Wearable Wrist Ultrasound for Real-Time Cursor and Keystroke Control

Capstone Proposal

Prepared by:

Tharusha Herath, 101221183

Leo Desbiens, 101192977

Ranveer Dhaliwal, 101227005

Supervised by:

Dr. Yuu Ono

Dr. Sreeraman Rajan

October 17, 2025

Table of Contents

1. Introduction & Problem Statement ~ *LD*

1.1 Background

As *virtual reality (VR)* and *extended reality (XR)* systems gain maturity, developing control interfaces that are both precise and intuitive is critical. Hand-gesture inputs have emerged as the leading modality due to the multiple degrees of freedom the human hand offers and its capacity for high-precision control [1]. Allowing users to execute natural, real-world gestures opens up the possibility to develop applications for multiple domains in which fine dexterity is required such as surgeries, remote operation of vehicles, etc. without any platform-specific retraining overhead [1]. To be viable in practice, these hand control interfaces must ensure they deliver low end-to-end latency, are highly accurate and demonstrate stable performance across time, users and varied environmental conditions.

1.2 Problem Definition

Currently, most control interface frameworks that classify hand gestures are camera-based, which imposes inherent line-of-sight limitations *(fig. 1)*. This approach also introduces numerous other failure points: poor lighting, self/object occlusion and subtle or rapid motions can all drastically reduce accuracy of these *computer-vision* based systems. [2] We propose a wearable, light-weight ultrasound-based sensing approach for wrist gestures and motion capture that removes the line-of-sight requirement. We aim to match state-of-the-art computer-vision baselines in terms of reliability and accuracy while providing properties not achievable with camera-based methods: occlusion resistance and field-of-view invariance.

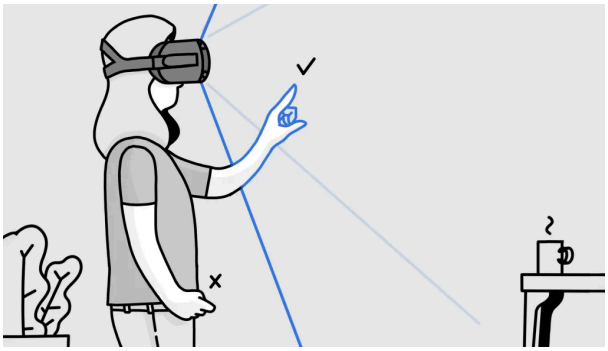

Figure 1. Current computer-vision based interfaces force the user to keep their hands within the field of view of a camera. Source: Meta

1.3 Motivation

We chose this project for its professional-development value and its potential for relevancy across multiple fields. Although *wearable ultrasound sensors (WUS)* are often associated with health tracking, advances in deep learning are unlocking increasingly complex applications beyond medicine, such as limb-pose estimation and silent-speech recognition [3]. The use cases will keep multiplying since ultrasound offers a reliable and economically viable way to analyze muscle tissues, which are information-rich. Most current research papers exploring the pose prediction abilities of A-mode ultrasound only highlight the theoretical feasibility of such projects, without actually implementing real-time solutions that leverage their findings. We want to create one of the first robust and usable documented end-to-end motion prediction systems both for the novelty and the usefulness of highlighting pain points that can occur when deploying machine-learning models analyzing ultra-sound signals in a real environment. More broadly, with the rise of new powerful transformer-based models, learning how to deploy machine-learning systems efficiently in resource-constrained environments at the edge will be a skill that will translate to multiple other industry use-cases. The modality of the ultrasound data is uncommon and differs from the more often seen image, time-series and textual modalities – learning how to adapt certain architectures to extract the most information out of this type of data will be educational and require us to gain domain knowledge in both deep learning and ultrasound signals.

Finally, since our work will span the complete path from ultrasonic echo reception all the way to a kinematics translation engine in a real-time setting, the breadth of the project will enable us to implement multiple engineering concepts learned throughout our academic careers: low-level memory management, interprocess communication, parallelism, communication protocols and solid object-oriented design practices.

2. Objectives ~ *RD*

This project aims to design, implement, and validate a real-time dual-pipeline ultrasound acquisition system that maps wrist ultrasound signals to interactive pointer-and-click controls, while simultaneously logging data for offline analysis and retraining. The system operates through two coordinated pipelines (*Fig 2)*. This dual-path architecture enables immediate responsiveness for user interaction while maintaining a comprehensive record of data and metadata for continuous model refinement.

1. Real-time control path: acquisition → preprocessing → local inference → UI mapping. The real-time control pipeline focuses on immediate interaction and responsiveness. It processes ultrasound data through acquisition, preprocessing, and local inference before

mapping the results to the user interface—ensuring minimal latency and high responsiveness for control actions.

2. Asynchronous logging path: acquisition → preprocessing → Kafka → archive.
   The asynchronous logging pipeline operates in parallel, capturing and transmitting preprocessed data for storage and later use. This path handles dataset collection, replay, and model improvement by archiving ultrasound data and metadata in cloud storage for subsequent retraining and performance refinement.

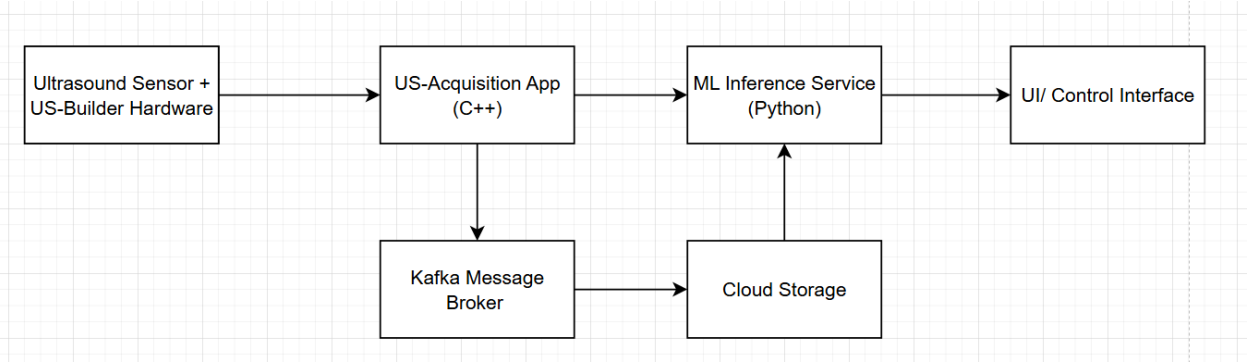

Figure 2. Dual-Pipeline Ultrasound Acquisition System Architecture

At its core, the system leverages machine learning to track wrist motion and classify hand gestures, allowing users to control a computer without a keyboard or mouse. Wrist movements are translated into cursor control, while finger gestures emulate keystrokes in real time. Alongside inference, a Kafka-based logging and analytics pipeline supports monitoring, evaluation, and ongoing retraining of the underlying models.

Unlike purely offline biomedical systems, this design prioritizes low-latency performance and efficiency to enable real-time interaction, while still preserving all collected data for long-term development. This separation of concerns follows best practices from industrial *Internet of Things (IoT)* and edge-AI systems: one path guarantees responsiveness and immediate feedback, and the other ensures observability, scalability, and maintainability [4].

The project's objectives are shaped by two main considerations. First, technical constraints and benchmarks from comparable ultrasound acquisition platforms, streaming systems, and GPU inference services inform design and performance targets. Second, application-level requirements for *human–computer interaction (HCI)*, where responsiveness, reliability, and usability determine user acceptance, drive the user-facing requirements. To satisfy these needs, objectives are organized into functional requirements, non-functional requirements, and measurable success metrics so the system can be quantitatively evaluated and effectively demonstrated.

2.1 Functional Objectives

At its core, this project must demonstrate the ability to capture ultrasound wrist signals and transform them into actionable computer interface controls. To achieve this, we define the following functional objectives, informed by prior work in real-time biomedical acquisition systems, IoT streaming pipelines, and VR/XR interaction frameworks.

- **Continuous streaming real-time ultrasound acquisition:**
  The project must reliably read A-scan frames from the US-Builder and expose a timestamped frame stream so downstream processing has continuous input. The US-Builder documentation confirms native USB/serial interfaces and example code for streaming frames, which supports low-latency capture and developer integration [5].

- **Preprocessing pipeline:**
  Raw A-scan signals require deterministic preprocessing to produce consistent model inputs. Documented approaches in ultrasound sensing and gesture work show a denoise→envelope→normalize pipeline improves model robustness for wrist/gesture tasks [6].

- **Low-latency inference service :**
  For interactive pointer control the inference path must be as short and deterministic as possible. Prior work in ultrasound-based motion/gesture estimation demonstrates feasible accuracy when inference is performed on optimized runtimes [7].

- **Calibration & UI mapping:**
  A small calibration routine that maps predicted wrist angles to screen coordinates and thresholds for click detection is necessary for usability and repeatable demos. Direct-manipulation HCI literature emphasizes that mapping & feedback enable acceptable control experience[8].

- **Asynchronous logging & dataset archive:**
  Separating an asynchronous logging pipeline allows full-frame, metadata and prediction capture for later retraining without introducing latency to the control path. Kafka is industry-standard for high-throughput, persistent streaming and is appropriate for this use case [9].

2.2 Non-Functional Objectives

Beyond core functionality, the system must meet a set of performance and quality constraints to ensure it is practical, reliable, and extensible. These non-functional objectives are informed by benchmarks from real-time biomedical acquisition systems, VR/AR interaction pipelines, and distributed streaming frameworks.

