Pressure Plates: Can Computers be Taught to Detect Jumps?

Joshua Gao[1⋆], Advisor: Larry Wilen[1], Alexander Wei[1]

⋆ Submitted in partial fulfillment of the requirements for the Bachelor of Science degree

[1]School of Engineering and Applied Science, Yale University
15 Prospect St, New Haven, CT 06511, USA

Abstract

Over the last decade, athletic training has become increasingly scientific and data-driven in nature. Sports teams commonly incorporate techniques from biomechanics and data science to optimize performance, prevent and predict injuries, and make strategic in-game decisions. Training with sports science involves (1) collecting human performance data, (2) analyzing said data, and (3) translating findings into practical athletic training improvements. This paper seeks to improve upon current techniques for step 2: analyzing human performance data. Effective analysis requires reliably identifying key events to enable long-term comparative analysis. Existing algorithms require user adherence to stringent testing protocols, making in-situ testing processes cumbersome and natural movement data prohibitively challenging to assess. We propose a framework for parsing temporal movement data that mimics human intuition for pattern classification. We use ground reaction force (GRF) data from pressure plates, treating vertical jumps as the target event. We develop several complementary detection algorithms and borrow techniques from machine learning to develop a generalizable, computationally lightweight, linear-time complexity jump detection algorithm. A training dataset of 279 jumps across 27 participants was collected, with the optimal algorithm achieving 28 errors (90% accuracy). The framework has been successfully deployed in Vault One, a mobile application used by trainers and athletes.

Keywords: Pattern Recognition, Interpretable Machine Learning, Temporal Data Analysis

1 Introduction

Athletic training has become increasingly data-driven, with sports science workflows that involve vast quantities of data analysis. Various types of raw data are collected to assess performance, including data from video, GPS, or force plates. In recent years, significant interest has been placed on analyzing ground reaction force (GRF) data; every movement that an athlete makes must be counteracted by forces from the ground. Studies have shown that GRF data can be sufficiently high dimensional to recreate data collected from other types of sensors like video cameras.[1]

A challenge with early pressure plate data is that raw data can be noisy, uncalibrated, and at such high quantities, difficult to analyze manually. If future investment to collect comprehensive athlete performance data from GRF sensors increases, this problem will only become more prevalent. This paper works with GRF data from a 48-sensor capacitive pressure plate sampled at 50Hz. The sensors are arranged in a 6x8 grid, providing spatial resolution of force distribution across the pressure plate as people perform vertical jumps.

To any trained human analyst, it is easy to identify jumps in GRF data. We can see the force build-up during countermovement and propulsion, the rapid unloading at takeoff, the near-zero force while airborne, and the rapid reloading at landing. This paper seeks to encode this human intuition into deterministic algorithms that can be used to detect jumps in GRF data. We think of it similarly to the MNIST handwritten digit recognition challenge: we collected a labeled dataset of 279 jumps, then developed various generalizable algorithms to detect jumps in the data. We borrow concepts from machine learning (e.g. loss landscape traversal, parameter tuning) to streamline the process, and the goal is to maintain interpretable, deterministic algorithms.

Raw data is loaded into matrix $\mathbf{P} \in \mathbb{R}^{T\times 48}$, where T is the number of time samples. Information from individual sensor readings can provide granular insights into how weight is shifted across various parts of ground contact to generate propulsion and absorption. For detecting vertical jump events in this paper, we use the pooled signal. Individual sensors record pressure $P_i = F_i/A_i$ (force per unit area), where F_i is the force on sensor i and A_i is its area. Pooling these pressure measurements gives a signal related to total force: $\sum_{i=1}^{48} P_i = \sum_{i=1}^{48} F_i/A_i \propto F(t)$. Pooling the sensor signal also smooths out idiosyncratic sensor noise.

$$F(t) = \sum_{i=1}^{48} P_{t,i} \tag{1}$$

Figure 1 illustrates the data structure and the detection challenge. The figure shows three key elements: (1) **Raw sensor data**: Each thin line represents a unique sensor from the 48-sensor array, capturing individual pressure measurements over time. (2) **Pooled signal**: The thick blue line represents the pooled sensor values. Note that this signal is not calibrated to absolute force units; it is proportional to force by some factor, possibly nonlinear, depending on sensor characteristics and manufacturing variations. (3) **Ground truth markers**: The vertical dashed lines indicate manually annotated jump events—markings where jumps actually happened. Our goal is to automatically detect these events using deterministic algorithms.

We propose a framework that mimics human intuition for pattern classification. We encode human jump detection qualitative cues into interpretable signal-processing algorithms: (1) **threshold-**

[1]Related work includes the MIT intelligent carpet system, which uses pressure-sensing technology to infer 3D human pose from tactile signals. See Y. Liu et al., "Intelligent Carpet: Inferring 3D Human Pose from Tactile Signals," CVPR 2021.

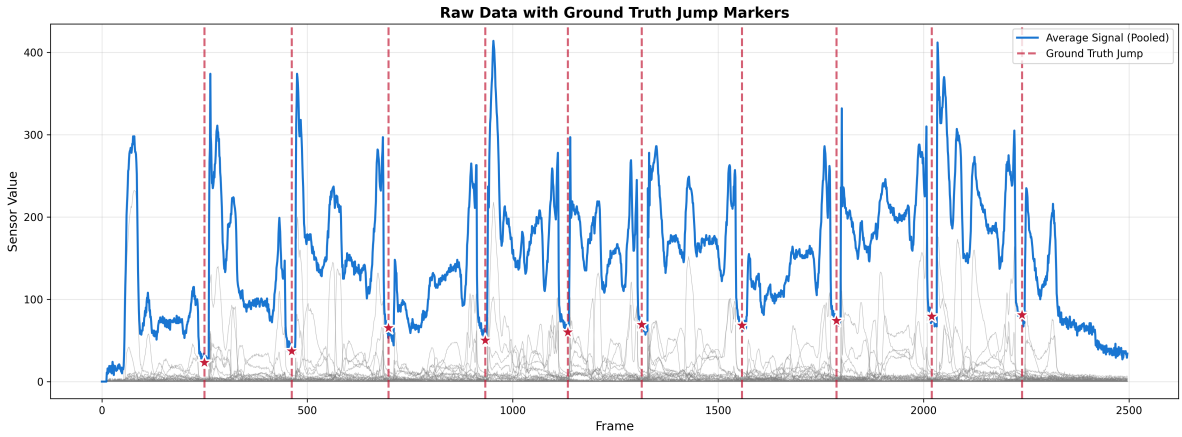

Figure 1: Raw sensor data with ground truth jump markers. Each colored line represents a unique sensor from the 48-sensor array. The thick blue line shows the pooled signal. Vertical dashed lines indicate manually annotated jump events that we aim to detect automatically.

