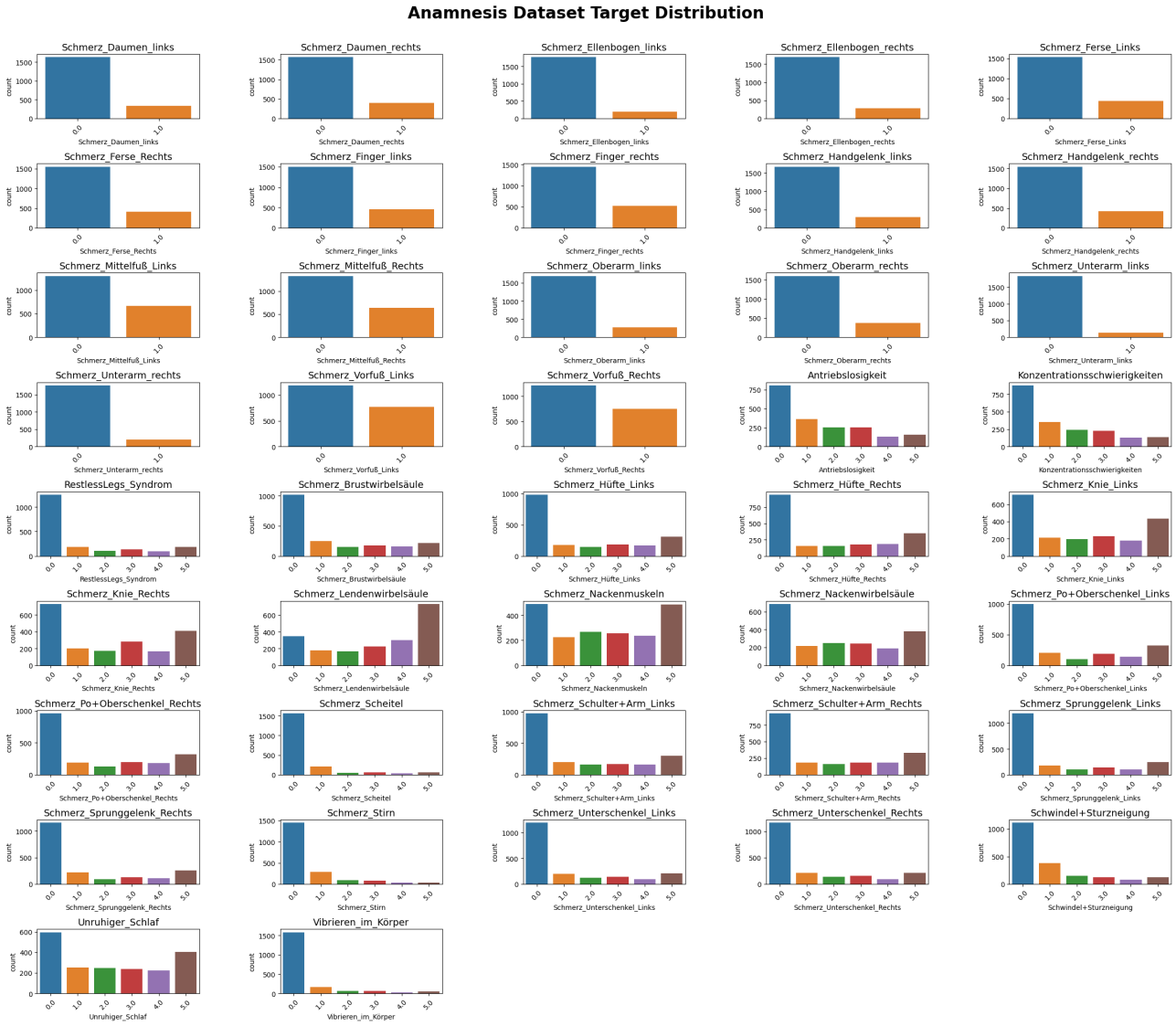


Figure 4: Data: Anamnesis Data Target Distribution

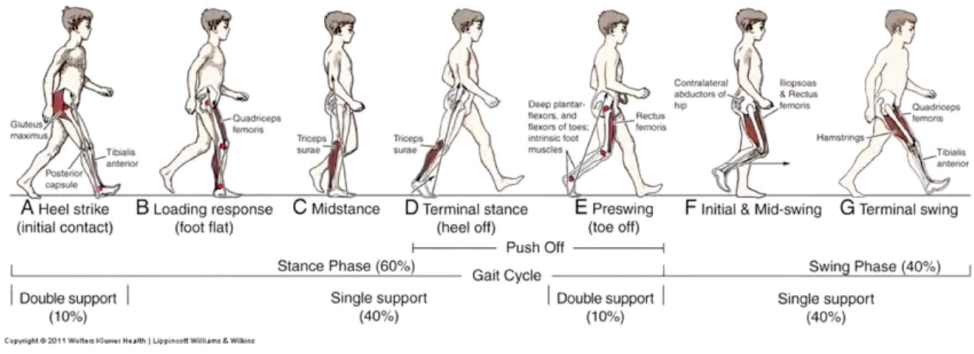


Figure 5: **Gait cycle with key events.** A typical human gait cycle, illustrating the sequence of gait events: Heel Strike (initial contact), Foot Flat (loading response), Heel Off (end of mid-stance), and Toe Off (pre-swing). The cycle then continues into the swing phase until the next heel strike.

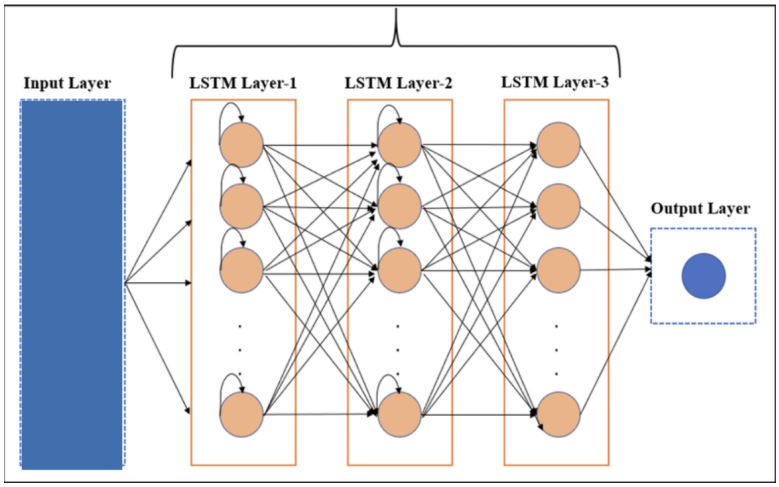


Figure 6: **LSTM Architecture for Gait Event Detection.** Schematic of the 3-layer LSTM model used for gait event detection. The model takes as input a time series of IMU sensor readings (3-axis acceleration and 3-axis gyroscope). It processes the sequence through stacked LSTM layers (each with 128 units), and outputs class probabilities for each time step (No event, Heel strike, Foot flat, Heel off, Toe off). Dropout layers (not shown for clarity) were applied between LSTM layers during training for regularization.

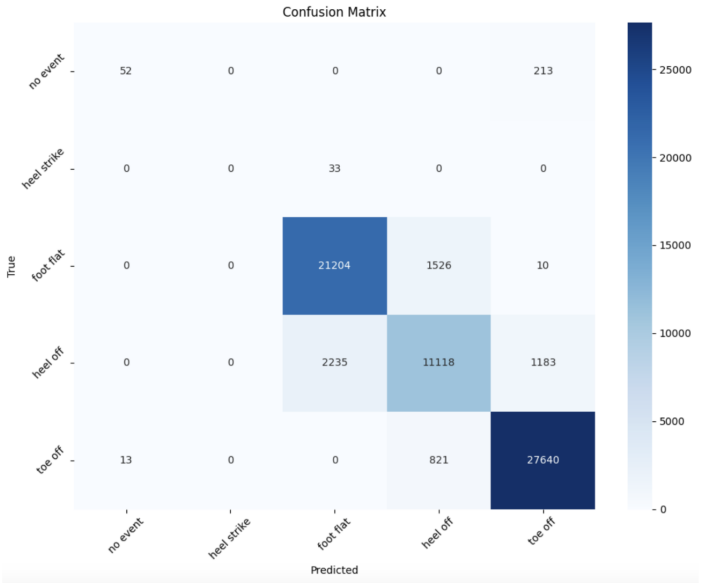


Figure 7: **Confusion Matrix for Gait Event Detection.** The confusion matrix of the LSTM model's predictions on the test set. Rows represent true classes and columns represent predicted classes. Each cell shows the count of time steps of a given true class that were predicted as a given class. The matrix highlights that most confusion occurred by misclassifying some *Heel Strike* instances as *Foot Flat* (since Heel Strike events were often missed), while the other classes have high true positive counts on the diagonal.