- **End-to-end latency:**
  Human–computer interaction research shows that users perceive control interfaces as "instantaneous" when latency remains below roughly 100 ms. To ensure smooth and natural pointer control, the system targets an average response time under 70 ms and keeps the 95th percentile (p95) below 90 ms. This threshold provides a design margin for network and inference variability while maintaining perceived real-time responsiveness [10][11].

- **Inference latency:**
  Modern optimized runtimes and model tuning routinely achieve sub-30 ms inference for compact models on GPU or well-tuned CPU builds. By keeping model inference within this range, the majority of the latency budget can be reserved for data acquisition, streaming, and visualization, ensuring the end-to-end response remains under the 90 ms perceptual threshold for instantaneous interaction [12][13].

- **Accuracy (wrist-angle/gesture classification) and false-click rate:**
  Published ultrasound hand/gesture recognition studies commonly report accuracy in the mid-80s to 90s% for supervised models on A-mode or short ultrasound sequences, which indicates that a ≥85% target is achievable for a well-curated dataset and reasonable model. The false-click constraint controls spurious commands in the UI [14][15].

- **Cross-Subject Robustness:**
  Cross-subject and small probe placement changes often cause accuracy drops in wearable ultrasound systems. Setting an allowable degradation (≤10%) helps guide data collection and augmentation for robustness. Prior A-mode ultrasound work highlights cross-subject evaluation as essential and reports similar performance sensitivity to probe repositioning [16][17].

- **Frame retention continuous runs:**
  Continuous data integrity is critical for both live operation and later retraining, especially during long acquisition sessions. The system targets a frame retention rate of at least 95% during continuous 30+ minute runs, ensuring minimal data loss across the logging pipeline. This threshold balances practical hardware and network constraints with the

need for reproducible datasets [18].

- **Logging pipeline throughput:**
  Handle 2× expected publish rate with acceptable consumer lag. Benchmarked headroom protects against bursts and future scaling; Confluent/Kafka docs describe tuning and testing procedures to ensure durable ingestion under load [19] .

3. Scope of the Project ~ *RD*

3.1 In-Scope

The core system will consist of a dual-pipeline architecture: one branch for real-time inference and another for data logging/retraining. The real-time control path is prioritized for responsiveness and will run edge-first (local inference on a workstation or edge GPU).The parallel logging branch records all raw inputs, extracted features, model outputs, and timing metadata for offline analysis and model improvement. This design mirrors industrial IoT and edge-ML best practices, where separating the low-latency control path from a data-analytics path preserves responsiveness while enabling continuous learning [20]. In our project, the in-scope deliverables will include:

Table 1: In-Scope Deliverables

Deliverable	**Description**
Ultrasound acquisition & validation	Build device interface for the *LeCœur US-Builder*; validate signal quality and implement a timestamped frame stream with low jitter.
Acquisition service & bindings	C++ acquisition service with a circular buffer; provide Python bindings for downstream processing and analysis.
Real-time preprocessing & inference (edge-first)	Deterministic preprocessing pipeline (denoise → envelope → normalize → window) to produce model-ready tensors; deploy a compact ML model for wrist-angle/gesture prediction on edge CPU/GPU; implement a short calibration routine mapping predicted wrist angles to screen coordinates and click thresholds.
Asynchronous logging & replay pipeline	Publish preprocessed/raw frames, predictions, and metadata to durable storage via Kafka (or equivalent); provide export and replay tools for offline training, evaluation, and dataset curation.
Control interface & demo	Simple desktop UI that converts model outputs to computer actions (cursor movement, clicks) to demonstrate real-time functionality.
Validation artifacts & datasets	Collect and curate labeled data (target ≥300 labeled clips per gesture, ≥5 subjects where feasible); produce an anonymized dataset snapshot; supply test suites for latency, frame retention, stability, and accuracy.

Monitoring & metrics	Instrument per-hop timestamps for end-to-end latency measurement (target avg < 70 ms, p95 < 90 ms), inference latency (target ≤ 30 ms), and logging retention (target ≥ 95% frames retained over 30+ minute runs).

The primary design objective is to keep the real-time control path local so that the system satisfies the latency and reliability metrics above. The logging path may use cloud storage for dataset persistence and model retraining. replay always proceeds through the same stream pipeline (Kappa-style) to ensure consistent preprocessing.

3.2 Out-of-Scope

To keep the project focused and feasible, the following extensions are explicitly out of scope for this cycle:

Table 2: Out-of-Scipe Deliverables

Deliverable	Reason / Details
Distributed multi-modal inference	Fusion of multiple sensors or distributing inference across devices is excluded due to added complexity and latency. We limit this phase to a single ultrasound modality and single-device processing [21].
Full VR/AR integration	Full immersion and headset integration are out of scope — handling 3D rendering and maintaining motion-to-photon latency (~20 ms) introduces significant delay and engineering effort. Deferred to future work [20].
Anatomically accurate hand reconstruction	We will not attempt full per-joint hand reconstruction. Scope is wrist-pose prediction and gesture classification only, which reduces model complexity and target dimensionality.

These exclusions ensure we do not over-extend our scope. In each case, adding the feature would introduce substantial new challenges (networking, latency, hardware constraints) that would distract from proving the core ultrasound control concept.

3.3 Scope Motivation

The chosen scope focuses engineering effort where it maximizes impact and feasibility, building a working, low-latency interface (edge-first) while simultaneously creating a durable logging pipeline that supports iterative model improvements. This dual-path approach mirrors industrial edge-AI patterns and maps directly to our non-functional targets (latency, retention, accuracy) and the testing plan in *Section 10*. By constraining the modalities, deployment targets, and deliverables, the team can produce a robust, demo-ready prototype and a reproducible dataset

that lays the groundwork for future extensions (multi-sensor fusion, cloud scaling, clinical studies).

4. Literature Review & Related Work ~ *LD*

As of October 2025, a few different research teams have attempted to use wearable ultrasound A-mode sensors to predict hand and/or wrist kinematics, each to different degrees of freedom and levels of accuracy. The most comprehensive study comes from Sgambato et al. at Imperial College [21]. In 2021, they were able to achieve good results in their attempt to predict the wrist motion angles (two degrees of freedom: flexion and radio-ulnar deviation, refer to Figure 3) with an R-squared coefficient of 0.99 using a 24 channel-receiver.

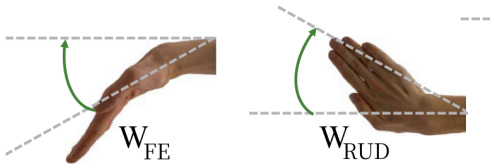

Figure 3. Flexion (Wfe) & Radial-Ulnar Deviation (Wrud) degrees of Freedom (Source: [23])

Interestingly, their modelling algorithm does not leverage machine learning at all. They use linear regression at the end of a complex preprocessing pipeline which is fully described in Figure 4. They initially discard 12 channels due to redundant information provided from parallel sensors, then apply time gain compensation, noise and delay correction before smoothing and computing the envelopes using the Hilbert Transform. Instead of feeding the full-resolution preprocessed envelopes to their regressors, they window the samples (window of size 200) and only take the root mean square. They then perform Principal Component Analysis (top 100 components) to further reduce the dimensions of the input A-mode signals. This study is extremely comprehensive in their descriptions of the methodology they used, and the preprocessing techniques they use will be relevant to our use case.

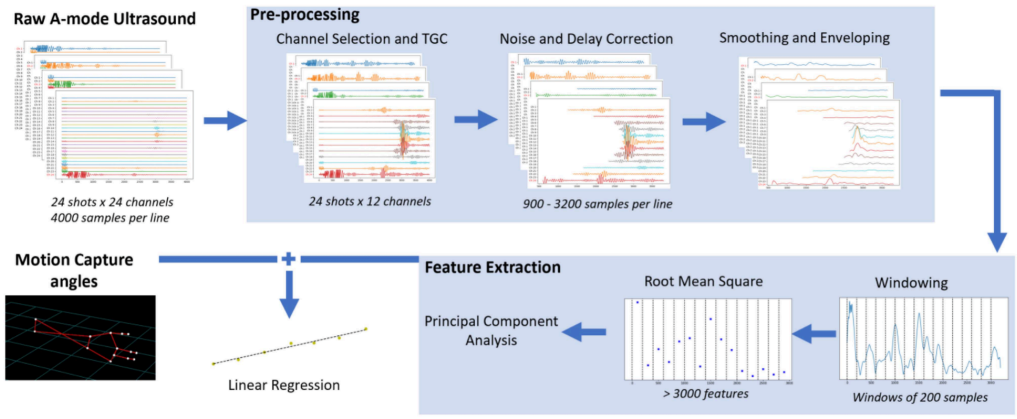

Figure 4. Preprocessing Pipeline used by Sgambato et al., 2021 (Source: [23])

They also present a very interesting comparison study with regards to the impacts of the number of channels used on performance (measured using R-squared). Figure 5 demonstrates that around three sensors seems to provide the most worthwhile increase in performance, before the curve flattens too much.