based: identifies flight periods when force falls below a threshold; (2) **derivative-based**: detects paired takeoff and landing events in the derivative signal; (3) **correlation-based**: searches for a characteristic signal within the pooled signal resembling a jump by matching buffer regions of the signal with a pre-specified template; (4) **landing derivative**: focuses exclusively on landing detection, with jump centers calculated by offsetting backward from landing points, and (5) **ensemble**: integrates multiple algorithms through weighted voting on binary condition indicators. All algorithms are evaluated using the same loss function and parameter optimization framework, enabling direct performance comparison. All algorithms can operate in $O(T)$ time, enabling real-time practical deployment.

2 Methods

2.1 Algorithms

For each algorithm, we describe (1) a conceptual overview of the algorithm's approach, (2) a formal algorithm specification with inputs, outputs, and step-by-step pseudocode, and (3) a discussion of the algorithm's strengths and limitations.

2.1.1 Threshold Algorithm

Threshold Algorithm: Identifies flight periods when force falls below threshold θ, validated by requiring significant derivative changes at segment boundaries. The algorithm creates a binary mask for low-force periods:

$$m_{\text{threshold}}(t) = \begin{cases} 1 & \text{if } \bar{F}(t) < \theta \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

The algorithm also computes a derivative mask to identify rapid changes:

$$m_{\text{derivative}}(t) = \begin{cases} 1 & \text{if } |F'(t)| > \delta \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

where δ is the derivative threshold. Detected segments are filtered by physics constraints (minimum flight time $t_{\min} \approx 0.2$ s and maximum flight time $t_{\max} \approx 1.2$ s) and validated by requiring derivative activity at segment boundaries, ensuring detected jumps correspond to realistic flight periods with characteristic takeoff and landing events.

Input: Pooled signal $\bar{F}(t)$, threshold θ, derivative threshold δ

Output: List of detected jumps $\mathcal{J}$

1. Compute threshold mask: $m_{\text{threshold}}(t) \leftarrow (\bar{F}(t) < \theta)$
2. Compute derivative: $F'(t) \leftarrow \nabla \bar{F}(t)$
3. Compute derivative mask: $m_{\text{derivative}}(t) \leftarrow (|F'(t)| > \delta)$
4. Apply physics constraints: $m_{\text{physics}}(t) \leftarrow \text{filter}(m_{\text{threshold}}(t), t_{\min}, t_{\max})$
5. Validate segments: $m_{\text{validated}}(t) \leftarrow \text{validate\_boundaries}(m_{\text{physics}}(t), m_{\text{derivative}}(t))$
6. Extract jump segments: $\mathcal{J} \leftarrow \text{extract\_segments}(m_{\text{validated}}(t))$
7. **return** $\mathcal{J}$

The threshold algorithm provides a baseline approach for jump detection. Figure 2 demonstrates the limitations of a naive threshold approach, where a threshold of 90 detects 14 jumps after applying physics constraints (minimum flight time 0.2 s, maximum flight time 1.2 s). During the countermovement phase of a jump, the apparent force of a person on the ground as they accelerate downward decreases and can be confused with a flight period. In addition, notice the baseline levels of the signal may drift as the sensors compress over time. These two aspects push the idealized threshold in opposite directions and motivate a need for more sophisticated detection methods.

2.1.2 Derivative Algorithm

Derivative Algorithm: Detects paired takeoff and landing events in the derivative signal. The algorithm first computes the derivative $F'(t)$ of the pooled signal, then creates two binary masks by thresholding:

$$m_{\text{upper}}(t) = \begin{cases} 1 & \text{if } F'(t) > \theta_{\text{upper}} \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

$$m_{\text{lower}}(t) = \begin{cases} 1 & \text{if } F'(t) < \theta_{\text{lower}} \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

The algorithm pairs lower threshold crossings (takeoff events, where $F'(t) < \theta_{\text{lower}}$) with subsequent upper threshold crossings (landing events, where $F'(t) > \theta_{\text{upper}}$). Each valid pair $(t_{\text{lower}}, t_{\text{upper}})$ must satisfy physics-based flight time constraints:

$$t_{\min} \leq t_{\text{upper}} - t_{\text{lower}} \leq t_{\max} \tag{6}$$

To ensure that derivative threshold crossings indicate jumps and not sudden shifts on the mat, an additional validation step is added. The algorithm checks that the pooled signal falls below an in-air threshold θ_{air} during the flight phase for at least one frame in the detected jump period:

$$\exists t \in [t_{\text{lower}}, t_{\text{upper}}] : \bar{F}(t) < \theta_{\text{air}} \tag{7}$$

Input: Pooled signal $\bar{F}(t)$, thresholds θ_{upper}, θ_{lower}, θ_{air}

Output: List of detected jumps $\mathcal{J}$

1. Compute derivative: $F'(t) \leftarrow \nabla \bar{F}(t)$
2. Create upper mask: $m_{\text{upper}}(t) \leftarrow (F'(t) > \theta_{\text{upper}})$
3. Create lower mask: $m_{\text{lower}}(t) \leftarrow (F'(t) < \theta_{\text{lower}})$
4. Pair crossings: $\mathcal{P} \leftarrow \text{pair\_crossings}(m_{\text{lower}}, m_{\text{upper}}, t_{\min}, t_{\max})$
5. Validate pairs: $\mathcal{P}_{\text{valid}} \leftarrow \{p \in \mathcal{P} : \text{in\_air}(\bar{F}, p, \theta_{\text{air}})\}$
6. Convert to jumps: $\mathcal{J} \leftarrow \text{pairs\_to\_jumps}(\mathcal{P}_{\text{valid}})$
7. **return** $\mathcal{J}$

The derivative algorithm leverages the characteristic signature of jumps in the derivative signal. Figure 3 demonstrates derivative-based detection using thresholds of +20 for landing and -15 for takeoff, detecting 13 jumps with highlighted regions showing improved accuracy compared to the naive threshold approach.