Figure 8: **Pain Anamnesis Input Example.** An example interface (Eversion app) for collecting pain anamnesis data. Patients report pain levels across multiple body regions. Some fields are binary and others use a numeric (Likert) scale for pain severity (0 = no pain to 5 = extreme pain). In our dataset, 42 such outputs were recorded for each patient (covering various joints, limbs, and symptoms).

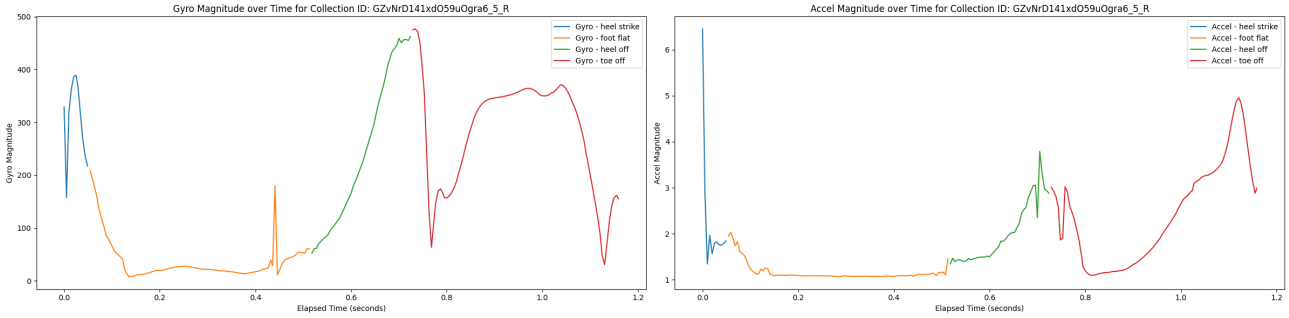


Figure 10: Gait Event-XGboost: Magnitudes of gyroscope data (left) and accelerometer data (right) collection ID : ”GZvNrD141xdO59uOgra6_5_R”.

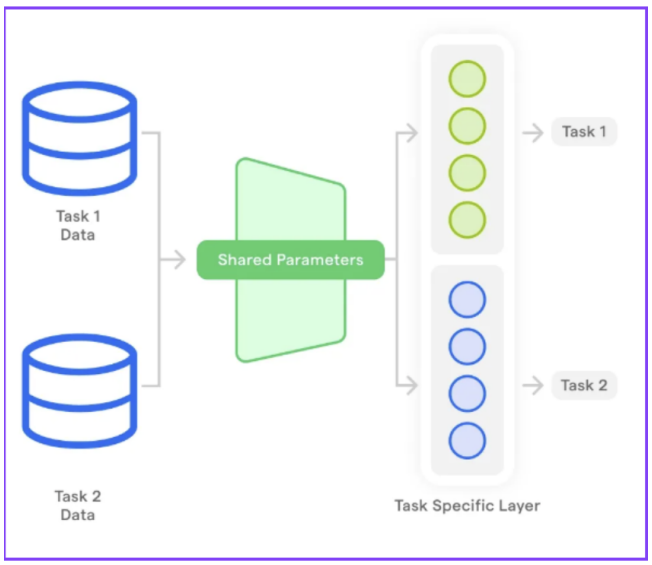


Figure 9: **Multi-Task Neural Network for Pain Classification.** Diagram of the shared-trunk multi-output network used for pain anamnesis classification. The input gait feature vector (7 features) passes through shared dense layers (the trunk). From the trunk, two sets of outputs are produced: 18 sigmoid outputs for binary pain indicators (one per region, indicating pain presence), and 24 groups of ordinal outputs (5 sigmoid neurons per group, each group corresponding to a region's pain severity thresholds 1–5). The ordinal outputs use the CORAL approach to infer a 0–5 pain level. During training, losses from both binary and ordinal outputs are combined to update the shared trunk, allowing the model to learn a representation beneficial to both tasks.

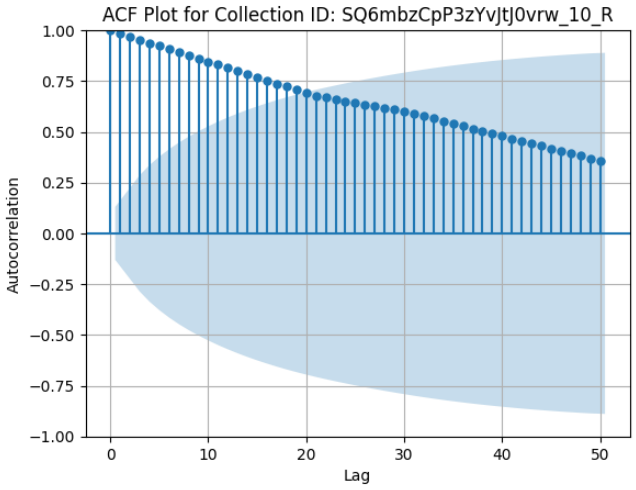


Figure 11: Gait Event-XGboost: Autocorrelation function for one data collection ID (blue region: 95% confidence interval of correlation at a lag K)

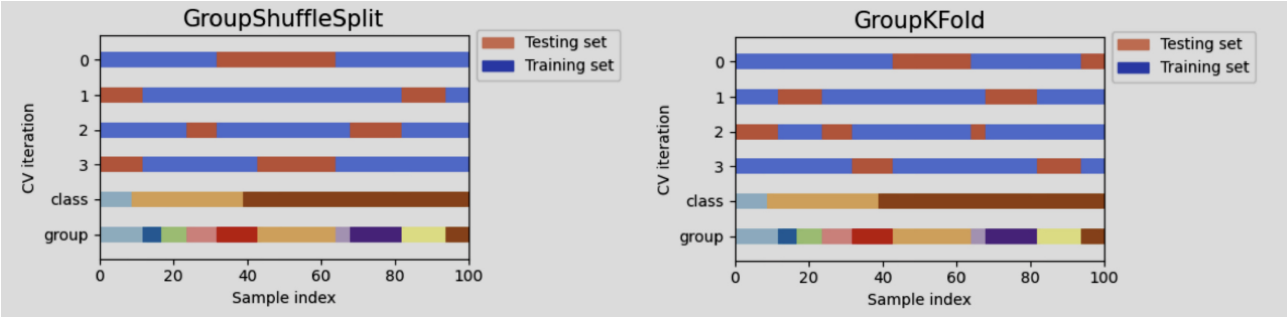


Figure 12: Gait Event-XGboost: GroupShuffleSplit vs GroupKFold comparison[19](The main difference is that GroupShuffleSPlit allows the same group to be used multiple times while GroupKFold allows only the distinct groups to be used for each fold.)

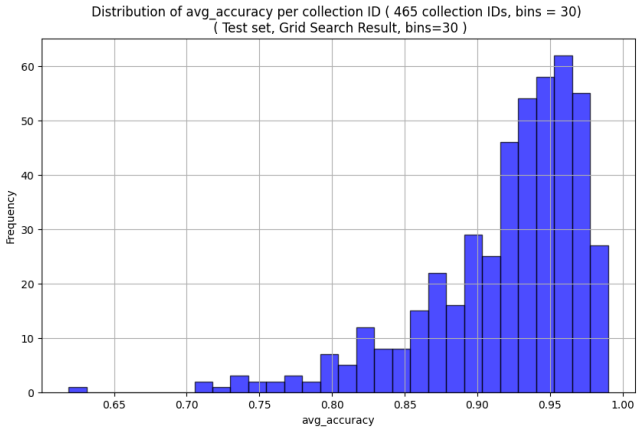


Figure 13: Gait Event-XGboost(Grid Search): Accuracy distribution per collection ID

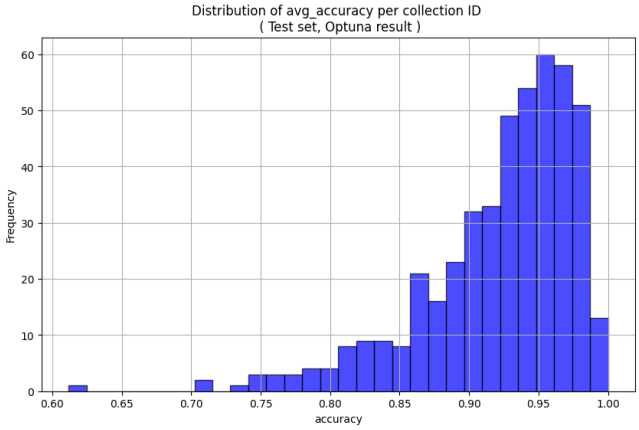


Figure 14: Gait Event-XGboost(Optuna): Accuracy distribution per collection ID

[19]https://scikit-learn.org/stable/auto_examples/model_selection/plot_cv_indices.html