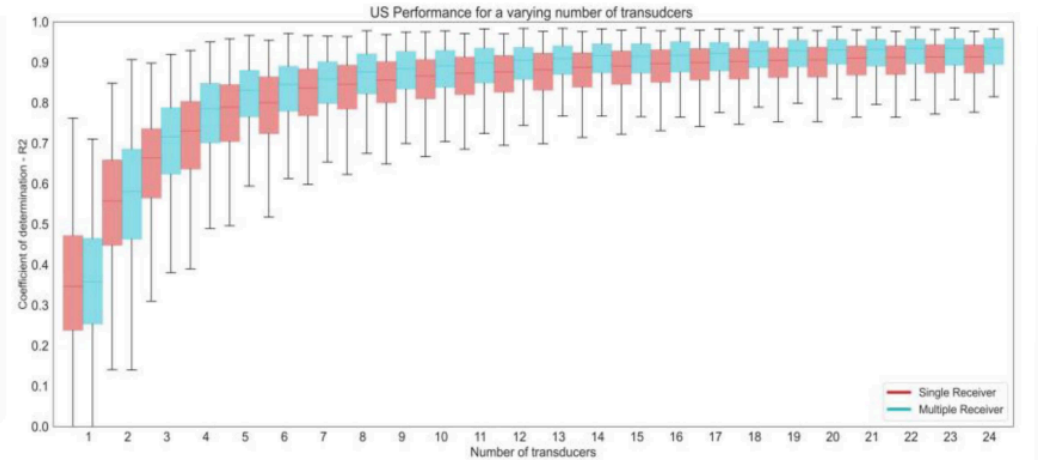

Figure 5. US performance for a varying number of transducers

However, their work presents two obvious limitations, the most glaring one being the sheer size of the 24-channel apparatus. They conclude that it is too cumbersome to be considered wearable outside of a laboratory setting. The second is their decision to only use a linear model for their regressors for each angle. By only using linear regression, they discard the possibility of leveraging non-linear relationships between muscle activations and wrist/finger motions in favour of simplicity and better interpretability. Several other studies also showcase great performance using small convolutional neural networks demonstrating clearly the benefit of using non-linear models for joint regression. [23]

Another paper by Yang et al. (2021) which performs simultaneous wrist and finger motion classification [24], also highlights the usefulness of *Principal Component Analysis (PCA)* as a feature-reduction algorithm. They used *PCA* attempting to correctly identify and separate wrist motion from simultaneous finger motions. They found that the first component is actually linear to the angle of the wrist. This is promising for our system since we will be tracking both wrist motions for the control of the cursor while also classifying gestures for key strokes.

Since there are quite a few other papers attempting similar tasks, albeit using less detailed methodology, a table below summarizes their results. The error metrics are logged, whether the system was tested in real-time and whether any control interface was developed.

Table 3: Survey of Ultrasound-based Hand Gesture Prediction Systems

Paper	**Task**	**Preprocessing**	**Prediction Model**
Novel Method for Predicting Dexterous Individual Finger Movements by Imaging Muscle Activity Using a Wearable Ultrasonic System (Sikdar et al., 2014) [25]	Individual finger flexion classification and proportional speed control using A-mode on the forearm. Real-time? No. Study only reported offline analysis Control Interface? No. No implementation mentioned. Strengths: - High accuracy in single gesture classification. -Low complexity pipeline Weaknesses: - Mechanical Scanning is not viable outside of a laboratory setting. - Method only tested for isolated finger flexions, does not handle simultaneous gestures.	Echogenicity-change maps i.e. computed tables of rest v. contraction brightness.	Flexion Classification: Correlation-based template matching (ultrasound echo pattern changes) Results: - 98% accuracy - 97% recall - 98% precision (classifying which finger performed a flexion)
Ultrasound vs. sEMG for Finger Motion & Joint Angle Estimation (Huang et al., 2018) [24]	Multi-finger gesture classification (14 gestures) and MCP joint angle regression using b-mode on the forearm. Real-time? No. Offline analysis only. Control Interface? No. No implementation mentioned. Strengths:	- Segmentation of a region of interest on the forearm. - Frame averaging - Gaussian Filtering	Joint Angle Regression: - Multi-linear Regression Results: - R value = 0.89 Gesture Classification: - Support Vector Machine - Linear Discriminant Analysis Results: - 95.9% accuracy

	Confirms ultrasound outperforms emg in finger gesture classification tasks. Weaknesses: Offline only, and the system is not portable.		
Voluntary & FES-Induced Finger Movement Estimation via Ultrasound (Zhou et al., 2020) [26]	Finger joint angle regression (voluntary & stimulated motion) using 4-channel A-mode on the forearm. Real-time? No. Offline analysis only. Control Interface? No. No implementation mentioned. Strengths: Their system is actually wearable outside of a laboratory setting. They demonstrate good accuracy can be achieved without a bulky non-wearable setup. Weaknesses: They only look at middle finger contractions, 1 degree of freedom, therefore their results might not scale to multi-degree of freedom applications.	- Time-gain compensation - Gaussian Filtering - Hilbert Envelope - Log compression - Principal Component Analysis	Joint Angle Regression: - Linear Ridge Regression -Least Squares Support Vector Machine Results: - R-value of 0.90

Simultaneous Wrist and Hand Motion Prediction using Ultrasound (Yang et al., 2020) [27]	Continuous wrist rotation regression and 8 discrete finger gesture classification using b-mode. Real-time? Yes. Their error metrics were measured during live experiments. Control Interface? Yes. Small GUI where the user would attempt to make a cursor track a waveform with wrist rotation/ Strengths: The SDA technique is very interesting, and their paper shows that the first principal component of the ultrasound after having been pre-processed with SDA directly linearly maps to the wrist rotation. Weaknesses:	- Time-gain compensation - Subclass Discriminant Analysis - Principal Component Analysis	Regression: - Principal Component Analysis (PCA) - Linear regression Results: - R-squared value = 0.954 Classification: - Subclass Discriminant Analysis Results: - 96.5 % accuracy

	Only one degree of freedom for the wrist motion prediction. No robustness or generalization impact. The testing is done with the user only being able to perform one of the 8 pre-defined gestures, reliability outside of these defined gestures is not mentioned.		
Wearable Real-Time Hand Gesture Recognition with A-mode US (Lu et al., 2022) [28]	Classification of 10 finger gestures using 4-channel A-mode Real-time? Yes. This paper focuses heavily on being able to reliably detect gestures with low-latency in a real-time environment. Control Interface? No. The gestures were only classified by the system in real-time, no control interface was implemented on top of these predictions. Strengths: Great blueprint for low latency C++ pipeline implementation. Extremely detailed report on metrics for latency and generalization across different subjects in a real-time environment. Weaknesses: Limited to discrete gestures. Huge performance decrease with sensor shift (96% -> 84%)	- Gaussian Filtering - Principal Component Analysis (All implemented in C++)	Gesture Classification: - Linear Discriminant Analysis - Support Vector Machine - Naive Bayes Results: - 96% (for both LDA & SVM)

After a review of the literature, it is clear that no single system currently offers all of these properties:

- Real-time full range of motion wrist tracking with simultaneous multi-finger gestures classification.
- Robustness across multiple users and sessions with no significant performance degradation.
- Robustness to sensor shifts and variance.
- Wearable outside of laboratory setting.
- Integration of a low-latency general keyboard/mouse interface for Windows/Linux.

Therefore, it is clear that our system would be a novel contribution. It is also interesting to note that multiple approaches to the classification/regression problems are similar across studies.

For the preprocessing phase, three different techniques appear repeatedly:

- **Time-gain compensation:**

In ultrasound acquisition, the deeper echoes have lower amplitude due to tissue attenuation. In order to compensate for this loss of energy, time-gain compensation linearly increases the amplitude of the signal as the depth increases. This will improve the quality and the uniformity of the A-mode data. [29]

- **Gaussian filtering**:

  A simple low-pass filter which smooths the signal by suppressing high-frequency noise while preserving its peaks and structure. [30]

- **Principal Component Analysis:**

  Principal Component analysis is a technique used to reduce the dimensionality of data. It selects the top component of the data by prioritizing directions where the data varies the most, since more variance implies more information [31]. This dimensionality reduction is useful since we will be working within limited computing constraints.

As for the prediction models widely used, three modeling techniques commonly appear:

- **Support Vector Machines:**

  Supervised machine learning algorithm that is commonly used for classification tasks. It tries to find the best boundary (hyperplane) between the different classes of data, which in our case will be different gestures. It can be implemented to find linear or non-linear planes. [31]

- **Linear Discriminant Analysis:**

  Linear Discriminant analysis is another supervised classifier that attempts to build planes between different classes from high-dimensional data, but makes the assumption that the populations have the same covariance and that the data is normally distributed. [31]

- **Convolutional Neural Networks**:

  Neural network with learnable convolutional filters that leverage spatial locality to learn features from high-dimensional data before feeding them into classifiers or regressors. Most modern technique out of the ones mentioned. [32]

5. System Design ~ *TH*

5.1 Overview

The proposed system follows a Kappa architecture design pattern, enabling the system to cast real-time predictions of hand movements through continuous ultrasound sensor data. The architectural decision is based on the confidence in scalability, fault tolerance, and the ability to support continuous learning through replay and monitoring, which are essential for the development of a real-time machine learning model. Because of its simplified design, replay capability, and consistent data flow for both online and offline processing, the team chose the Kappa architecture over the Lambda architecture [33][34].

5.2 Background

When choosing an architecture, there are two primary approaches to handling data: batch processing and stream processing. Batch processing refers to processing a vast amount of data (typically historical) within a specific time span. On the other hand, stream processing is the continuous, real-time processing of data (live data). Due to the nature of the project, the model requires live data in order to make instant predictions in real-time. Therefore, stream processing was chosen as the method to handle data; this leads us to two architectures to choose from: Kappa and Lambda Architecture.

5.3 Justification of Kappa Architecture

The Kappa architecture was chosen over the Lambda architecture because of its simple design, which eliminated the batch layer that stored incoming data and periodically processed batches. The batch layer comes at a high cost of latency, which is a major requirement for the real-time system. Instead, the Kappa architecture follows the idea of performing stream processing for all operations: ingestion, storage, and serving. This enables the system to handle both streaming and batch-like processes on a single stream-processing technology [35][36].