2.1.3 Correlation Algorithm

Correlation Algorithm: Performs template matching on the derivative signal to detect jump signatures. The algorithm creates a template buffer T of size B that encodes the characteristic jump pattern across three phases:

$$T[i] = \begin{cases} -1 & \text{if } 0 \leq i < w_{\text{neg}} \quad \text{(takeoff phase)} \\ 0 & \text{if } w_{\text{neg}} \leq i < w_{\text{neg}} + w_{\text{zero}} \quad \text{(in air phase)} \\ 1 & \text{if } w_{\text{neg}} + w_{\text{zero}} \leq i < B \quad \text{(landing phase)} \end{cases} \tag{8}$$

For each position t in the derivative signal $F'(t)$, the algorithm computes the cross-correlation:

$$C(t) = T \cdot F'_{t:t+B} \tag{9}$$

where $F'_{t:t+B}$ is the window of the derivative signal of length B starting at position t. Jump centers are identified at local maxima where $C(t) > \theta_{\text{corr}}$. The algorithm then estimates takeoff and landing events around each center and applies flight time constraints to validate detections.

Input: Pooled signal $\bar{F}(t)$, template parameters B, w_{neg}, w_{zero}, w_{pos}, correlation threshold θ_{corr}

Output: List of detected jumps $\mathcal{J}$

1. Compute derivative: $F'(t) \leftarrow \nabla \bar{F}(t)$
2. Create template: $T \leftarrow \text{create\_template}(w_{\text{neg}}, w_{\text{zero}}, w_{\text{pos}}, B)$
3. Compute correlations: $C(t) \leftarrow T \cdot F'_{t:t+B}$ for all t
4. Find peaks: $\mathcal{C} \leftarrow \{t : C(t) > \theta_{\text{corr}} \text{ and } C(t) \text{ is local maximum}\}$
5. Estimate boundaries: For each center $c \in \mathcal{C}$, find takeoff and landing around c
6. Apply constraints: $\mathcal{J} \leftarrow \{j \in \text{estimated\_jumps} : t_{\min} \leq \text{duration}(j) \leq t_{\max}\}$
7. **return** $\mathcal{J}$

The correlation algorithm leverages template matching to identify the characteristic jump signature in the derivative signal. By encoding the expected pattern of negative (takeoff phase), zero (flight-phase), and positive (landing phase) derivatives, it can detect jumps even when absolute thresholds on derivatives vary. Note that careful selection of the template parameters is crucial for performance. To ensure that the algorithm is functional for jumps of varying height (and thus flight time), we set either w_{neg} or w_{pos} to be large relative to the other. The smaller boundary detects the precise moment of takeoff or landing, while the larger boundary captures the other key event under varying flight times.

2.1.4 Landing Derivative Algorithm

Landing Derivative Algorithm: Focuses exclusively on landing detection by identifying positive derivative threshold crossings, then estimates jump centers by offsetting backward from landing points. The algorithm first creates a landing mask:

$$m_{\text{landing}}(t) = \begin{cases} 1 & \text{if } F'(t) > \theta_{\text{landing}} \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

For each detected landing at time t_{landing}, the algorithm estimates a jump center by offsetting backward by a fixed amount:

$$c = t_{\text{landing}} - \text{offset} \tag{11}$$

where the offset is a parameter that provides a rough guess of the jump center location. Each detected jump must satisfy flight time constraints, and optionally must contain an in-air period where $\bar{F}(t) < \theta_{\text{air}}$.

Input: Pooled signal $\bar{F}(t)$, landing threshold θ_{landing}, center offset, in-air threshold θ_{air} (optional)

Output: List of detected jumps $\mathcal{J}$

1. Compute derivative: $F'(t) \leftarrow \nabla \bar{F}(t)$

2. Create landing mask: $m_{\text{landing}}(t) \leftarrow (F'(t) > \theta_{\text{landing}})$
3. Detect landings: $\mathcal{L} \leftarrow \text{find\_landings}(m_{\text{landing}}(t))$
4. For each landing $l \in \mathcal{L}$:
   (a) Estimate center: $c \leftarrow l - \text{offset}$ (offset backward from landing to guess center)
   (b) Estimate jump boundaries around center c to get start s and end e
   (c) Check flight time: if $t_{\min} \leq (e - s)/f_s \leq t_{\max}$:
      i. If θ_{air} specified: check if $\exists t \in [s, e] : \bar{F}(t) < \theta_{\text{air}}$
      ii. If valid: add jump (s, e, c) to $\mathcal{J}$
5. **return** $\mathcal{J}$

The landing derivative algorithm simplifies detection by focusing on the most distinctive event—the landing impact—which produces a clear positive spike in the derivative signal. A problem when using both negative and positive derivative thresholds was that some unloading phases (whether influenced by gradual foot rolloff from the ground or sensor decompression hysteresis) had unclear negative derivative crossings.

2.1.5 Ensemble Algorithm

Ensemble Algorithm: Combines multiple detection algorithms through weighted voting on binary condition indicators. The algorithm runs all individual algorithms (threshold, derivative, correlation) and extracts binary condition signals from each. It then computes a weighted score at each time step:

$$S(t) = \sum_{i=1}^{11} w_i \cdot c_i(t) \tag{12}$$

where $c_i(t) \in \{0, 1\}$ are binary condition signals (e.g., threshold mask, derivative upper mask, correlation above threshold) and w_i are their corresponding weights. A jump mask is created by thresholding the score:

$$m_{\text{jump}}(t) = \begin{cases} 1 & \text{if } S(t) \geq \theta_{\text{score}} \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

The final jump mask is filtered by physics constraints (flight time bounds) to produce the final detections.