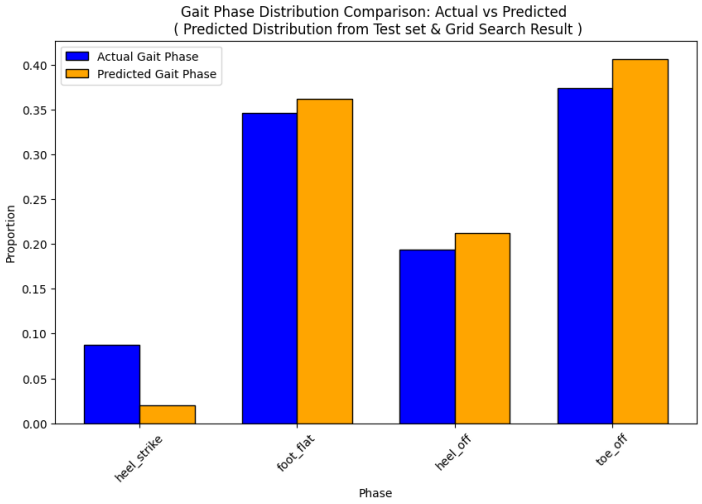


Figure 15: Gait Event-XGboost(Grid Search): actual vs predicted gait phases

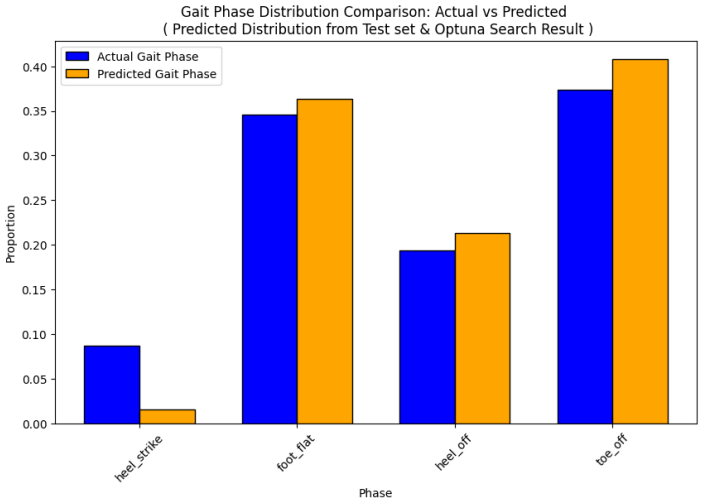


Figure 16: Gait Event-XGboost(Optuna): actual vs predicted gait phases

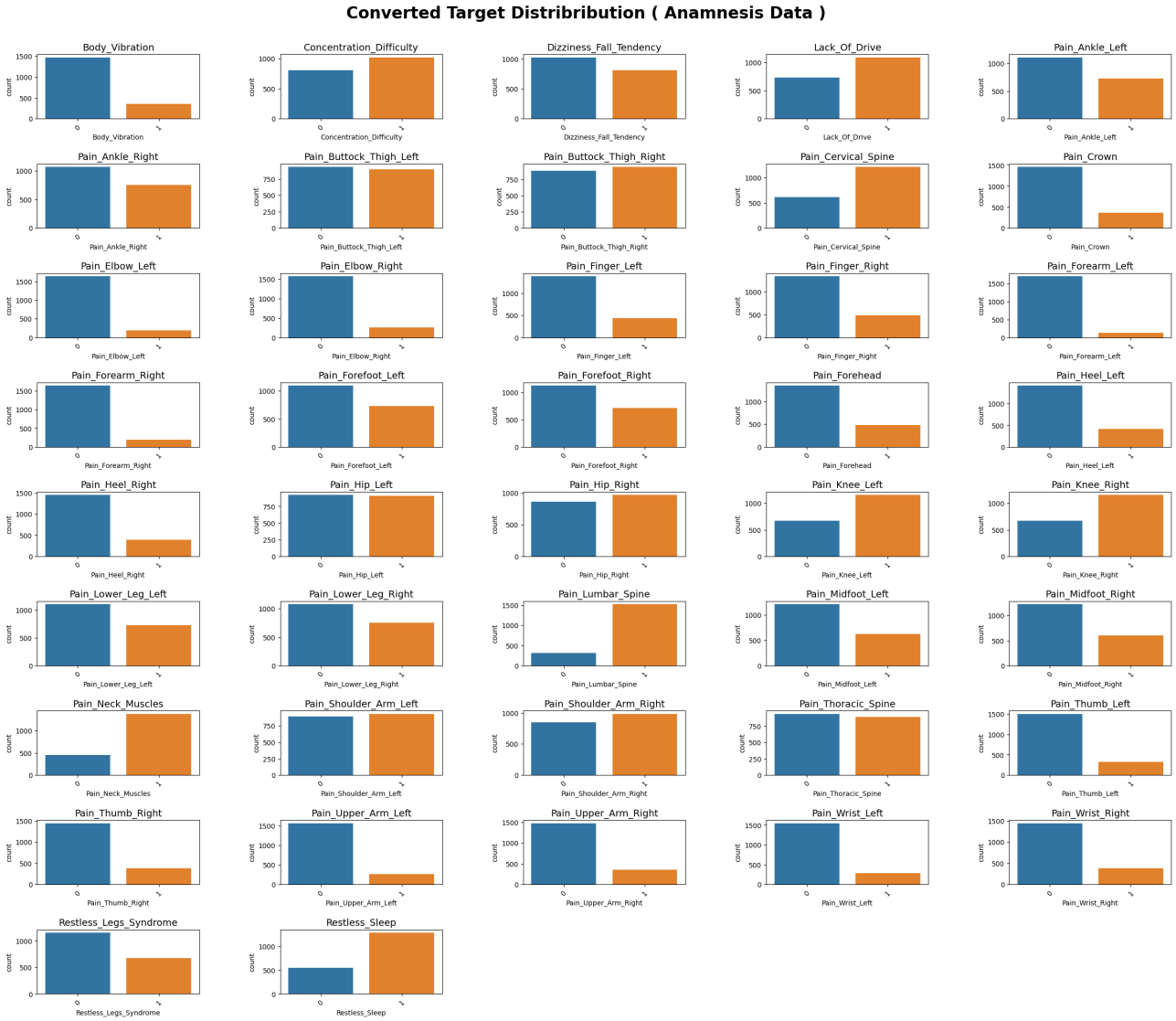


Figure 17: Anamnensis-XGBoost: Converted Target Distribution (Binary Target)

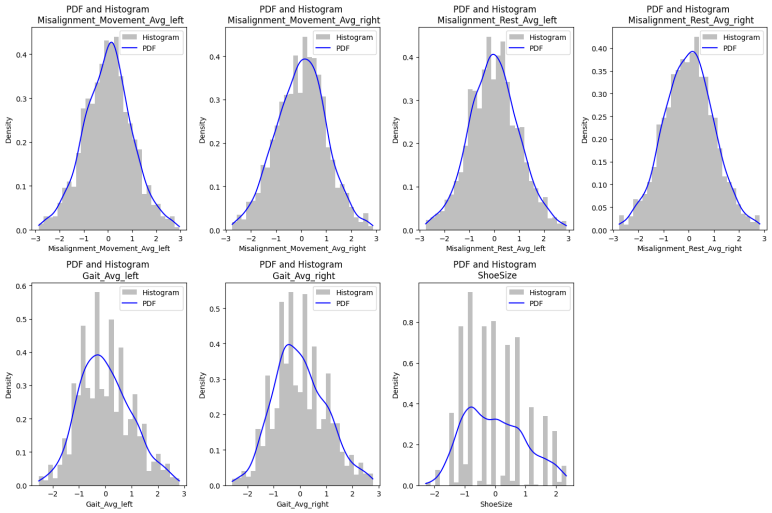


Figure 18: Anamnensis-XGBoost: Standardized feature distribution (rows with missing values are excluded)

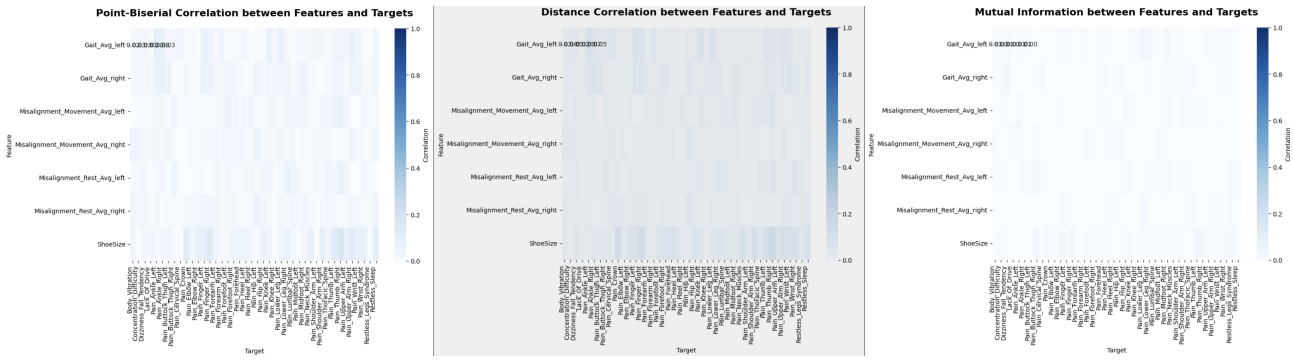


Figure 19: Anamnensis-XGBoost: Point-Biserial Correlation and MI between features and target values. Left to Right: Point-Biserial correlation, Distance correlation, Mutual Information (darker colors indicate stronger associations; lighter colors indicate values closer to zero).