Table 4: Comparison of Kappa and Lambda Architectures [35][36]

Feature	Kappa Architecture	Lambda Architecture
Data Flow Design	Single unified stream processing pipeline that handles both real-time and historical data.	Dual-layer design with separate batch and stream layers for processing data.
Complexity	Simpler design since only one codebase is maintained for all data.	Higher complexity due to maintaining two different pipelines for batch and

		streaming data.
Latency	Very low latency due to continuous real-time processing.	Slightly higher latency since batch results are merged with streaming results periodically.
Maintenance Effort	Easier to maintain and debug since the same logic is applied across all data.	Requires duplicate logic for batch and stream layers, increasing maintenance effort.
Model Retraining	Retraining is handled by replaying data through the same stream pipeline.	Retraining typically depends on batch recomputation using stored datasets.
Use Case Fit	Real-time ML systems	Analytics and long-term reporting

5.4 Kappa Architecture

As illustrated in *Figure 6*, the Kappa Architecture simplifies the Lambda model by removing the batch layer and processing all data as a continuous stream. It uses an immutable log and stream-processing engine to manage both real-time and historical data through event replay, which reduces complexity, enhances scalability, and ensures consistent processing. For the proposed wearable ultrasound system, the Kappa model offers a strong framework for continuous signal acquisition, low-latency inference, and adaptive retraining. The following subsections describe each architecture layer: data acquisition, stream ingestion, stream processing, output sink, and monitoring [37].

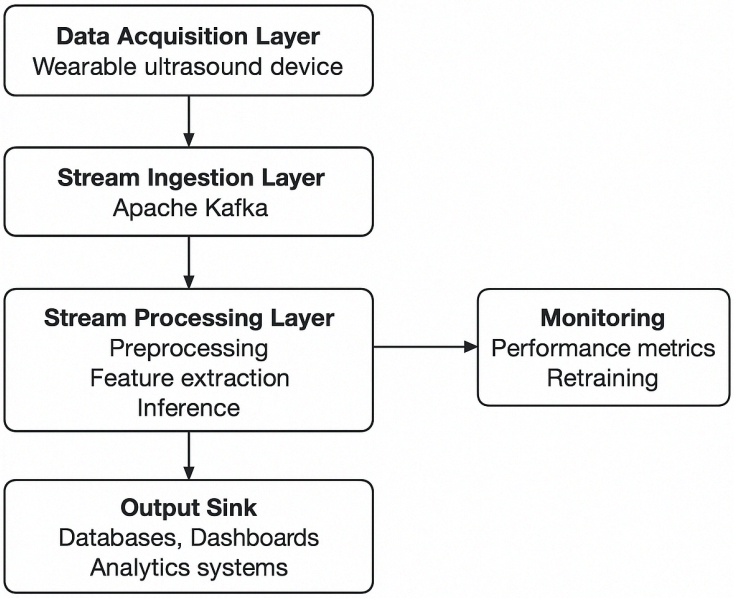

Figure 6. Kappa Architecture for Real-Time Ultrasound Data Processing

Data Acquisition Layer

The data source is a wearable ultrasound device designed to monitor wrist muscle activity, coupled with a microcontroller-based acquisition platform (*Le Coeur Électronique US Builder*). This hardware continuously captures analog ultrasound waveforms that correspond to wrist muscle activity. The microcontroller digitizes the signals using its on-board ADC. It streams the data to the ingestion layer via a wireless or serial connection. Each data frame is timestamped and serialized. This ensures temporal consistency for downstream processing.

Stream Ingestion Layer

The system uses a distributed message broker such as Apache Kafka to ingest and buffer real-time data. Kafka provides an immutable, ordered event log, enabling both real-time processing and historical replay. Each ultrasound frame is a discrete Kafka topic event, so consumers like preprocessing modules, ML inference services, and monitoring systems can subscribe independently. This structure enhances scalability, resilience, and reproducibility, essential for retraining and model validation [38].

Stream Processing Layer

The stream processing engine—implemented using frameworks like Apache Flink or Kafka Stream; performs the core data transformations and inference tasks.

The processing pipeline executes:

- Preprocessing: Noise filtering, normalization, and segmentation of continuous ultrasound data into time windows.
- Feature Extraction: Generating statistical and time–frequency domain features relevant to hand motion prediction.
- Inference: Applying the current ML model to generate real-time hand position or gesture predictions.

In Kappa Architecture, the same processing logic ensures consistency for both live and replayed data, keeping retraining pipelines and inference pipelines consistent [39].

Output Sink and Monitoring Layer

The system outputs processed data and model predictions to different output sinks, each responsible for a unique function in the data pipeline. These are databases for archiving, dashboards for real-time visualization, and downstream analytics systems for further

interpretation of the data. Among these, the machine learning model itself is one of the primary output sinks, consuming processed feature streams from Apache Flink and outputting real-time hand motion predictions from the ultrasound input data.

Model output, along with intermediate quantities such as feature vectors or confidence, are pushed back to the Kafka event stream. This makes other system modules, such as monitoring or retrain modules, able to access live and historic output for analysis.

A dedicated monitoring service continuously tracks key performance metrics like prediction accuracy, latency, and signs of model drift. When significant degradation or behavioral drift is identified, the system can trigger replaying past event data from Kafka to retrain the model and resubmit an updated version seamlessly.

This closed-loop structure ensures that the system will remain responsive to changing signal properties, sensor drift, and inter-individual variability in a long-term manner. By integrating the model and monitoring feedback into a unified event-driven framework, the structure achieves both real-time responsiveness as well as long-term learning stability.

5.5 Architectural Rationale and Summary

The Kappa Architecture offers a good trade-off between system responsiveness and real-time adaptability. It does away with the redundancy in the Lambda Architecture, which has two batch and streaming layers but retains the ability to reprocess older event logs for model retraining; a key aspect of machine learning systems based on continuous sensor inputs.

By unifying the data flow into a single, stream-based pipeline, Kappa simplifies system design, reduces maintenance overhead, and ensures scalability and fault tolerance for both live and historical data processing. The architecture is a lean, maintainable setup that provides uninterrupted learning and high-speed real-time processing of ultrasound signals.

By employing the Kappa Architecture, the proposed system is able to achieve:

- Low-latency, real-time inference of hand motion from ultrasound data,
- Continuous retraining with replay of event logs and continuous monitoring,
- Simple development and maintenance in a single, unified processing pipeline.

Overall, this design choice is consistent with the objective of the project to develop a strong, scalable, and adaptable wearable ultrasound platform that will generate accurate hand movement predictions in real-time.

6. Implementation Plan ~ *RD*

This section outlines the project's organizational framework and coordination strategy. The project is structured into six main sections: (1) Project Management & Coordination, (2) Hardware and Data Acquisition, (3) Data Pipeline and System Architecture, (4) Machine Learning Development, (5) Evaluation and Testing, and (6) Documentation & Deliverables. Each section represents a key phase of progress, with well-defined dependencies and deliverables. However, several activities—such as hardware setup, data acquisition, and model prototyping—will proceed in parallel to maximize development efficiency and reduce idle time across the team. The accompanying Work Breakdown Structure (WBS) and Gantt chart detail the scheduling, resource allocation, and overlapping workflows that guide project execution, ensuring that technical milestones and reporting deliverables remain aligned with course deadlines and supervisor expectations.

Table 5: Milestone Outline

ID	Milestone (short)	Owner	Start (wk)	End (wk)	Duration	Depends on	Acceptance (one line)
1.1	Kick-off & Scope Alignment	RD, LD, TH	Wk 1	Wk 1	1 wk	N/A	Kickoff summary document with agreed project scope.
1.2	Communication & Meetings (ongoing)	RD, LD, TH	Wk 1	Wk 16	ongoing	1.1	Updated project logbook and meeting minutes.
1.3	Repository & Tool Setup	RD, LD, TH	Wk 1	Wk 1	1 wk	1.1	Functioning repository with documented structure.
2.1	Hardware Familiarization	RD	Wk 2	Wk 2	1 wk	1.3	Device verified functional with initial ultrasound stream.
2.2	Communication Setup	RD	Wk 2	Wk 3	1–2 wks	2.1	Successful streaming of ultrasound frames to acquisition app.
2.3	Data Buffering & Timestamping	LD	Wk 3	Wk 4	2 wks	2.2	Consistent timestamped stream without data loss.
2.4	Real-time Validation	LD	Wk 5	Wk 5	1 wk	2.3	Validated <5 ms latency from hardware to system memory.
3.1	Data Cleaning & Formatting	TH, LD	Wk 6	Wk 7	2 wks	2.4	Verified clean dataset ready for ML input.
3.2	Pipeline Architecture	RD, TH	Wk 8	Wk 9	2 wks	3.1	Pipeline transmits data to inference service reliably.