Input: Pooled signal $\bar{F}(t)$, algorithm parameters for threshold, derivative, correlation, weights $\{w_i\}_{i=1}^{11}$, score threshold θ_{score}

Output: List of detected jumps $\mathcal{J}$

1. Run threshold algorithm: $s_{\text{thresh}} \leftarrow \text{threshold\_pipeline}(\bar{F}(t))[0]$
2. Run derivative algorithm: $s_{\text{deriv}} \leftarrow \text{derivative\_pipeline}(\bar{F}(t))[0]$
3. Run correlation algorithm: $s_{\text{corr}} \leftarrow \text{correlation\_pipeline}(\bar{F}(t))[0]$

4. Extract condition signals: $\{c_i(t)\}_{i=1}^{11} \leftarrow \text{extract\_conditions}(s_{\text{thresh}}, s_{\text{deriv}}, s_{\text{corr}})$
5. Compute weighted score: $S(t) \leftarrow \sum_{i=1}^{11} w_i \cdot c_i(t)$
6. Create jump mask: $m_{\text{jump}}(t) \leftarrow (S(t) \geq \theta_{\text{score}})$
7. Apply physics constraints: $m_{\text{physics}}(t) \leftarrow \text{filter}(m_{\text{jump}}(t), t_{\min}, t_{\max})$
8. Extract jumps: $\mathcal{J} \leftarrow \text{extract\_segments}(m_{\text{physics}}(t))$
9. **return** $\mathcal{J}$

The ensemble algorithm leverages the complementary strengths of multiple detection approaches. By combining condition signals through weighted voting, it can achieve higher accuracy than individual algorithms, though at the cost of increased complexity and a larger parameter space (20+ parameters including all weights and thresholds).

2.1.6 Algorithm Visualization

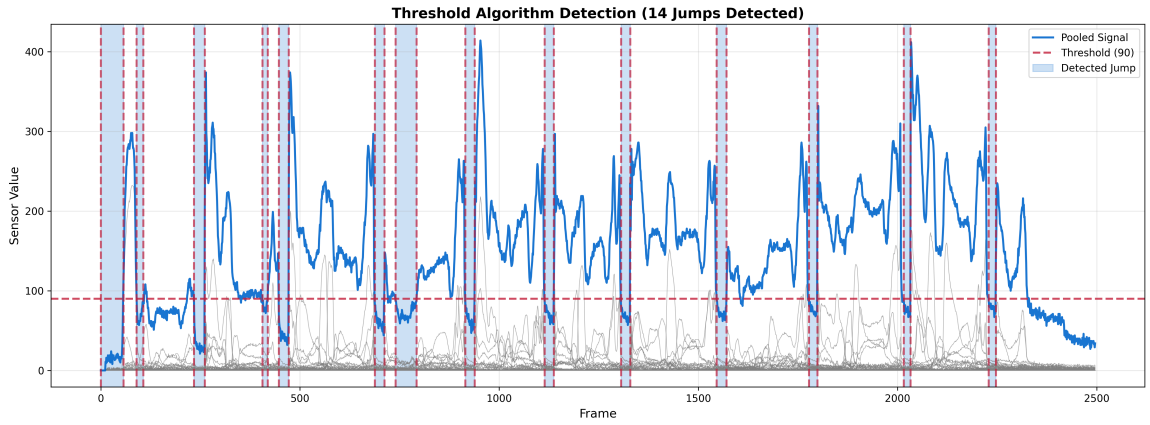

Figure 2: Threshold algorithm detection (threshold = 90) with multisensor background, pooled signal overlay, and highlighted jump boundaries.

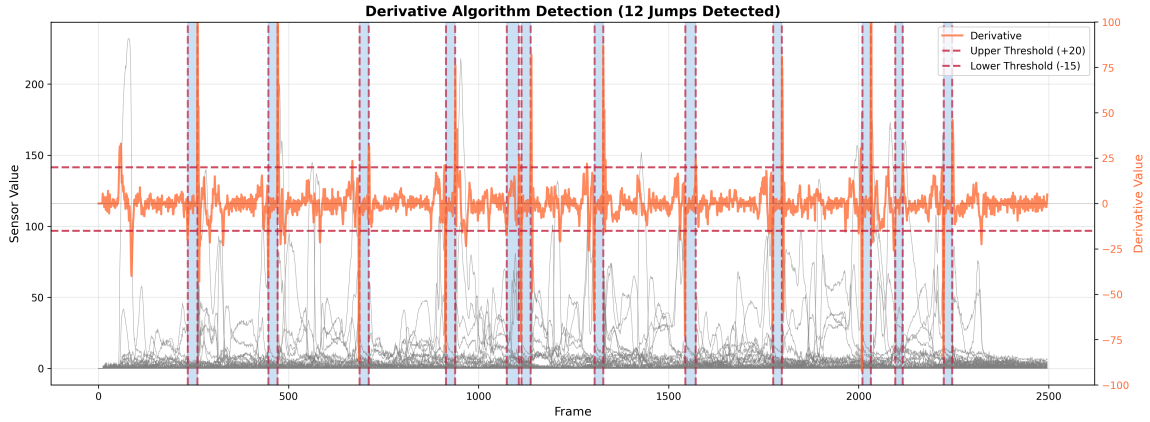

Figure 3: Derivative-based detection (thresholds: +20 / -15) with multisensor background, derivative overlay, and highlighted jump boundaries. This approach pairs takeoff and landing events, validated by in-air thresholds, resulting in more accurate detection than naive threshold methods.