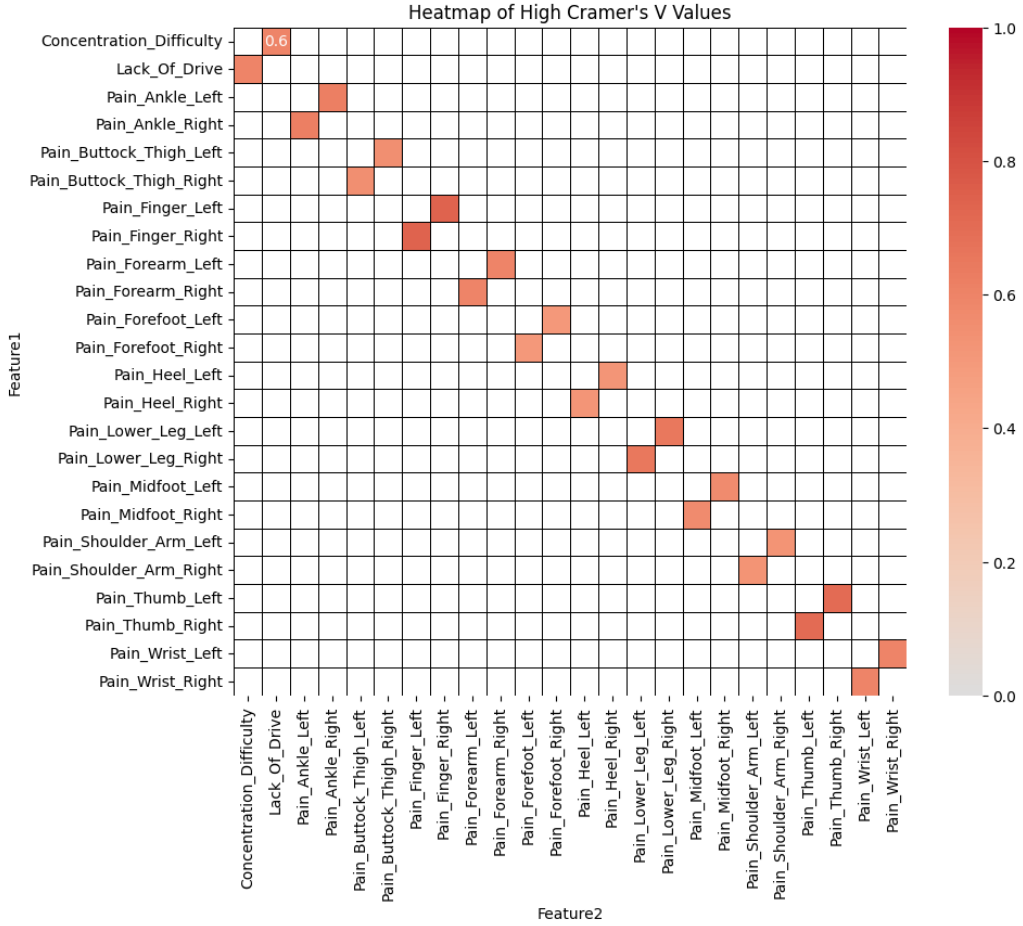


Figure 20: Anamnensis-XGBoost: Cramer's V between target variables. Only the pairs with Cramer's V greater than 0.4 are colored

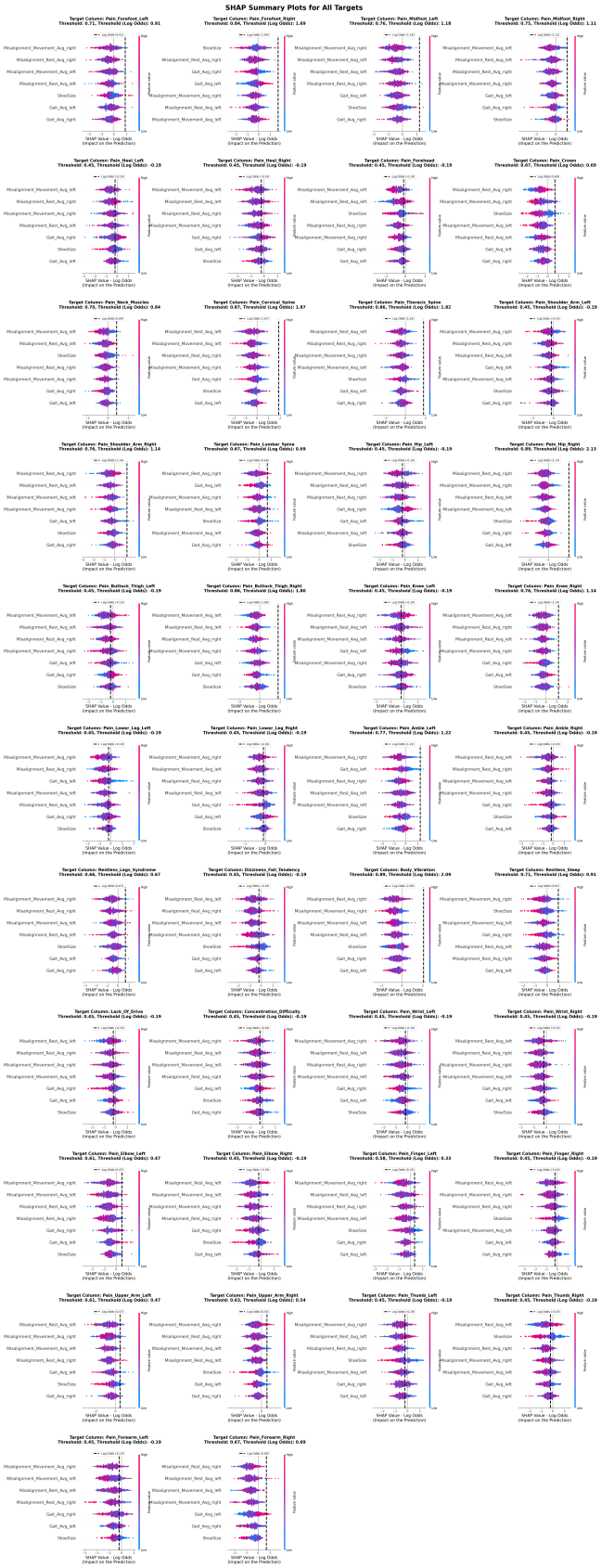


Figure 21: Anamnensis-XGBoost: SHAP Summary Plot (Each point in the plot represents a single data point from the feature variables. Red indicates high SHAP values, while blue indicates low SHAP values. The greater the SHAP value, the more influence that feature has on pushing the prediction in the direction shown along the X-axis.)