3.3	Cloud Storage Integration	RD	Wk 9	Wk 9	1 wk	3.2	Verified cloud upload and retrieval process.
4.1	Baseline Model Training	TH, LD	Wk 10	Wk 11	2 wks	3.3	Baseline model trained with reproducible results.
4.2	Inference Integration	RD, TH	Wk 12	Wk 13	2 wks	4.1	Real-time inference pipeline functioning end-to-end.
4.3	Latency Optimization	LD	Wk 12	Wk 13	2 wks	4.2	Reduced total pipeline latency (target <120 ms).
4.4	Generalization Testing	TH	Wk 14	Wk 15	2 wks	4.3	Validated model robustness with >90% baseline accuracy.
5.1	Latency Measurement	RD	Wk 14	Wk 14	1 wk	4.3	Latency report (<150 ms end-to-end).
5.2	Accuracy & Reliability Tests	LD, TH	Wk 15	Wk 16	2 wks	4.4	Accuracy >85% across sessions.
5.3	Usability Evaluation	RD	Wk 15	Wk 15	1 wk	5.2	Usability report with ≥80% satisfaction score.
5.4	Refinement Iteration	LD, TH, RD	Wk 16	Wk 17	2 wks	5.3	Improved model + final system stability report.
6.1	Proposal Submission	RD, LD, TH	Wk 1	Wk 2	1–2 wks	1.x–2.x	Submitted proposal.
6.2	Progress Report	RD, LD, TH	Wk 8	Wk 9	1–2 wks	2.x–4.x	Submitted progress report.
6.3	Oral Presentation	RD, LD, TH	Wk 9	Wk 9	1 wk	6.2	Successful presentation delivery.
6.4	Final Deliverables	RD, LD, TH	Wk 16	Wk 17	2 wks	5.x	Submitted final package (poster, video, report).

Table 5 is a compact, high-level summary of the project milestones designed to keep Section 6 easy to scan. For full task descriptions, detailed acceptance tests, owners and deliverables, see *Appendix A — Table A.1* (Detailed Task Table), where each row in the appendix is cross-referenced by the milestone ID in Table 5.

6.1 Work Breakdown Structure

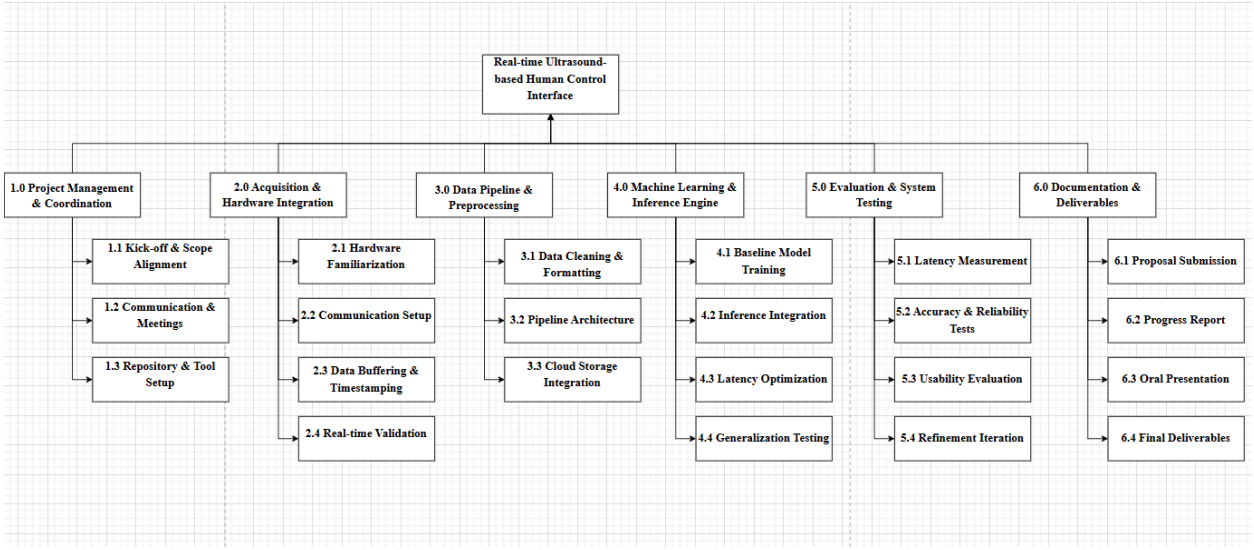

Figure 7: Work Breakdown Structure

The Work Breakdown Structure (WBS), shown in *Figure 7 above,* organizes the project into a clear hierarchy of deliverables and tasks. Level-1 nodes represent the six major project phases: Project Management, Hardware & Acquisition, Data Pipeline & Architecture, Machine Learning Development, Evaluation & Testing, and Documentation & Deliverables. Each Level-2 node decomposes a phase into discrete work packages (e.g., Data Buffering & Timestamping). The WBS numbering (e.g., 3.2 → Pipeline Architecture) is used throughout the schedule and milestone tables to indicate dependencies and ownership. Use the WBS as the authoritative map for scope, resource allocation, and change control.

6.2 Gantt Chart

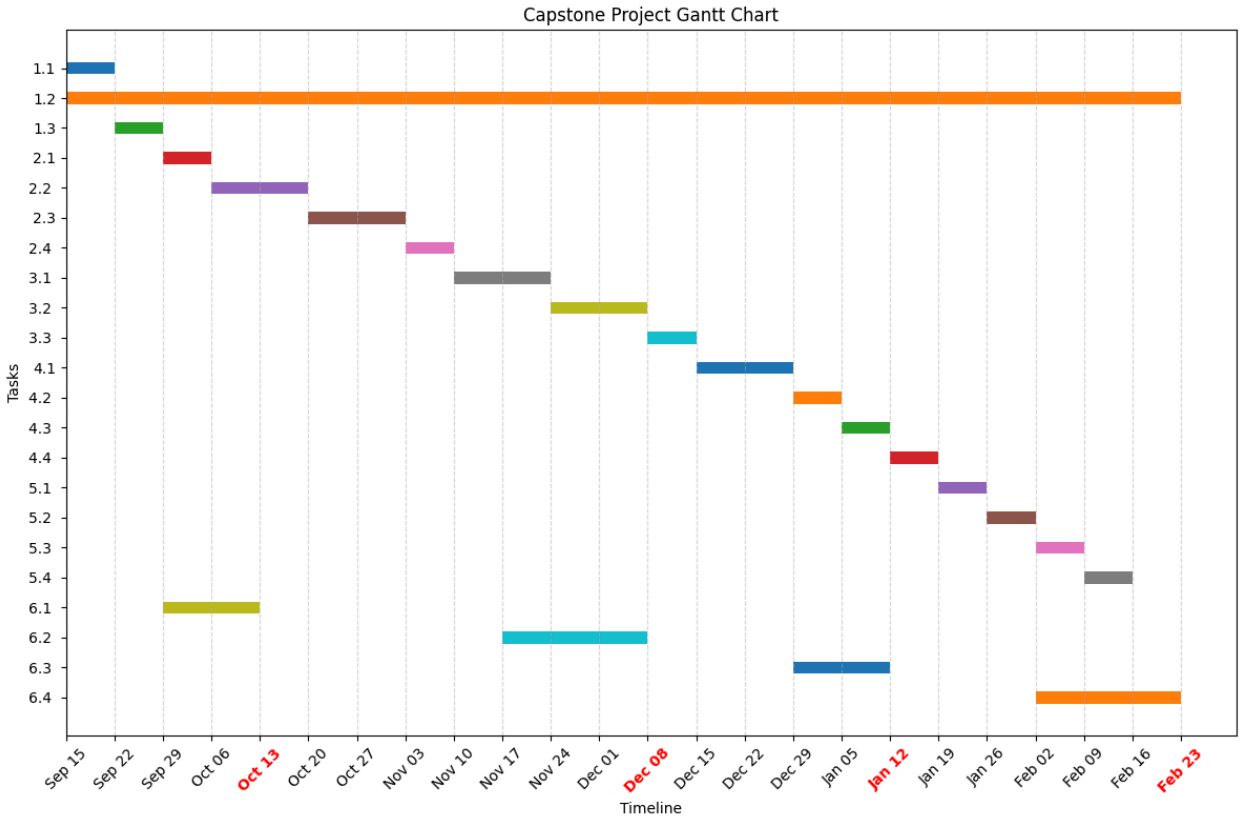

Figure 8: Gantt Chart

The Gantt chart, *figure 8* above, visualizes schedule, duration, dependencies and overlap across the WBS work packages. Bars show planned start and end dates for each major task. The chart highlights the critical path (longest chain of dependent tasks) so the team can prioritise actions that would delay project completion. The three main

7. Tools and Technologies ~ *TH*

The solution integrates hardware, software, and support technology to capture, process, and analyze real-time ultrasound data. Data acquisition is handled by wearables sensors and microcontrollers, data processing and model training is handled by streaming and ML platforms, and support tech enables scalability, reliability, and efficient performance monitoring.

7.1 Hardware Components

Table 6 outlines the key hardware components of the wearable ultrasound data acquisition system, including devices for signal capture, processing, and transmission. Each component contributes to reliable real-time data collection and preparation for machine learning analysis.

Table 6: Overview of Hardware Components for the Wearable Ultrasound System

Component	Description
Wearable Ultrasound Device	Captures muscle activity signals from the wrist. The device provides analog waveform data representing muscle contractions that correlate with specific hand movements.
Le Coeur Électronique US Builder Board	Serves as the main acquisition platform for the ultrasound transducer. It includes an analog front-end and microcontroller support, enabling reliable real-time signal digitization and streaming.
Microcontroller / Embedded Processor (e.g., ESP32, STM32, or Arduino-compatible)	Responsible for preprocessing and transmitting sensor data to the streaming layer. These boards are lightweight, energy-efficient, and capable of handling real-time communication.
Host Computer / Edge Node	Manages data ingestion and stream processing through platforms such as Apache Kafka and Flink. It performs feature extraction and forwards structured data to the machine learning module.

7.2 Software Components

Table 7 lists the main software components supporting data processing, streaming, and machine learning within the system. It includes programming languages, frameworks, and platforms used for real-time data handling, model training, and deployment.