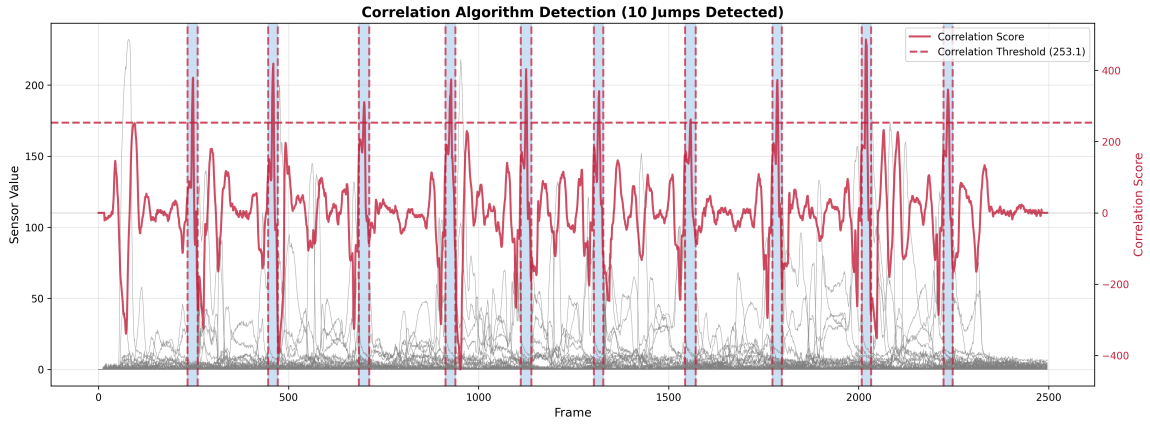

Figure 4: Correlation-based detection with multisensor background, correlation scores overlay, and highlighted jump boundaries. Template matching identifies jump signatures in the derivative signal through cross-correlation.

2.2 Optimization

2.2.1 Measuring Success

To systematically evaluate and compare algorithm performance across a large dataset, we require a quantitative measure of detection accuracy. We define a loss function:

$$L(\theta) = \text{FP}(\theta) + \text{FN}(\theta) \tag{14}$$

where θ represents the parameters for any algorithm used (e.g., threshold values, template parameters, ensemble weights), FP represents false positive jumps (detected jump regions that do not bound a corresponding ground truth marker), and FN represents false negative jumps (ground truth markers not captured by a detected jump region).

Manual Tagging: To efficiently annotate jump events, we developed a Python script using matplotlib that displays the force signal data and allows interactive tagging of takeoff frame indices.

2.2.2 Optimization Methodology

For algorithms with few parameters (typically 2-3), we can employ exhaustive grid search to find optimized parameter combinations. Grid search involves systematically evaluating the loss function at uniformly spaced points across the parameter space. For example, for the threshold algorithm, we can search over threshold $\theta \in [30, 500]$ and derivative threshold $\delta \in [0, 200]$ with 60 uniformly spaced values per dimension, creating a 60×60 grid requiring 3600 evaluations. Similarly, for the derivative algorithm, we can search over upper threshold $\theta_{\text{upper}} \in [0, 100]$ and lower threshold $\theta_{\text{lower}} \in [-100, 0]$ with 60 values each, creating a 60×60 grid requiring 3600 evaluations. This two-dimensional grid search creates loss landscapes that can be visualized as surfaces, revealing regions of optimal performance and enabling direct visual comparison between algorithms.

However, as the number of parameters increases, grid search becomes computationally prohibitive. To optimize all parameters of the derivative algorithm with 5 parameters, a grid search with just 50 values per dimension would require $50^5 = 312.5$ million evaluations. For the ensemble algorithm with 20+ parameters, exhaustive grid search is completely infeasible. This curse of dimensionality makes it essential to employ more sophisticated optimization techniques for high-dimensional parameter spaces.

To address this challenge, we employ Langevin Monte Carlo sampling—a stochastic optimization technique that allows us to take random samples in the grid search space without exhaustive enumeration. Langevin sampling combines gradient information with random noise to efficiently navigate complex loss landscapes, enabling comprehensive multi-parameter optimization that would be computationally prohibitive with exhaustive grid search.

Langevin Sampling Methodology:

Our implementation uses a Metropolis-Hastings (MH) sampling strategy with the following components:

- **Parameter initialization:** Parameters are initialized randomly within predefined bounds for each algorithm. For example, for the derivative algorithm, bounds are: $\theta_{\text{upper}} \in [0, 100]$, $\theta_{\text{lower}} \in [-100, 0]$, $\theta_{\text{air}} \in [100, 400]$, $t_{\min} \in [0.1, 0.5]$ seconds, and $t_{\max} \in [0.5, 1.5]$ seconds.

- **Parameter perturbation:** At each iteration, one parameter is selected sequentially (cycling through all parameters in order) and perturbed with Gaussian noise: $\theta_{\text{new}} = \theta_{\text{old}} + \mathcal{N}(0, \sigma^2)$, where the step size σ defaults to 5% of the parameter's range. This sequential approach ensures systematic exploration of each parameter dimension. Perturbed parameters are clipped to remain within bounds, and constraints (e.g., $t_{\min} < t_{\max}$) are enforced.

- **Metropolis-Hastings acceptance criterion:** Proposed parameters are accepted with probability:

$$P(\text{accept}) = \begin{cases} 1 & \text{if } L(\theta_{\text{proposed}}) \leq L(\theta_{\text{current}}) \\ \exp\left(-\frac{L(\theta_{\text{proposed}}) - L(\theta_{\text{current}})}{T}\right) & \text{otherwise} \end{cases} \tag{15}$$

  where $L(\theta)$ is the loss function and T is the current temperature. This ensures that improvements are always accepted, while worse configurations are accepted probabilistically to enable exploration.

- **Temperature schedule:** Initial temperature $T_0 = 10.0$ with exponential decay $T_{i+1} = 0.9995 \cdot T_i$ per iteration. This cooling schedule enables initial exploration of the parameter space, gradually transitioning to more greedy acceptance of improvements.

- **Best state tracking:** Throughout the sampling process, we maintain a separate record of the best (lowest loss) parameter configuration encountered.