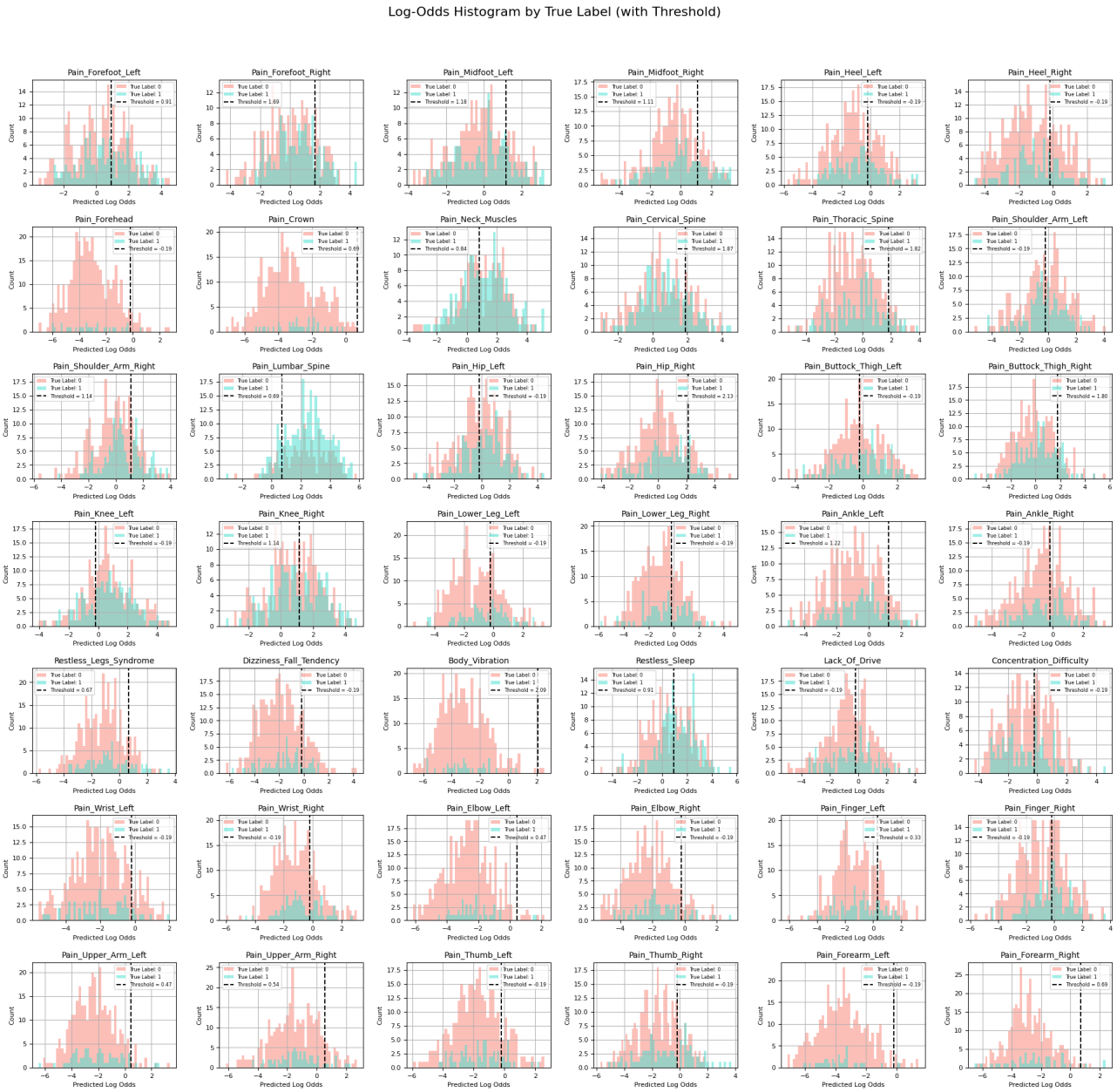


Figure 22: Anamnensis-XGBoost: Fanal model's predicted log-odds histogram with threshold for each target variables. (histograms in red are the data with actual true label(1) and the plots in green are the data with actual false label(0). The data on the right side of the thresholds(dashed line) are predicted as ture, and the data on the left side of the threshold are predicted zero. With validation accuracy as an object, the thresholds tend to be set with the higher values, increasing the false negative. The Optuna tried to find the balance between false negatives and false positives.

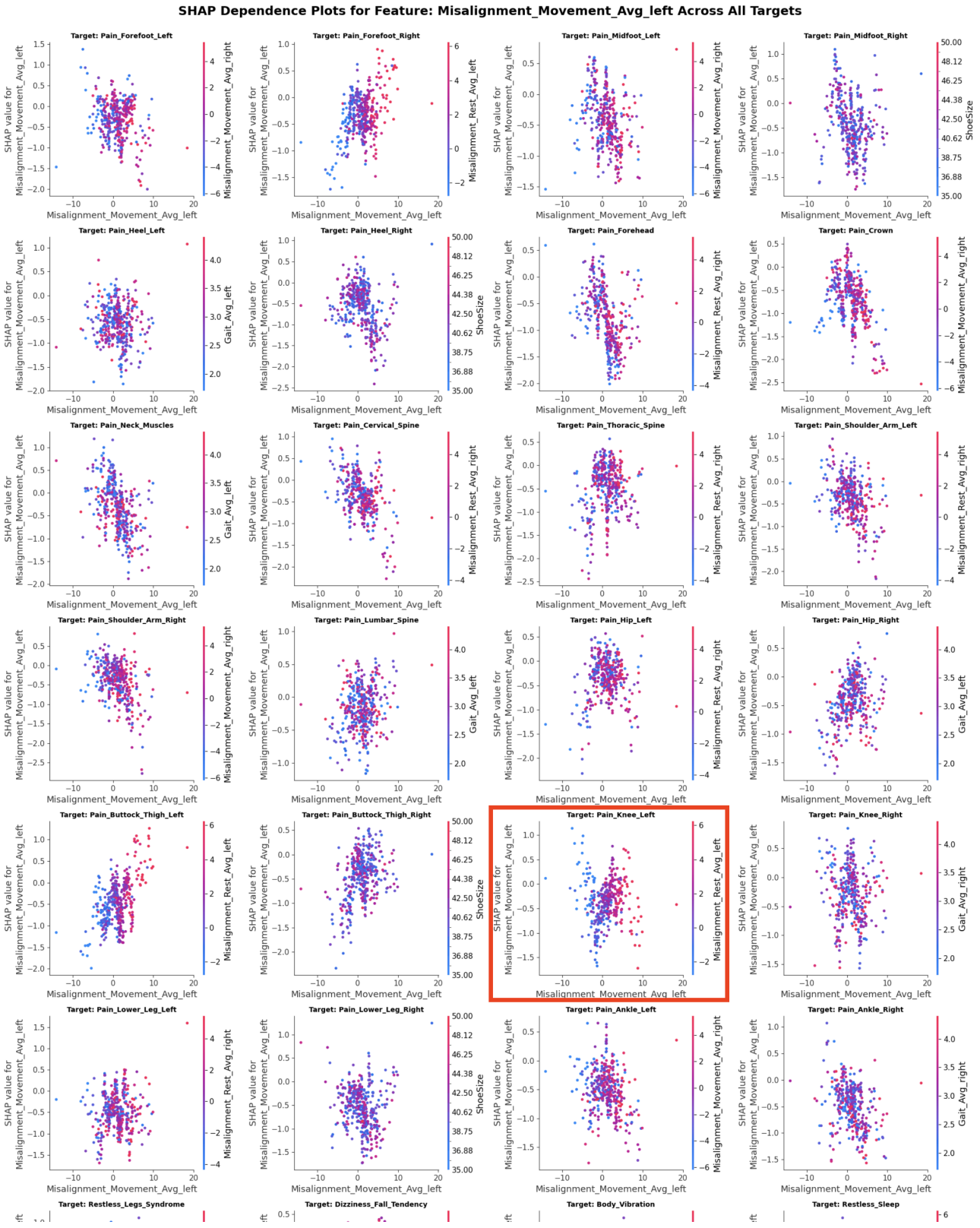


Figure 23: Anamnensis-XGBoost: SHAP dependence plot for Misalignment_Avg_left feature variables across all target variables. (red box: on the right side is the feature variable that has the largest interaction with Misalignment_Avg_left on the Pain_Knee_Left target variable. The colors in each point is Shapley values of the variable on the right side.

Tables

Index	collection_id	gyroscope_x	gyroscope_y	gyroscope_z	accelerometer_x	accelerometer_y	accelerometer_z	phase
0	vfn92aa0LcD2bzen2fN6_5_L	188.760	-158.620	-62.960	-1.525676	-2.760230	-8.625400	1
1	vfn92aa0LcD2bzen2fN6_5_L	269.720	-145.580	-162.430	-1.187636	-2.965912	-8.817513	1
2	vfn92aa0LcD2bzen2fN6_5_L	103.760	-334.530	-150.000	-2.154230	-2.208377	-9.178750	1
3	vfn92aa0LcD2bzen2fN6_5_L	83.360	-262.850	-160.850	-2.299520	-2.324800	-9.312470	1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
508445	r3kM0CfK4mY0zbt7z6_2_R	22.730	-30.670	-0.060	-1.822	-1.564	-7.476	4

Table 1: Data: Gait event raw dataset (508,446 total rows, 8 columns).