Table 7: Summary of Software Components Used in the System

Component	Description
Programming Languages	• C / C++ for embedded firmware and low-level data acquisition. • Python for stream orchestration, feature extraction, and ML model training. These were chosen due to their high performance and extensive support in embedded and ML ecosystems.
Apache Kafka	Acts as the primary event streaming platform for ingesting continuous ultrasound data. Kafka ensures durability and supports replay for retraining under the Kappa model.
Apache Flink	Processes streaming data in real time to extract relevant features before model inference. Its event-time processing and fault tolerance make it suitable for biomedical signal pipelines.
PyTorch	An open-source deep learning framework for building, training, and deploying models. It offers dynamic computation, GPU acceleration, and seamless integration with cloud and edge environments.
Google Colab	Used as the main development and training environment for the ML model. Colab provides GPU acceleration, integrated notebooks for experimentation, and seamless integration with Google Drive for dataset management and model checkpointing.

7.3 Other Technologies

Table 8 presents additional technologies that support system development, deployment, and monitoring. These tools enable version control, containerization, cloud storage, and performance visualization, ensuring reliability and scalability across the data pipeline.

Table 8: Summary of Supporting Technologies Used in the System

Component	Description
Git and GitHub	Used for version control and collaborative code management, ensuring proper documentation and reproducibility of development progress.
Docker	Employed for containerizing the data pipeline components, allowing consistent setup across local and cloud environments.
Apache Zookeeper	Coordinates Kafka brokers and ensures distributed system consistency.
Google Drive and Google Cloud Storage	Provide cloud-based storage for datasets, trained models, and logs. Integration with Colab ensures fast access to model checkpoints and datasets.

Grafana and Prometheus	Used for system monitoring, including latency tracking, data throughput, and inference performance visualization.

8. Expected Outcomes ~ *RD*

The project will deliver a working, dual-path prototype that converts wrist ultrasound data into real-time desktop controls while concurrently logging data for offline retraining and analysis. The system will demonstrate that ultrasound-based wrist motion can be used to control a standard computer interface (pointer movement and click gestures) with a reproducible data pipeline for iterative model improvement.

8.1 Deliverables (concrete artifacts)

As shown in Table 9, the project will produce a complete software and documentation package suitable for demonstration and reproducibility:

Table 9: Project Concrete Deliverables

Deliverable	Specification
Working prototype	End-to-end demo connecting US-Builder → acquisition service → preprocessing → inference → UI control.
Monorepo	All code organized in a single repository with subfolders for acquisition, preprocessing, model serving, and UI. Includes run scripts and Docker dev configuration.
Model artifacts	Baseline and optimized model checkpoints and inference exports (ONNX / TorchScript / TensorRT where appropriate).
Dataset snapshot	Anonymized, labeled ultrasound clips and metadata (target dataset described in Section 3 and Section 6).
Logging & replay tools	Kafka topic schemas, ingestion/consumer scripts, and utilities to replay archived streams for retraining.
Documentation & media	README, demo video, and a short usability summary.

8.2 Features & User interactions

From the user perspective, the prototype will provide real-time pointer control mapped to wrist flexion/extension, gesture-based click actions (single, double, right-click, left-click), and a brief calibration routine that maps each user's motion range to screen coordinates.A minimal status UI will display connection health, signal quality, model confidence, and latency.

From the developer perspective, the monorepo will support two deployment modes co-located and distributed. Co-located operations, where all services run on the same machine, will use low-latency IPC for intra-host communication. For distributed operation, where services run on separate machines, communication will be done over network ports and Kafka topics. The codebase will include FastAPI endpoints for model queries and telemetry, Kafka consumers/producers for logging, and scripts to run the retraining pipeline that consumes archived data.

8.3 Validation and Demonstration

The project will be validated through a combination of automated benchmarks and human trials. *Section 10* describes the specific metrics and test procedures (latency, frame retention, accuracy, robustness, and usability).

Demonstration will include users performing point-and-click and drag-and-drop tasks using wrist motion to control the pointer. Demo will show the UI, live model confidence, and latency counters.

8.4 Impact & beneficiaries

The deliverables will provide value for multiple audiences. For HCI and VR researchers, the system offers a camera-free, occlusion-resistant sensing modality for hand/wrist tracking. For accessibility research, the prototype shows a plausible path toward alternative input modalities for users who cannot rely on standard mice or cameras. For biomedical researchers, the logged ultrasound data and toolchain create a reproducible means of quantifying wrist motion in laboratory experiments. For systems and *Machine Learning Operations (MLOps)* practitioners, the project demonstrates a practical dual-path architecture that balances low-latency edge inference with a robust logging and retraining pipeline.

8.5 Limitations & Future Work

The current scope focuses on single-user wrist-level control on desktop/workstation hardware and does not include mobile/embedded optimization, full hand reconstruction, clinical validation, or full VR/AR integration. Future work will use the logged dataset and modular codebase to extend the system to additional modalities (e.g., finger-level tracking, eye-tracking fusion), cloud scaling, and embedded deployment after prototype validation.

9. Budget and Cost Analysis ~ LD

Wearable ultrasound (WUS) is a cost-efficient alternative to camera-based setups due to their relative simplicity. For our project, the hardware cost and its associated information is described in more details in table 10, below.

Table 10: Hardware Cost Overview

Component	Cost	Source	Information
US-Builder (Transmitter/Receiver/Digitizer/DAC)	3,500 $	*LeCoeur Electronique*	Provided by Dr. Ono Laboratory
Control Link (Bluetooth/USB)	N/A	*LeCoeur Electronique*	Bluetooth and USB modules are included with the US-Builder
Single-element Transducer	*50$*	*Misc*	Sourcing TBD.
Probe Cable	*20$*	*Misc*	Sourcing TBD.
Coupling Gel	*15$*	*Misc*	Sourcing TBD.
LeapMotion Camera Sensor	40 $	*UltraLeap*	Highest accuracy for wrist and hand kinematics annotation for the cost.
PC with RTX 3090	N/A	*Nvidia*	*Provided by Leo D.*

The most expensive component is the US-Builder by LeCoeur which encompasses all necessary hardware for the A-mode ultrasound acquisition. This system was provided to us by Dr. Yuu Ono's laboratory. The LeapMotion device used to capture angle baselines using IR and video was specifically selected for its price-to-performance ratio, hence the affordability.

All of these associated costs are related to the ultrasound acquisition system, the subsequent work on the machine-learning model will be at no cost due to us already possessing the hardware at home.

As for software costs, every level of our stack from metal to the machine-learning engine will be leveraging the following open-source, free to access software and libraries.

- C++ / Python
- Pytorch Library
- CUDA / TensorRT Libraries
- Apache Kafka/Flink/Zookeeper
- UltraLeap Gemini SDK
- Docker
- Git/Github

The only exception will be the limited use of Google Colab, a hosted Jupyter Notebook service which provides access to cost-efficient computing resources. It will incur a 15.99 $ monthly cost. Colab will be used to quickly prototype different machine-learning models before training them on our discrete local GPU.

10. Evaluation and Testing ~ *RD*

This project's evaluation plan verifies that the implemented system meets the functional and non-functional objectives described in *Section 2*. Tests combine automated measurements (latency, throughput, retention), quantitative model evaluation (accuracy, false positives), robustness/stability trials, and small-scale usability studies. Each test is designed to exercise the real-time (edge-first) control path and the parallel logging/replay path so we can demonstrate both immediate responsiveness and reproducible dataset quality for retraining. Table 11 summarizes the evaluation metrics, test methods, and corresponding validation criteria used to verify system performance and reliability.

Table 11: Test Matrix

Metric / Goal	Test method	Validation / Pass criteria
End-to-end latency (avg / p95)	Insert synchronized timestamps at key hops (device acquisition, post-preprocess publish, server receive, post-infer return, UI update). Collect ≥1,000 samples in both local and remote configurations.	avg < 70 ms and p95 < 90 ms for local (edge) inference path. Remote inference measured separately for comparison.
Inference latency	Measure model inference time (model input → model output) with over 1k runs on target hardware.	median ≤ 30 ms and p95 documented.
Frame retention / throughput	Continuous streaming test (30–60 minutes), Count frames produced by device vs. frames persisted by logging stack.	≥ 95% retained over 30-60 minutes of capturing
Model accuracy	Train/evaluate on held-out test sets using cross-validation; report accuracy, precision, recall, F1, and confusion matrix vs. ground truth.	Test accuracy ≥ 85%

False click rate	Run dedicated click-gesture sessions and measure spurious clicks during neutral motion.	False click rate ≤ 5%
Robustness (probe shift / cross-subject)	Repeat inference tests with small probe displacements (e.g., 1–2 cm / ± angle) and with multiple subjects (≥10).	Performance drop ≤ 10% relative to baseline Document failure modes.
Usability	Pilot with 5–10 participants performing standard UI tasks (point, click, drag). Measure success rate, completion time vs. baseline mouse, and subjective lag rating	Task success ≥ 90% Mean completion time ≤ 110% of baseline mouse in controlled tasks.
Stability / Resilience	30+ minute continuous runs. Kafka stress test at 2× expected throughput.	No crashes Consumer lag controlled
Retraining loop demonstration	Use archived logs to retrain and redeploy updated model Compare before/after metrics on validation set.	The retrained model shows measurable improvement (accuracy or robustness) on hold-out data.

11. Ethical, Environmental, and Safety Considerations ~ *RD*

This project places priority on responsible treatment of participants and safe operation of both hardware and software. Our focus is on two main areas, (1) obtaining an informed concert of participants and following ethical research guidelines, (2) maintaining physical and system safety during ultrasound use and control interface testing.