- **Comprehensive optimization:** We perform this optimization for 10,000 steps on each algorithm, repeating the trial several times to ensure we have found a good minimum.

3 Results and Discussion

3.1 Algorithm Implementation on Dataset

Each algorithm was applied to the complete dataset of 279 ground truth markers across 27 participants. We identified optimal parameters through Langevin sampling, which enables comprehensive multi-parameter optimization across the full parameter space.

Table 1: Algorithm Performance After Optimization

Algorithm	**Loss**	**Parameters Optimized**
Threshold	107	θ, δ, $t_{\min}$, $t_{\max}$
Derivative	**28**	θ_{upper}, θ_{lower}, θ_{air}, $t_{\min}$, $t_{\max}$
Correlation	103	B, w_{neg}, w_{zero}, w_{pos}, θ_{corr}
Landing Derivative	34	θ_{landing}, offset, window, θ_{air}, $t_{\min}$, $t_{\max}$
Ensemble	30	Weights (11), score threshold, $t_{\min}$, $t_{\max}$

Table 1 summarizes the performance of all algorithms after Langevin sampling optimization. The derivative algorithm achieved the lowest loss (28 errors, 90% accuracy), providing an excellent

balance between performance and interpretability. The ensemble algorithm should theoretically be capable of achieving the best performance, or at least performing as well as the derivative algorithm (which it can attain by identifying optimal derivative thresholds and zeroing out the weights for all other algorithms). However, it is possible that given the high dimensionality of the parameter space, the algorithm may not have found the global minimum.

For the derivative algorithm, Langevin sampling found optimal parameters: $\theta_{\text{upper}} = 17.71$, $\theta_{\text{lower}} = -15.19$, $\theta_{\text{air}} = 183.04$, with minimum flight time $t_{\min} = 0.19$ seconds and maximum flight time $t_{\max} = 0.60$ seconds, achieving a loss of 28 errors (90% accuracy). In practice, to ensure that this algorithm is extendable to future datasets, the physics parameter range should be broadened to 0.2 seconds to 1.2 seconds, filtering out unrealistically short jumps and jumps with flight times beyond humanly possible. The loss landscape visualization, where only the upper and lower derivative threshold parameters were adjusted, shows a broad optimal region, indicating robustness to parameter variations. This suggests that the derivative algorithm may be suitable for deployment across diverse conditions.

For the correlation algorithm, Langevin sampling found optimal parameters: buffer size $B = 33.4$, negative frames $w_{\text{neg}} = 18.2$, zero frames $w_{\text{zero}} = 7.2$, positive frames $w_{\text{pos}} = 5.2$, and correlation threshold $\theta_{\text{corr}} = 224.4$, achieving a loss of 103 errors (63% accuracy). While in Figures 2, 3, and 4 the correlation algorithm appeared to perform most optimally, on the larger dataset it was challenging for the algorithm to discover a more optimal fixed set of parameters that were generalizable across all jumps. Due to the large window size of the template region needed to accommodate different jump heights, the algorithm may be biased toward certain jump patterns.

Detailed performance metrics for each participant are provided in Table 2 (see Appendix).

Figure 5 shows the side view (XZ plane) of the loss landscapes, revealing the derivative algorithm's superior performance with lower loss values across the parameter space. The threshold algorithm consistently achieves higher loss than the derivative algorithm. The loss landscapes show clear concave structures with well-defined optimal regions, confirming that both algorithms are sensitive to their parameter choices.