	SchiefstandBewegungMmDurchschnitt_links	SchiefstandBewegungMmDurchschnitt_rechts	SchiefstandRuheMmDurchschnitt_links	SchiefstandRuheMmDurchschnitt_rechts	AuftrittDurchschnitt_links	AuftrittDurchschnitt_rechts	Schuhgröße
count	2603.000	2603.000	2603.000	2603.000	2603.000	2603.000	1966.000
mean	0.875	-0.099	1.399	0.660	2.969	3.006	41.037
std	3.703	3.846	3.035	3.249	1.010	0.980	2.662
min	-17.100	-26.400	-16.000	-24.800	0.000	0.000	35.000
25%	-1.400	-2.400	-0.400	-1.300	2.300	2.400	39.000
50%	0.800	-0.100	1.300	0.600	2.900	2.900	41.000
75%	2.800	2.100	3.100	2.600	3.500	3.600	43.000
max	25.500	22.000	15.500	16.400	9.500	7.300	50.000

Table 2: Data: Anamnesis feature summary statistics

Severity	Antriebslosigkeit	Konzentrationsschwierigkeiten	RestlessLegs_Syndrom	Schmerz_Brustwirbelsäule	Schmerz_Hüfte_Links	Schmerz_Hüfte_Rechts	Schmerz_Knie_Links
0.0	808	879	128	1018	985	842	713
1.0	364	353	187	294	175	157	212
2.0	251	241	110	154	144	153	194
3.0	256	224	134	176	187	178	236
4.0	129	190	95	197	168	185	180
5.0	159	139	107	218	307	351	437

Severity	Schmerz_Knie_Rechts	Schmerz_Lendenwirbelsäule	Schmerz_Nackenmuskeln	Schmerz_Schulter+Arm_Links	Schmerz_Schulter+Arm_Rechts	Schmerz_Sprunggelenk_Links	Schmerz_Sprunggelenk_Rechts
0.0	790	585	490	971	926	132	114
1.0	203	180	227	198	192	175	220
2.0	175	170	268	181	181	181	105
3.0	281	228	267	171	181	139	125
4.0	168	304	236	164	182	106	112
5.0	411	735	486	300	334	249	261

Severity	Schmerz_Unterschenkel_Links	Schmerz_Unterschenkel_Rechts	Schwindel+Sturzneigung	Unruhiger_Schlaf	Vibrieren_im_Körper
0.0	118	119	112	595	155
1.0	203	209	379	254	167
2.0	125	138	149	247	67
3.0	144	159	126	239	60
4.0	97	91	81	224	30
5.0	209	212	122	408	57

(a) Anamnesis non-binary target distribution (value counts of 0–5)

Value	Schmerz_Daumen_links	Schmerz_Daumen_rechts	Schmerz_Ellenbogen_links	Schmerz_Ellenbogen_rechts	Schmerz_Ferse_Links	Schmerz_Ferse_Rechts
0.0	1626	1563	1765	1659	1526	1351
1.0	340	403	201	277	440	415

Value	Schmerz_Finger_links	Schmerz_Finger_rechts	Schmerz_Handgelenk_links	Schmerz_Handgelenk_rechts	Schmerz_Mittelfuß_Links	Schmerz_Mittelfuß_Rechts
0.0	1504	1450	1669	1547	1300	1328
1.0	462	516	297	419	686	638

Value	Schmerz_Oberarm_links	Schmerz_Oberarm_rechts	Schmerz_Unterarm_links	Schmerz_Unterarm_rechts	Schmerz_Vorfuß_Links	Schmerz_Vorfuß_Rechts
0.0	1694	1598	1832	1767	1192	1218
1.0	272	368	134	199	774	748

(b) Anamnesis binary target distribution (value counts of 0–1)

Table 3: Data: Anamensis Target variable distributions: (a) ordinal (non-binary) and (b) binary pain indicators.

Gait Event	Precision	Recall	F1-Score	Support
No Event	0.84	0.21	0.34	265
Heel Strike	0.00	0.00	0.00	33
Foot Flat	0.90	0.94	0.92	22,740
Heel Off	0.82	0.77	0.80	14,536
Toe Off	0.96	0.97	0.96	28,474
Accuracy	–	–	**0.91**	66,048 (total)

Table 4: **Gait Event Detection Performance per Class.** Classification report on the test set for the LSTM gait event detection model. Precision, recall, and F_1 score are given for each gait event class, along with the support (number of true instances of each class in the test data). The model excels on the frequent classes (Foot Flat, Toe Off) but struggled on the rare Heel Strike class, as evidenced by the 0% recall for Heel Strike (only 33 instances in test set). Overall accuracy (weighted by support) is 91%.

	Binary Accuracy	Ordinal MAE (0-5 scale)
Binary-only NN model	78%	–
Multi-task (Binary+Ordinal) NN model	80%	1.4

Table 5: **Pain Classification Results (Binary vs Multi-Task).** Comparison of performance between a model trained only on binary pain classification and the multi-task model (binary+ordinal) on the test set. The multi-task model achieves slightly higher accuracy on the binary pain predictions and additionally provides ordinal severity predictions (with an average MAE of 1.4).

Index	collection_id	gyroscope_x_lag1	gyroscope_x_lag2	gyroscope_x_lag3	gyroscope_x_lag4	⋯	accelerometer_z_lag16	accelerometer_z_lag17	accelerometer_z_lag18	accelerometer_z_lag19	phase
0	vh92aaJeQLxDazem…	0.573279	0.574568	0.573335	0.569859	⋯	0.490814	0.563660	0.471328	0.219917	0
1	vh92aaJeQLxDazem…	0.568659	0.573279	0.574568	0.573335	⋯	0.477446	0.490814	0.563660	0.471328	0
2	vh92aaJeQLxDazem…	0.573279	0.574568	0.573335	0.569859	⋯	0.490814	0.563660	0.471328	0.219917	1
3	vh92aaJeQLxDazem…	0.568659	0.573279	0.574568	0.573335	⋯	0.477446	0.490814	0.563660	0.471328	1
4	vh92aaJeQLxDazem…	0.565710	0.568569	0.573279	0.574568	⋯	0.474859	0.477446	0.490814	0.563660	1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
464585	rbk3m0CFK4mQVt2…	0.483488	0.479031	0.491057	0.518334	⋯	0.436954	0.433309	0.431450	0.431752	3
464586	rbk3m0CFK4mQVt2…	0.500392	0.483488	0.479031	0.491057	⋯	0.441797	0.436954	0.433309	0.431450	3
464587	rbk3m0CFK4mQVt2…	0.524557	0.500392	0.483488	0.479031	⋯	0.447386	0.441797	0.436954	0.433309	3
464588	rbk3m0CFK4mQVt2…	0.547853	0.524557	0.500392	0.483488	⋯	0.454835	0.447386	0.441797	0.436954	3
464589	rbk3m0CFK4mQVt2…	0.566368	0.547853	0.524557	0.500392	⋯	0.467155	0.454835	0.447386	0.441797	3

Table 6: Gait Event-XGboost: gait event dataset with lagged features (464,590 total rows, 116 columns).

Hyperparameter	Type	Optuna Hyperparameter Range	Parameter Grid (Grid Search)	Default
early_stopping_rounds	integer	100	100	∞(no default value)
learning_rate	float	$[0.01, 0.3]$	$0.01, 0.05, 0.1$	0.1
n_estimator	integer	$[100, 500]$	$100, 200, 300$	100
max_depth	integer	$[3, 10]$	$6, 8, 10$	6
subsample	float	$[0.5, 1.0]$	$0.8, 1$	1
colsample_bytree	float	$[0.5, 1.0]$	$0.8, 1$	1
gamma	float	$(0, 5]$	$0, 1$	0

Table 7: Gait Event-XGboost: Predefined XGBoost hyperparameter range and hyperparameter grid for gait event detection

Gait Phase	precision	recall	F1-score	Support
Heel Strike	0.68	0.60	0.64	1,849
Foot Flat	0.91	0.94	0.92	33,777
Heel Off	0.86	0.81	0.83	19,754
Toe Off	0.97	0.97	0.97	37,920

Table 8: Gait Event-XGboost: performance summary on the test set (Grid Search)

Gait Phase	precision	recall	F1-score	Support
Heel Strike	0.71	0.53	0.61	1,423
Foot Flat	0.92	0.93	0.92	33,698
Heel Off	0.87	0.80	0.83	19,807
Toe Off	0.95	0.95	0.96	38,372

Table 9: Gait Event-XGboost: performance summary on the test set (Optuna)

Hyperparameter	Type	Optuna Hyperparameter Range	Default
early_stopping_rounds	integer	100	∞(no default value)
scale_pos_weight	float	$[1, 10]$	1
learning_rate	float	$[0.01, 0.3]$	0.1
n_estimator	integer	$[100, 500]$	100
max_depth	integer	$[3, 10]$	6
subsample	float	$[0.5, 1.0]$	1
colsample_bytree	float	$[0.5, 1.0]$	1
gamma	float	$(0, 5]$	0