Working Responsibly with Human Participants

All participant involvement in this project will follow established research ethics principles. Before any data collection begins, participants will receive a clear explanation of the project's purpose, what type of data will be collected (ultrasound signals and timing information), and how it will be used. Written informed consent will be obtained from each participant, following the ethical standards set by Canada's *Tri-Council Policy Statement: Ethical Conduct for Research Involving Humans (TCPS2)* and institutional Research Ethics Board (REB) guidelines [40][41] .

Only the minimum amount of data needed for analysis will be retained, and participants will have the right to withdraw their data at any point before it is analysed. All procedures and consent materials will be reviewed or exempted through the university's REB process, and any revisions will be documented in the final report. This ensures that our work meets ethical research standards while maintaining transparency and respect for participant autonomy.

Ultrasound and System Safety Measures

The ultrasound hardware will be operated strictly within the manufacturer's recommended acoustic power and duty-cycle limits to ensure participant safety. All sessions will adhere to the *As Low As Reasonably Achievable (ALARA)* principle for ultrasound exposure, as recommended by the *American Institute of Ultrasound in Medicine (AIUM)* and similar professional bodies [42][43].

On the software side, additional safety and security measures will be built into the system. Network communications between services (e.g., acquisition, inference, and logging) will use encryption protocols that meet *National Institute of Standards and Technology (NIST) Guidelines for Transport Layer Security (TLS)* [44]. Data stored locally or in the cloud will be encrypted at rest and protected by access-controlled permissions. The user-interface software will default to conservative behavior, avoiding any automated or irreversible actions and will revert to a safe idle state if the model produces unexpected outputs or if network connectivity is lost. Together, these safeguards help ensure both physical and digital safety during all project activities.

12. Group Skills & Academic Relevance ~ *LD, RD & TH*

Table 12: Group Skills

Area/Task	Tharusha	Leo Computer Systems	Ranveer Software Engineering	Evidence (course / experience)
Hardware integration & drivers	★	★★	★★	LD: CSE Architecture classes
C++ & performance code	★★	★★★★	★	LD: Co-op C++
Python & ML frameworks	★★★	★★★★★	★★★★★	RD: SYSC 4415, Co-op ML pipelines LD: Research experience in applied ML

				TH: Online Courses
Messaging / deployment	★★★	★★★	★★★	All: SYSC 3303
UX / calibration / demo	★★★★★	★	★	TH: Co-op experience
Data collection	★	★★★★★	★★★★★	RD: Co-op experience TH: Co-op experience

In table 12, each area of the project has an associated grade from zero to five for every team member. This grade indicates the level of familiarity or experience we have with tasks of similar nature.

Leo D. is a computer systems engineering student, with some research experience in the field of applied machine-learning. He also wrote extensive code in C++ during his co-op terms in industry. He will be able to help with the training of the machine-learning model and good design practices in C++.

Tharusha H. is a software engineering student with experience in front-end web development gained through internships. He has also completed several online courses in machine learning, providing him with a solid understanding of model development. Tharusha will be able to contribute to the design and implementation of the system's user interface as well as assist in the integration of machine-learning components.

Ranveer D. is a software engineering student with strong experience in Python-based data analysis and machine learning, developed through previous co-op terms focused on building ML pipelines and data collection tools. He has worked extensively with model evaluation, networked data flows, and visualization frameworks. Through coursework such as SYSC 3303, he has also gained experience in messaging and deployment architectures, which will support the project's integration, real-time communication, and system testing efforts.

13. Risks and Mitigation Strategies ~ LD

Due to the large scope of the project, there are multiple potential failure points that must be evaluated and mitigated for. For clarity and due to their differing natures, the two tables below identify and split the main risks relating to our:

I. Machine-learning Model
II. Real-Time WUS Acquisition System

Table 13. Potential Risks Relating to Development Cycle of the Machine-Learning Model

Risk	Likelihood	Impact	Sub-stream	Mitigation
High-latency	High	High	Inference	- Increase Compute (Dedicated GPU) - Shrink model size. - Dimensionality Reduction - Model Quantization. - Lighter weight architectures
Poor Generalization Across Subjects	High	Med	Training/ Inference	- Increase dataset diversity (i.e. more test subjects) - Unsupervised training loop to learn per-subject bias
Sensor Placement Variance	High	High	Training/ Inference	- Increase dataset diversity (i.e. introduce sensor variance) - Data Augmentation
Unstable Model Convergence	Low	High	Training	- Different loss functions / scheduling - Data Augmentation
Ground-Truth Reliability	Low	Low	Training	- Use Mediapipe - Combine LeapMotion data with other sensors

Table 14. Potential Risks Relating to Development of Real-Time WUS Acquisition System

Risk	Likelihood	Impact	Stream	Mitigation
High-latency	High	High	Software	- Optimize memory management - Limit computations on Arduino board
Poor Hardware/Firmware Documentation	High	Med	Hardware	- Email LeCoeur company directly - Pair-programming
Priority Inversion due to ISR's	Med	Low	Software	- Priority inheritance - Priority ceiling protocols - Minimize resources
Data Integrity	Med	Low	Software	- Error Checking - In-depth Logging

14. References

[1] W. Chen, C. Yu, C. Tu, Z. Lyu, J. Tang, S. Ou, Y. Fu, and Z. Xue, “A Survey on Hand Pose Estimation with Wearable Sensors and Computer-Vision-Based Methods,” Sensors, vol. 20, no. 4, p. 1074, Feb. 2020, doi: 10.3390/s20041074.

[2] Meta, “Troubleshooting and Limitations — Hand Tracking,” Meta Horizon OS Developers, 2025. [Online]. Available: https://developers.meta.com/horizon/documentation/unity/unity-handtracking-troubleshooting-limitations/. Accessed: Oct. 17, 2025.

[3] X. Yang, J. Yan, Y. Fang, D. Zhou, and H. Liu, “Simultaneous Prediction of Wrist/Hand Motion via Wearable Ultrasound Sensing,” IEEE Transactions on Neural Systems and Rehabilitation Engineering, vol. 28, no. 4, pp. 970–977, Apr. 2020, doi: 10.1109/TNSRE.2020.2977908.

[4] “Machine learning is going real-time,” huyenchip.com. https://huyenchip.com/2020/12/27/real-time-machine-learning.html

[5] “US-BUILDER,” Lecoeur-electronique.net, 2025. https://www.lecoeur-electronique.net/us-builder.html

[6] Elfi Fertl et al., “End-to-End Ultrasonic Hand Gesture Recognition,” Sensors, vol. 24, no. 9, pp. 2740–2740, Apr. 2024, doi: https://doi.org/10.3390/s24092740.

[7] “Performance,” onnxruntime, 2025. https://onnxruntime.ai/docs/performance/

[8] J. Nielsen, “Response Time Limits: Article by Jakob Nielsen,” Nielsen Norman Group, Jan. 01, 1993. https://www.nngroup.com/articles/response-times-3-important-limits/

[9] “Apache Kafka® Performance, Latency, Throughout, and Test Results,” Confluent. https://developer.confluent.io/learn/kafka-performance/

[10] J. Nielsen, “Response Time Limits: Article by Jakob Nielsen,” Nielsen Norman Group, Jan. 01, 1993. https://www.nngroup.com/articles/response-times-3-important-limits/

[11] R. Jota, A. Ng, P. Dietz, and D. Wigdor, “How Fast is Fast Enough? A Study of the Effects of Latency in Direct-Touch Pointing Tasks.” Available: https://www.tactuallabs.com/papers/howFastIsFastEnoughCHI13.pdf

[12] “Tune Mobile Performance (ORT <1.10 only),” onnxruntime, 2025. https://onnxruntime.ai/docs/performance/mobile-performance-tuning.html (accessed Oct. 17, 2025).

[13] NVIDIA, “NVIDIA TensorRT,” NVIDIA Developer, Jul. 18, 2019. https://developer.nvidia.com/tensorrt

[14] Elfi Fertl et al., “End-to-End Ultrasonic Hand Gesture Recognition,” Sensors, vol. 24, no. 9, pp. 2740–2740, Apr. 2024, doi: https://doi.org/10.3390/s24092740.

[15] K. Bimbraw, A. Talele, and H. K. Zhang, “Hand Gesture Classification Based on Forearm Ultrasound Video Snippets Using 3D Convolutional Neural Networks,” arXiv.org, 2024. https://arxiv.org/abs/2409.1643 1 (accessed Oct. 17, 2025).

[16] X. Yang, J. Yan, Y. Fang, D. Zhou, and H. Liu, “Simultaneous Prediction of Wrist/Hand Motion via Wearable Ultrasound Sensing,” [PDF], Research Portal, University of Portsmouth, 9 pp.

[17] J. Yan et al., “A lightweight ultrasound probe for wearable human–machine interfaces,” IEEE Sensors J., vol. 19, no. 14, pp. 5895–5903, Jul. 2019, doi: 10.1109/JSEN.2019.2905243.

[18] “Apache Kafka® Performance, Latency, Throughout, and Test Results,” Confluent. https://developer.confluent.io/learn/kafka-performance/

[19] A. Povzner, “Tail Latency at Scale with Apache Kafka,” Confluent, Feb. 25, 2020. https://www.confluent.io/blog/configure-kafka-to-minimize-latency/ (accessed Oct. 17, 2025).

[20] P. Lara-Benítez, M. Carranza-García, J. García-Gutiérrez, and J. C. Riquelme, “Asynchronous dual-pipeline deep learning framework for online data stream classification,” vol. 27, no. 2, pp. 101–119, Jan. 2020, doi: https://doi.org/10.3233/ica-200617.