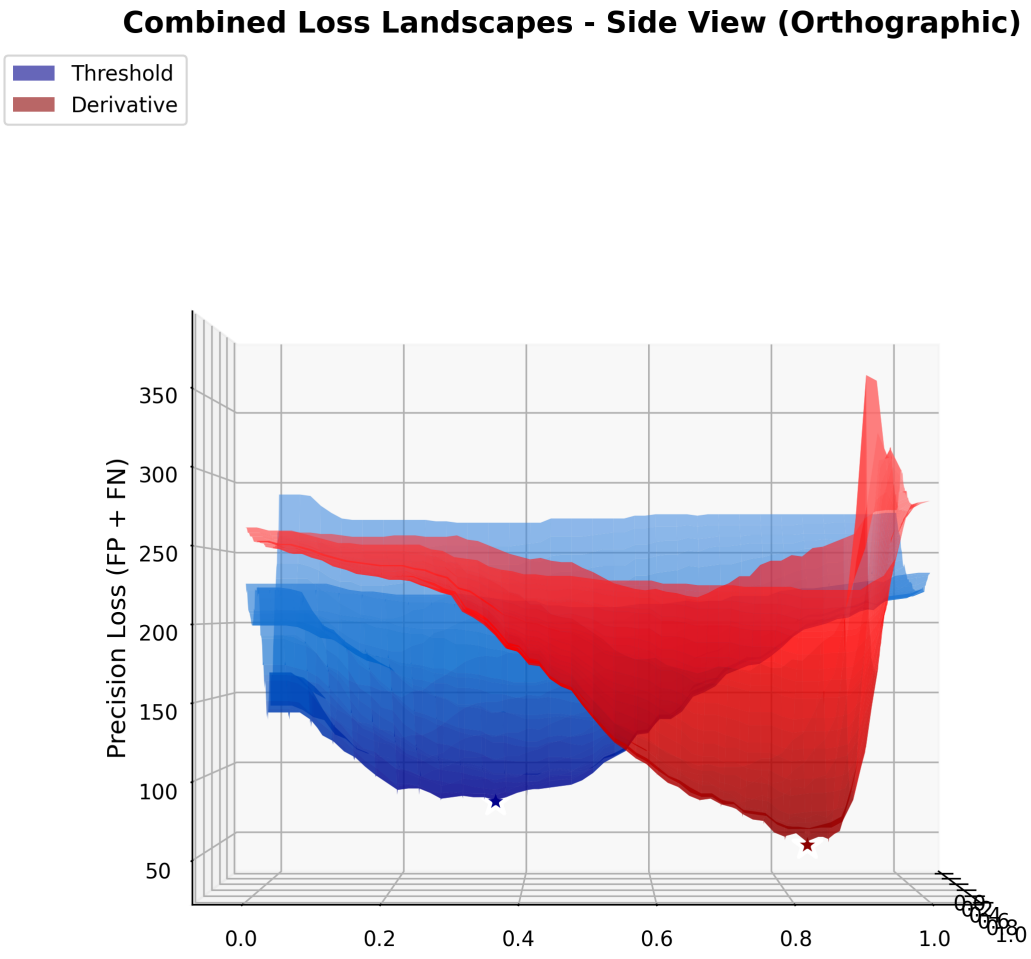

Figure 5: Two-dimensional grid search loss landscapes comparing threshold and derivative algorithms. Threshold: $\theta \in [30, 500]$, $\delta \in [0, 200]$ (60 values each); Derivative: $\theta_{\text{upper}} \in [0, 100]$, $\theta_{\text{lower}} \in [-100, 0]$ (60 values each). The side view (XZ plane) reveals the derivative algorithm's superior performance with lower loss values across the parameter space. Both landscapes show clear variation across parameter combinations, demonstrating the sensitivity of each algorithm to its respective parameters.

3.2 Downstream Analysis

Once jumps are reliably detected, we can extract precise boundaries and compute performance metrics. We refine jump boundaries using half-peak thresholding: for jump center c, find peaks p_{before} and p_{after} within ± 70 frames, then locate boundaries where signal exceeds half-peak values.

To improve the granularity of our 50Hz sampling, rather than flagging the discrete sample where the threshold is crossed, we linearly interpolate through neighboring points to calculate a more precise time of threshold crossing. These precisely detected takeoff and landing times enable the calculation of vertical jump height, flight time, and vertical takeoff velocity using kinematics. Given the mass of a jumper, the impulse can also be calculated.

For a detected jump with takeoff time t_{takeoff} and landing time t_{landing}, we compute the following metrics:

$$t_{\text{flight}} = t_{\text{landing}} - t_{\text{takeoff}} \tag{16}$$

$$h = \frac{1}{8} g t_{\text{flight}}^2 \tag{17}$$

$$v_{\text{takeoff}} = \frac{1}{2} g t_{\text{flight}} \tag{18}$$

$$J = m \cdot v_{\text{takeoff}} = \frac{1}{2} m g t_{\text{flight}} \tag{19}$$

where $g = 9.81$ m/s^2 is gravitational acceleration, m is the jumper's mass, h is jump height, v_{takeoff} is vertical takeoff velocity, and J is the impulse.

Beyond individual jump analysis, automated detection enables longitudinal tracking and trend analysis. By storing detected jump data over time, we can monitor athlete progress, identify performance trends, and track evolution of key metrics. This capability transforms isolated measurements into actionable insights: trainers can observe how jump height improves over a training period, how contact time decreases with improved reactive strength, or how force production patterns evolve with technique refinement.

To validate the accuracy of our mat-based flight time measurements, we compared them against video-based ground truth measurements. We analyzed 31 jumps with synchronized video recordings at 240 fps and mat sensor data. Video measurements were extracted using two methods: heel-to-heel flight time (from last heel contact at takeoff to first heel contact at landing) and toe-to-toe flight time (from last toe contact at takeoff to first toe contact at landing). Linear regression analysis revealed correlations between mat measurements and video measurements. The mat measurments straddle in between the toe-to-toe and heel-to-heel measurements, suggesting that the derivative thresholds trigger at a point between heel and toe contact on the ground.

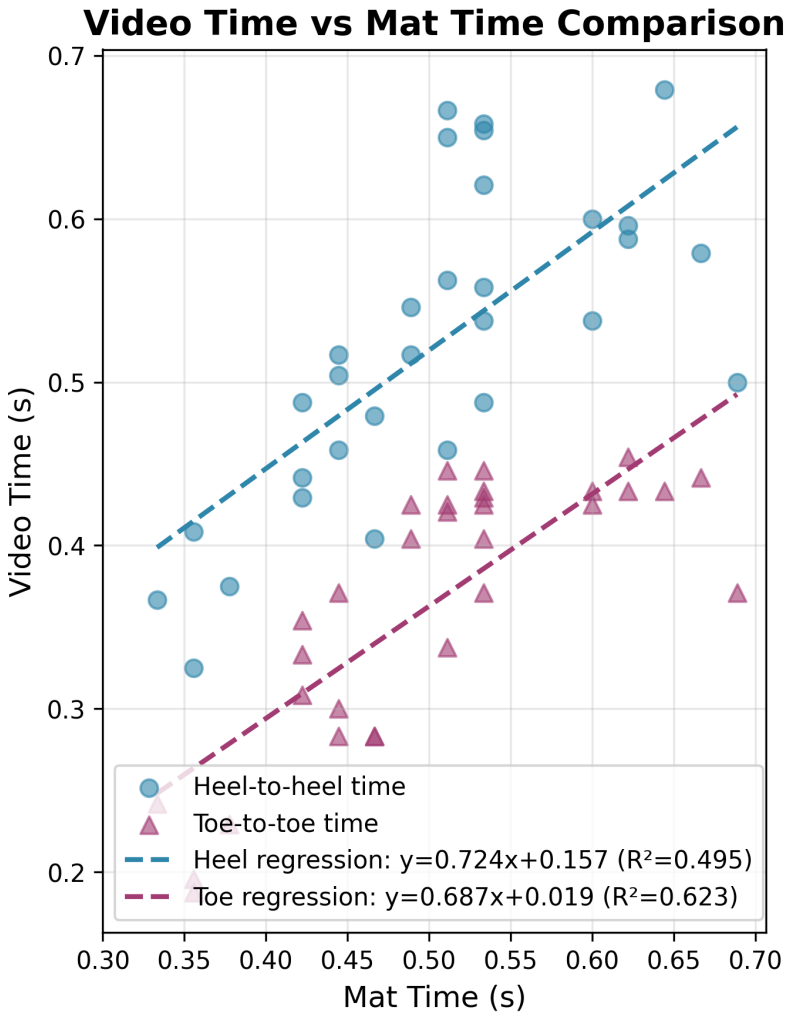

Figure 6: Comparison of video-based flight time measurements (heel-to-heel and toe-to-toe) versus mat sensor measurements. Linear regression lines are shown for each comparison, with R^2 values indicating the strength of correlation. The toe-to-toe measurements show stronger correlation ($R^2 = 0.623$) with mat measurements than heel-to-heel measurements ($R^2 = 0.495$).