Table 10: Anamnesis-XGboost: Predefined XGBoost hyperparameter range and hyperparameter grid for anamnesis analysis

target_column	best_accuracy	best_recall	best_f1_score	best_auc	best_aucpr	best_threshold	best_precision	early_stopping_rounds	max_depth	learning_rate	n_estimators	subsample	colsample_bytree	scale_pos_weight	gamma	threshold
Pain_Forefoot_Left	0.634921	0.129032	0.217687	0.575726	0.464650	0.713488	0.695652	87	3	0.039749	799	0.897361	0.632604	4	1.563624	0.713488
Pain_Forefoot_Right	0.673016	0.125242	0.201560	0.604568	0.417983	0.844381	0.565217	78	3	0.026033	598	0.711673	0.782361	4	0.853382	0.844381
Pain_Midfoot_Left	0.695238	0.089109	0.157895	0.550106	0.413741	0.764435	0.692308	82	5	0.096471	339	0.998889	0.888056	3	0.759070	0.764435
Pain_Midfoot_Right	0.714286	0.000000	0.000000	0.600049	0.356593	0.752584	0.000000	76	8	0.032435	645	0.700213	0.667002	2	1.965562	0.752584
Pain_Heel_Left	0.787302	0.000000	0.000000	0.557342	0.241604	0.452122	0.000000	80	4	0.019107	775	0.942519	0.791384	1	1.368466	0.452122
Pain_Heel_Right	0.777778	0.000000	0.000000	0.515190	0.245307	0.452122	0.000000	80	4	0.019107	775	0.942519	0.791384	1	1.368466	0.452122
Pain_Forehead	0.923810	0.000000	0.000000	0.607245	0.094951	0.452122	0.000000	80	4	0.019107	775	0.942519	0.791384	1	1.368466	0.452122
Pain_Crown	0.933333	0.047619	0.086957	0.563045	0.121456	0.666458	0.500000	69	10	0.112479	599	0.746798	0.717425	9	1.202230	0.666458
Pain_Neck_Muscles	0.561905	0.645570	0.596491	0.573248	0.559459	0.698182	0.554348	67	6	0.048749	741	0.891105	0.826841	3	1.915866	0.698182
Pain_Cervical_Spine	0.612698	0.134921	0.217949	0.551356	0.466074	0.866981	0.566667	84	10	0.127557	727	0.994129	0.749893	6	0.375925	0.866981
Pain_Thoracic_Spine	0.714286	0.046516	0.117647	0.537813	0.364422	0.860106	0.666667	80	4	0.095630	671	0.802686	0.729222	7	1.146055	0.860106
Pain_Shoulder_Arm_Left	0.685714	0.000000	0.000000	0.519033	0.337045	0.452122	0.000000	80	4	0.019107	775	0.942519	0.791384	1	1.368466	0.452122
Pain_Shoulder_Arm_Right	0.657143	0.087719	0.155250	0.554988	0.449460	0.757588	0.714286	63	5	0.067110	414	0.706274	0.776299	3	1.488760	0.757588
Pain_Lumbar_Spine	0.631746	1.000000	0.774390	0.490946	0.623319	0.666458	0.631746	69	10	0.112479	599	0.746798	0.717425	9	1.202230	0.666458
Pain_Hip_Left	0.660317	0.000000	0.000000	0.530935	0.360060	0.452122	0.000000	80	4	0.019107	775	0.942519	0.791384	1	1.368466	0.452122
Pain_Hip_Right	0.666667	0.090909	0.160000	0.555851	0.431887	0.893407	0.714286	71	6	0.146337	583	0.773084	0.759034	4	1.454025	0.893407
Pain_Buttock_Thigh_Left	0.653492	0.000000	0.000000	0.538368	0.381998	0.452122	0.000000	80	4	0.019107	775	0.942519	0.791384	1	1.368466	0.452122
Pain_Buttock_Thigh_Right	0.666667	0.018868	0.036697	0.527626	0.380160	0.858592	0.666667	70	5	0.043534	489	0.835468	0.701527	4	1.693992	0.858592
Pain_Knee_Left	0.619048	0.459259	0.508197	0.598313	0.521742	0.452122	0.568807	80	4	0.019107	775	0.942519	0.791384	1	1.368466	0.452122
Pain_Knee_Right	0.571429	0.208333	0.307692	0.544368	0.494368	0.757588	0.588235	63	5	0.067110	414	0.706274	0.776299	3	1.488760	0.757588
Pain_Lower_Leg_Left	0.777778	0.000000	0.000000	0.545656	0.258499	0.452122	0.000000	80	4	0.019107	775	0.942519	0.791384	1	1.368466	0.452122
Pain_Lower_Leg_Right	0.790476	0.000000	0.000000	0.546063	0.233736	0.452122	0.000000	80	4	0.019107	775	0.942519	0.791384	1	1.368466	0.452122
Pain_Ankle_Left	0.746032	0.024691	0.047619	0.584335	0.388590	0.771581	0.666667	90	6	0.028140	583	0.804590	0.822216	2	0.990322	0.771581
Pain_Ankle_Right	0.755556	0.000000	0.000000	0.518635	0.243081	0.452122	0.000000	80	4	0.019107	775	0.942519	0.791384	1	1.368466	0.452122
Restless_Legs_Syndrome	0.787302	0.028986	0.056338	0.586898	0.315867	0.662233	1.000000	100	10	0.010581	764	0.926972	0.791077	2	0.739635	0.662233
Dizziness_Fall_Tendency	0.828571	0.000000	0.000000	0.584991	0.171567	0.452122	0.000000	80	4	0.019107	775	0.942519	0.791384	1	1.368466	0.452122
Body_Vibration	0.936508	0.047619	0.090909	0.529802	0.126464	0.889510	1.000000	69	5	0.126023	478	0.862809	0.742277	9	0.149101	0.889510
Restless_Sleep	0.587302	0.649351	0.606061	0.599258	0.570001	0.713874	0.568182	77	7	0.093276	536	0.734435	0.825212	7	1.563243	0.713874
Lack_Of_Drive	0.771429	0.000000	0.000000	0.475766	0.216888	0.452122	0.000000	80	4	0.019107	775	0.942519	0.791384	1	1.368466	0.452122
Concentration_Difficulty	0.755556	0.000000	0.000000	0.515197	0.277171	0.452122	0.000000	80	4	0.019107	775	0.942519	0.791384	1	1.368466	0.452122
Pain_Wrist_Left	0.850794	0.000000	0.000000	0.527390	0.177422	0.452122	0.000000	80	4	0.019107	775	0.942519	0.791384	1	1.368466	0.452122
Pain_Wrist_Right	0.803175	0.000000	0.000000	0.563622	0.220216	0.452122	0.000000	80	4	0.019107	775	0.942519	0.791384	1	1.368466	0.452122
Pain_Elbow_Left	0.895238	0.029412	0.057143	0.483148	0.133182	0.614970	1.000000	79	3	0.082525	655	0.972299	0.987931	3	1.901230	0.614970
Pain_Elbow_Right	0.876190	0.000000	0.000000	0.543943	0.137817	0.452122	0.000000	80	4	0.019107	775	0.942519	0.791384	1	1.368466	0.452122
Pain_Finger_Left	0.806349	0.040129	0.031746	0.579139	0.265031	0.582930	1.000000	59	5	0.146479	395	0.912678	0.759191	2	0.799146	0.582930
Pain_Finger_Right	0.771429	0.000000	0.000000	0.541009	0.285538	0.452122	0.000000	80	4	0.019107	775	0.942519	0.791384	1	1.368466	0.452122
Pain_Upper_Arm_Left	0.876190	0.075000	0.133333	0.548455	0.208461	0.614970	0.600000	79	3	0.082525	655	0.972299	0.987931	3	1.901230	0.614970
Pain_Upper_Arm_Right	0.819048	0.066574	0.123077	0.584807	0.325836	0.632144	0.400000	81	3	0.082445	655	0.988179	0.997285	3	1.920067	0.632144
Pain_Thumb_Left	0.819048	0.000000	0.000000	0.594665	0.230473	0.452122	0.000000	80	4	0.019107	775	0.942519	0.791384	1	1.368466	0.452122
Pain_Thumb_Right	0.803175	0.000000	0.000000	0.606719	0.253069	0.452122	0.000000	80	4	0.019107	775	0.942519	0.791384	1	1.368466	0.452122
Pain_Forearm_Left	0.942857	0.000000	0.000000	0.583474	0.077707	0.452122	0.000000	80	4	0.019107	775	0.942519	0.791384	1	1.368466	0.452122
Pain_Forearm_Right	0.911111	0.035714	0.066667	0.491289	0.125335	0.666458	0.500000	69	10	0.112479	599	0.746798	0.717425	9	1.202230	0.666458

Table 11: Anamnesis-XGBoost: Hyperparameter optimization results for pain classification models