[21] “Welcome To Zscaler Directory Authentication,” Nature.com, 2025. https://www.nature.com/articles/s41598-025-18344-9?error=cookies_not_supported&code=cd3d30be-5ed9-4e57-9c16-f25cc4cc4602 (accessed Oct. 17, 2025).

[22]“ResearchGate | Share and discover research,” ResearchGate, 2019. http://researchgate.net

[23] B. G. Sgambato, M. H. Hasbani, D. Y. Barsakcioglu, J. Ibanez, A. Jakob, M. Fournelle, M.-X. Tang, and D. Farina, “High Performance Wearable Ultrasound as a Human-Machine Interface for Wrist and Hand Kinematic Tracking,” *IEEE Transactions on Biomedical Engineering*, vol. 71, no. 2, pp. 484–493, Feb. 2024, doi: 10.1109/TBME.2023.3307952.

[24] Y. Huang, X. Yang, Y. Li, D. Zhou, K. He, and H. Liu, “Ultrasound-Based Sensing Models for Finger Motion Classification,” IEEE Journal of Biomedical and Health Informatics, vol. 22, no. 5, pp. 1395–1405, Sep. 2018, doi: 10.1109/JBHI.2017.2766249.

[25] S. Sikdar, H. Rangwala, E. B. Eastlake, I. A. Hunt, A. J. Nelson, J. Devanathan, A. Shin, and J. J. Pancrazio, “Novel Method for Predicting Dexterous Individual Finger Movements by Imaging Muscle Activity Using a Wearable Ultrasonic System,” *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 22, no. 1, pp. 69–76, Jan. 2014, doi: 10.1109/TNSRE.2013.2274657.

[26] Y. Zhou, J. Zeng, K. Li, Y. Fang, and H. Liu, “Voluntary and FES-Induced Finger Movement Estimation Using Muscle Deformation Features,” *IEEE Transactions on Industrial Electronics*, vol. 67, no. 5, pp. 4002–4012, May 2020, doi: 10.1109/TIE.2019.2920595.

[27] X. Yang, J. Yan, Y. Fang, D. Zhou, and H. Liu, “Simultaneous Prediction of Wrist/Hand Motion via Wearable Ultrasound Sensing,” *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 28, no. 4, pp. 970–977, Apr. 2020, doi: 10.1109/TNSRE.2020.2977908.

[28] Z. Lu, S. Cai, B. Chen, Z. Liu, L. Guo, and L. Yao, “Wearable Real-Time Gesture Recognition Scheme Based on A-Mode Ultrasound,” *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 30, pp. 2623–2629, 2022, doi: 10.1109/TNSRE.2022.3205026.

[29] S. Pithadia and R. Prakash, “Time Gain Control (Compensation) in Ultrasound Applications,” Texas Instruments, Application Report SLAA724, Dec. 2016. [Online]. Available: https://www.ti.com/lit/an/slaa724/slaa724.pdf (accessed Oct. 17, 2025)

[30] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 4th ed. London, U.K.: Pearson, 2018. [Online]. Sample chapters available: https://www.imageprocessingplace.com/…/dip4e_sample_chapters.pdf (accessed Oct. 17, 2025

[31] I. T. Jolliffe and J. Cadima, “Principal component analysis: a review and recent developments,” *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 374, no. 2065, p. 20150202, Apr. 2016. [Online]. Available: https://pmc.ncbi.nlm.nih.gov/articles/PMC4792409/ (accessed Oct. 17, 2025)

[31] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd ed. New York, NY, USA: Springer, 2009. [Online]. Available (mirror): https://esl.hohoweiya.xyz/book/The%20Elements%20of%20Statistical%20Learning.pdf (accessed Oct. 17, 2025)

[32] A. Amidi and S. Amidi, “CS 230 — Convolutional Neural Networks Cheatsheet,” 2018–2023. [Online]. Available: https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks (accessed Oct. 17, 2025).

[33] J. Kreps, N. Narkhede, and J. Rao, *“Kafka: A Distributed Messaging System for Log Processing,”* in *Proceedings of the NetDB Workshop*, 2011.

[34] N. Marz and J. Warren, *Big Data: Principles and Best Practices of Scalable Realtime Data Systems*, Manning Publications, 2015.

[35] GeeksforGeeks, “Difference between Batch Processing and Stream Processing,” GeeksforGeeks, Oct. 26, 2020. https://www.geeksforgeeks.org/operating-systems/difference-between-batch-processing-and-stream-processing/

[36] GeeksforGeeks, “Kappa Architecture System Design,” GeeksforGeeks, Sep. 09, 2024. https://www.geeksforgeeks.org/system-design/kappa-architecture-system-design/#what-is-kappa-architecture

[37] “Kappa Architecture,” Hazelcast, Jan. 29, 2025. https://hazelcast.com/foundations/software-architecture/kappa-architecture/

[38] M. Kleppmann, *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*, O’Reilly Media, 2017.

[39] T. Akidau et al., *“The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing,”* *Proceedings of the VLDB

[40] Government of Canada, “Tri-Council Policy Statement: Ethical Conduct for Research Involving Humans – TCPS 2 (2022),” ethics.gc.ca, 2022. https://ethics.gc.ca/eng/policy-politique_tcps2-eptc2_2022.html

[41] Office of Research Ethics - Carleton University,” Carleton.ca, 2019. https://carleton.ca/researchethics/

[42] American Institute of Ultrasound in Medicine, “As Low As Reasonably Achievable (ALARA) Principle,” AIUM Official Statements, 2023. [Online]. Available: https://www.aium.org/resources/official-statements/view/as-low-as-reasonably-achievable-%28alara%29-principle

[43] American Institute of Ultrasound in Medicine, “Prudent Clinical Use and Safety of Diagnostic Ultrasound,” AIUM Official Statements, 2023. [Online]. Available: https://www.aium.org/resources/official-statements/view/prudent-clinical-use-and-safety-of-diagnostic-ultrasound

[44] K. A. McKay and D. A. Cooper, “Guidelines for the selection, configuration, and use of Transport Layer Security (TLS) implementations,” Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations , Aug. 2019, doi: https://doi.org/10.6028/nist.sp.800-52r2.

Appendices

A - Tables

Table A.1 - Detailed Task Table

ID	Milestone	Subtasks / Description	Acceptance
1.1	Kick-off & Scope Alignment	brainstorm project ideas; meet with supervisor; review papers; record notes	Initial alignment and project definition phase — Kickoff summary document with agreed project scope
1.2	Communication & Meetings	schedule recurring meetings; track feedback; update logs	Regular team and supervisor coordination — Updated project logbook and meeting minutes
1.3	Repository & Tool Setup	create monorepo; configure GitHub; define coding conventions; setup issue tracker	Establish development environment and repository — Functioning repository with documented structure
2.1	Hardware Familiarization	connect ultrasound hardware; verify signal; note parameters	Test and validate US-Builder hardware setup — Device verified functional with initial ultrasound stream
2.2	Communication Setup	implement serial/USB link; send control commands; test data transfer	Establish communication between hardware and software — Successful streaming of ultrasound frames to acquisition app
2.3	Data Buffering & Timestamping	implement ring buffer; manage frame timestamps; store temporary clips	Enable reliable real-time data buffering and storage — Consistent timestamped stream without data loss
2.4	Real-time Validation	run streaming tests; measure acquisition latency; record benchmark logs	Verify latency performance of acquisition system — Validated <5 ms latency from hardware to system memory
3.1	Data Cleaning & Formatting	implement denoising; normalization; format to tensors	Prepare and preprocess ultrasound data for ML — Verified clean dataset ready for ML input

3.2	Pipeline Architecture	implement Kafka producer–consumer links; test data flow between services	Build distributed pipeline between acquisition and ML — Pipeline transmits data to inference service reliably
3.3	Cloud Storage Integration	set up remote storage; implement uploads for retraining data	Store and retrieve collected datasets for retraining — Verified cloud upload and retrieval process
4.1	Baseline Model Training	preprocess collected data; train initial model; log metrics	Establish baseline ML model for inference testing — Baseline model trained with reproducible results
4.2	Inference Integration	connect model to Kafka; stream predictions to UI	Deploy trained model for real-time inference — Real-time inference pipeline functioning end-to-end
4.3	Latency Optimization	profile inference time; parallelize GPU ops; tune batch size	Improve end-to-end inference efficiency — Reduced total pipeline latency (<120 ms target)
4.4	Generalization Testing	test model across subjects; collect new validation data	Evaluate model's cross-subject performance — Validated model robustness with >90% baseline accuracy
5.1	Latency Measurement	time acquisition → prediction; log statistics	Measure and benchmark total pipeline latency — Latency report (<150 ms end-to-end)
5.2	Accuracy & Reliability Tests	compare model output vs ground truth; record confusion matrix	Evaluate prediction accuracy and model consistency — Accuracy >85% across sessions
5.3	Usability Evaluation	run user trials; collect subjective ratings	Assess user experience with control interface — Usability report with ≥80% satisfaction score
5.4	Refinement Iteration	fix issues; retune parameters; final validation	Implement feedback and finalize stable release — Improved model + final system stability report
6.1	Proposal Submission	draft methods; initial results; feedback from advisors	Prepare and submit initial proposal deliverable — Submitted proposal
6.2	Progress Report	document results; figures; lessons learned	Mid-year report summarizing progress — Submitted progress report
6.3	Oral Presentation	prepare slides; rehearse; integrate demo clips	Presentation of project and progress — Successful presentation delivery
6.4	Final Deliverables	finalize poster; record video; compile final report	Complete and submit final documentation package — Submitted final package