3.3 Future Work

Future directions include: (1) extending to other movement tasks (sprints, change-of-direction) with task-specific parameter tuning; (2) adaptive parameter tuning based on real-time signal characteristics; and (3) multi-sensor fusion leveraging spatial force patterns.

One interesting observation from our optimized parameters is that the lower derivative threshold ($\theta_{\text{lower}} = -15.19$) is smaller in magnitude than the upper threshold ($\theta_{\text{upper}} = 17.71$). This asymmetry is partially due to jump biomechanics: takeoff events must be more gradual than abrupt landings. However, a more likely contributing factor is the hysteresis and relaxation times of the dielectric layer used in the capacitive pressure plate sensor. This dielectric layer, composed of foam, can affect the response characteristics differently during compression (landing) versus decompression (takeoff).

Future work should integrate with research on selecting optimal dielectric materials and characterizing various foam compositions. By systematically evaluating different materials' hysteresis properties, relaxation times, and response characteristics, materials that minimize signal distortion and improve detection accuracy can be identified. This integration would enable optimal sensor design specifications that complement the algorithmic approach in this paper, ultimately achieving the best possible performance at the intersection of hardware and software optimization.

3.4 Real-World Deployment: Vault One

The framework has been deployed in Vault One, a mobile application for trainers and athletes. The derivative algorithm was selected as the final model based on its superior performance (28 errors) and interpretability. The optimized parameters are: $\theta_{\text{upper}} = 17.71$, $\theta_{\text{lower}} = -15.19$, $\theta_{\text{air}} = 183.04$, with $t_{\min} = 0.19$ s and $t_{\max} = 0.60$ s.

Implementation: The algorithm was implemented in JavaScript for real-time processing. The $O(T)$ complexity enables real-time mobile deployment, allowing trainers and athletes to receive immediate feedback during training sessions (Figure 7).

Figure 7: Vault One capacitive pressure plate with the derivative algorithm.

4 Conclusion

This paper introduces a lightweight, interpretable framework for automated jump detection in ground reaction force data. By mimicking human intuition through simple signal-processing rules and physics-informed constraints, we achieve strong performance on 279 jumps across 27 participants, while maintaining $O(T)$ complexity suitable for real-time mobile deployment.

The framework's contributions are threefold: (1) encoding human-intuitive pattern detection as interpretable algorithms, providing an alternative to black-box machine learning; (2) comprehensive parameter optimization via Langevin sampling with grid search visualization for algorithm comparison; and (3) validation on real-world data.

While the framework achieves strong performance, there is room for improvement. Human analysts can often distinguish jump events that algorithms miss, suggesting there may be additional contextual information or levels of judgment not yet captured. Future work could explore more sophisticated approaches such as machine learning or sliding window analysis, though these may sacrifice the interpretability and generalizability that make the current approach valuable. The dataset and code are available at `https://github.com/joshuagao7/thesisresearch`.

Acknowledgments

I thank Dr. Larry Wilen, my thesis advisor and mentor, for his guidance and support over the past four years. I co-invented the pressure sensing mat together with him, and he taught me many things during this project, including how to sail.

I thank Alexander Wei for writing the hardware research paper that I reference, and for building the many pressure sensing mats used in this study. I thank Professor Rebecca Kramer-Bottiglio for her support on this thesis. Her lab's papers on capacitive measurements were essential reading when I first started working with these sensors. I thank Professor Kyle Jensen for encouraging me to learn how to code, which led to this full writeup. I thank Eric Wang for always being willing to chat with me about code and the latest thoughts.

I thank Matthew Riley and Paul Douglass for data collection, testing, and building together. I thank the friends and colleagues I had the chance to work with on this project over my summers: Eunice Han, Justin Pan, Diana Cao, Matt Sahagun, Skylar Wang. I have learned a lot from you all. I thank Dean Vince for teaching the class where this project was initially conceived, and Abby Quinn for voicing the need for pressure data collection in athletics. I thank Jorge Torres for his mentorship throughout. I thank Yale Athletics, Yale Engineering, and Tsai City for their support. Finally, I thank all participants who provided data for this study.

5 Appendix

Table 2: Jump Detection Performance by Participant

Participant	Expected Jumps	Detected Jumps	Loss (FP+FN)
Dan Braun	10	10	0
Darwin	10	10	0
Joey	10	12	2
Joshua Gao	10	13	2
Joshua Gao (natural)	10	10	0
Joshua Kerner	10	11	1
Matthew Riley	10	10	0
PWG Subject 1	10	10	0
PWG Subject 2	10	13	3
PWG Subject 3	10	10	0
PWG Subject 5 (190 lbs)	10	13	3
PWG Subject 6 (160 lbs)	10	7	4
PWG Subject 7 (220 lbs)	10	10	0
PWG Subject 8 (175 lbs)	10	11	1
PWG Subject 9 (170 lbs)	10	13	5
Matthew (2scaleup)	10	10	0
Dante (205 lbs, 5'7")	10	12	2
Tommy (200 lbs, 6ft)	10	10	0
Izayah (149 lbs, 5'7")	10	14	4
Lofty (70 kg, 5'7")	10	12	2
Matthew (scaleup)	10	11	1
Mouse Subject 2 (145 lbs)	10	11	1
Mouse Subject 3 (145 lbs)	10	6	4
Mouse Subject 4 (5'5", 135 lbs)	10	5	5
Mouse Subject 5 (5'6", 160 lbs)	9	6	3
Mouse Subject 6	10	7	3
Noah (130 lbs)	12	12	0
Total	**279**	**279**	**28**