Target	Accuracy	Recall	Precision	F1_Score	AUC	AUCPR
Pain_Forefoot_Left	0.553299	0.428571	0.428571	0.428571	0.558279	0.452137
Pain_Forefoot_Right	0.578680	0.202532	0.444444	0.278261	0.583619	0.480054
Pain_Midfoot_Left	0.593909	0.202797	0.386667	0.266055	0.530048	0.408793
Pain_Midfoot_Right	0.642132	0.267176	0.437500	0.331754	0.570865	0.424200
Pain_Heel_Left	0.609137	0.320755	0.293103	0.306306	0.545303	0.317134
Pain_Heel_Right	0.621827	0.258824	0.203704	0.227979	0.530249	0.231172
Pain_Forehead	0.883249	0.033333	0.055556	0.041667	0.499542	0.089423
Pain_Crown	0.906091	0.000000	0.000000	0.000000	0.605496	0.119292
Pain_Neck_Muscles	0.515228	0.573529	0.529412	0.550588	0.505289	0.546046
Pain_Cervical_Spine	0.538071	0.206250	0.375000	0.266129	0.519284	0.423390
Pain_Thoracic_Spine	0.713198	0.121495	0.406250	0.187050	0.588883	0.355201
Pain_Shoulder_Arm_Left	0.487310	0.482270	0.345178	0.402367	0.497239	0.338939
Pain_Shoulder_Arm_Right	0.637056	0.333333	0.489583	0.396624	0.610153	0.457790
Pain_Lumbar_Spine	0.614213	0.866667	0.651917	0.744108	0.513218	0.659343
Pain_Hip_Left	0.510152	0.617188	0.354260	0.450142	0.527931	0.342360
Pain_Hip_Right	0.647208	0.175573	0.425926	0.248649	0.583403	0.386004
Pain_Buttock_Thigh_Left	0.558376	0.548872	0.390374	0.456250	0.544551	0.364254
Pain_Buttock_Thigh_Right	0.647208	0.182482	0.480769	0.264550	0.574086	0.396265
Pain_Knee_Left	0.482234	0.756098	0.430556	0.548673	0.521209	0.426810
Pain_Knee_Right	0.522843	0.446667	0.389535	0.416149	0.528962	0.434765
Pain_Lower_Leg_Left	0.675127	0.452381	0.316667	0.372549	0.624923	0.295760
Pain_Lower_Leg_Right	0.644670	0.383721	0.275000	0.320388	0.584755	0.279958
Pain_Ankle_Left	0.725888	0.115789	0.314286	0.169231	0.568139	0.293346
Pain_Ankle_Right	0.583756	0.430233	0.243421	0.310924	0.520915	0.228366
Restless_Legs_Syndrome	0.751269	0.187500	0.312500	0.234375	0.573806	0.297032
Dizziness_Fall_Tendency	0.703046	0.186667	0.200000	0.193103	0.550637	0.211004
Body_Vibration	0.903553	0.000000	0.000000	0.000000	0.535537	0.103677
Restless_Sleep	0.550761	0.554286	0.494898	0.522911	0.605245	0.524173
Lack_Of_Drive	0.560914	0.467890	0.307229	0.370909	0.518300	0.298627
Concentration_Difficulty	0.558376	0.393939	0.254902	0.309524	0.472248	0.261907
Pain_Wrist_Left	0.758883	0.116667	0.142857	0.128440	0.509381	0.171375
Pain_Wrist_Right	0.682741	0.333333	0.275510	0.301676	0.583955	0.245678
Pain_Elbow_Left	0.873096	0.044444	0.222222	0.074074	0.560904	0.140650
Pain_Elbow_Right	0.786802	0.136364	0.250000	0.176471	0.596914	0.233389
Pain_Finger_Left	0.675127	0.208333	0.277778	0.238095	0.533452	0.257362
Pain_Finger_Right	0.588832	0.480769	0.316456	0.381679	0.590385	0.313443
Pain_Upper_Arm_Left	0.812183	0.046875	0.187500	0.075000	0.538731	0.205818
Pain_Upper_Arm_Right	0.748731	0.120000	0.214286	0.153846	0.527900	0.204849
Pain_Thumb_Left	0.725888	0.200000	0.212121	0.205882	0.537125	0.194558
Pain_Thumb_Right	0.708122	0.377778	0.365591	0.371585	0.604459	0.395825
Pain_Forearm_Left	0.888325	0.055556	0.166667	0.083333	0.460583	0.089872
Pain_Forearm_Right	0.880711	0.083333	0.571429	0.145455	0.587247	0.227428

Table 12: Anamnesis-XGBoost: Final model results for pain classification models

Appendix

A.1 Pain Anamnesis - XGBoost: column name English mapping

German	English
Feature Variables	
SchiefstandBewegungMmDurchschnitt_links	Misalignment_Movement_Avg_left
SchiefstandBewegungMmDurchschnitt_rechts	Misalignment_Movement_Avg_right
SchiefstandRuheMmDurchschnitt_links	Misalignment_Rest_Avg_left
SchiefstandRuheMmDurchschnitt_rechts	Misalignment_Rest_Avg_right
AuftrittDurchschnitt_links	Gait_Avg_left
AuftrittDurchschnitt_rechts	Gait_Avg_right
Schuhgröße	ShoeSize
Target Variables	
Schmerz_Vorfuß_Links	Pain_Forefoot_Left
Schmerz_Vorfuß_Rechts	Pain_Forefoot_Right
Schmerz_Mittelfuß_Links	Pain_Midfoot_Left
Schmerz_Mittelfuß_Rechts	Pain_Midfoot_Right
Schmerz_Ferse_Links	Pain_Heel_Left
Schmerz_Ferse_Rechts	Pain_Heel_Right
Schmerz_Handgelenk_links	Pain_Wrist_Left
Schmerz_Handgelenk_rechts	Pain_Wrist_Right
Schmerz_Ellenbogen_links	Pain_Elbow_Left
Schmerz_Ellenbogen_rechts	Pain_Elbow_Right
Schmerz_Finger_links	Pain_Finger_Left
Schmerz_Finger_rechts	Pain_Finger_Right
Schmerz_Oberarm_links	Pain_Upper_Arm_Left
Schmerz_Oberarm_rechts	Pain_Upper_Arm_Right
Schmerz_Daumen_links	Pain_Thumb_Left
Schmerz_Daumen_rechts	Pain_Thumb_Right
Schmerz_Unterarm_links	Pain_Forearm_Left
Schmerz_Unterarm_rechts	Pain_Forearm_Right
Schmerz_Stirn	Pain_Forehead
Schmerz_Scheitel	Pain_Crown
Schmerz_Nackenmuskeln	Pain_Neck_Muscles
Schmerz_Nackenwirbelsäule	Pain_Cervical_Spine
Schmerz_Brustwirbelsäule	Pain_Thoracic_Spine
Schmerz_Schulter+Arm_Links	Pain_Shoulder_Arm_Left
Schmerz_Schulter+Arm_Rechts	Pain_Shoulder_Arm_Right
Schmerz_Lendenwirbelsäule	Pain_Lumbar_Spine
Schmerz_Hüfte_Links	Pain_Hip_Left
Schmerz_Hüfte_Rechts	Pain_Hip_Right
Schmerz_Po+Oberschenkel_Links	Pain_Buttock_Thigh_Left
Schmerz_Po+Oberschenkel_Rechts	Pain_Buttock_Thigh_Right
Schmerz_Knie_Links	Pain_Knee_Left
Schmerz_Knie_Rechts	Pain_Knee_Right
Schmerz_Unterschenkel_Links	Pain_Lower_Leg_Left
Schmerz_Unterschenkel_Rechts	Pain_Lower_Leg_Right
Schmerz_Sprunggelenk_Links	Pain_Ankle_Left
Schmerz_Sprunggelenk_Rechts	Pain_Ankle_Right
RestlessLegs_Syndrom	Restless_Legs_Syndrome
Schwindel+Sturzneigung	Dizziness_Fall_Tendency
Vibrieren_im_Körper	Body_Vibration
Unruhiger_Schlaf	Restless_Sleep
Antriebslosigkeit	Lack_Of_Drive
Konzentrationsschwierigkeiten	Concentration_Difficulty

Table 13: English - Column name mapping (English column names are translated from the original German column names)

Declaration of Authorship

We hereby declare that this report is our own work. All direct or indirect sources used are acknowledged as references.

Contribution

Hassan Rasheed contributed to:

- Section 1. Introduction
- Section 3. LSTM-Based Approach for Gait Event Detection and Pain Classification
- Section 5. Discussion & Conclusion

Jaeyeop Chung contributed to:

- Section 2. Data
- Section 4. XGBoost-Based Approach