Brain–Computer Interface Benchmark Project Plan

Person 1 Person 2

September 23, 2025

Introduction

The goal of this project is to design and implement a reproducible benchmarking pipeline for key signal-processing kernels used in brain–computer interface (BCI) systems. Unlike prior work that focuses primarily on classification accuracy, this pipeline standardizes measurement of latency, jitter, throughput, memory footprint and energy consumption under realistic real-time deadlines. The harness will allow kernels (implemented as shared libraries) to plug into a controlled environment with fixed input/output shapes and identical background loads, exposing trade-offs between computation cost and signal quality. The tasks below are informed by the project proposal and augmented with additional considerations drawn from related BCI benchmarking literature.

Division of Responsibilities

Person 1 will focus on the system harness, scheduling, telemetry infrastructure and overall integration. This includes the dataset replayer, real-time scheduler, plugin interface definition, config and telemetry formats, and energy/latency measurement. Person 1 will also implement the base set of kernels once the mathematical specification and reference implementations are provided.

Person 2 will take charge of selecting and preparing the dataset, writing formal mathematical specifications for each kernel, generating reference (oracle) outputs using trusted scientific libraries, and producing gold standard test bundles. Person 2 also writes documentation describing the dataset, channel mappings and kernel mathematics, and collaborates with Person 1 to validate correctness.

Phased Timeline

The project is divided into four phases across nine weeks. Each phase has specific deliverables and milestones. Tasks are assigned to the two participants based on the division above rather than by name.

Weeks 1–2: Specification and Setup

- **Dataset Selection and Documentation (Person 2).** Choose a public EEG dataset with approximately 64 channels and at least 160 Hz sampling. Document the selected dataset in a `DATASET.md` including subject identifiers, sessions used, channel order, units, reference scheme and any preprocessing. Fix the key parameters: sampling rate ($F_s =$ 160 Hz), window length ($W = 160$ samples), hop ($H = 80$ samples) and number of channels ($C = 64$). Freeze these values by the end of Week 2 to ensure reproducibility.

- **Kernel Specifications (Person 2).** For each kernel in the project scope—common average reference (CAR), notch IIR (biquad) at 50/60 Hz, band-pass FIR (8–30 Hz), Goertzel bandpower and optional Welch PSD—write a formal mathematical specification. Each spec should include:
  - Signal model and input/output domain: $x[t, c]$ defined over windows of length W with C channels, units (µV) and assumed sampling rate.
  - Exact equations (e.g., difference equations for IIR, impulse response for FIR) and design parameters (f_0, Q, number of taps, window type, passband/stopband tolerances, group delay). Boundary conditions and initial state should be explicit.
  - Parameterization: fixed F_s, W, H, C and kernel parameters.
  - Numerical format: specify float32 as default and plan for quantized formats (Q15 and Q7), with tolerances for comparing quantized outputs to float32 references.
  - Edge-case handling: NaNs, missing channels, zero padding and persistent state across windows.
  - Oracle definition: provide a reference Python function using SciPy/MNE that implements the kernel exactly, along with numerical tolerances (absolute and relative) for correctness checks.
  - Acceptance criteria: maximum absolute and relative error thresholds, plus group delay considerations.

  The specifications should be compiled in a `KERNELS.md` file and delivered by the end of Week 2.

- **Reference Data and Gold Bundle (Person 2).** Using the selected dataset and SciPy/MNE oracles, produce a small gold bundle of input windows and corresponding outputs for each kernel. This bundle is used to verify correctness before timing. Include at least three 1–2 second segments with known ground truth.

- **Plugin ABI Definition (Person 1).** Design and document a minimal C ABI for kernel plugins. Each plugin exports an `init()` function that receives a configuration structure (with F_s, W, H, C, data type and kernel-specific parameters), a `process()` function that processes a window of shape $[W \times C]$ into the kernel's output, and a `teardown()` function. A metadata structure should describe supported data types, input/output shapes, internal state size and any quantization support. This ABI is written in `PLUGIN_INTERFACE.md` and frozen by Week 2.

- **Run Configuration and Telemetry Schema (Person 1).** Define YAML schema for run configurations (dataset file, duration, sampling rate, window size, hop size, number of channels, CPU affinity, scheduling policy, background load profile). Define a CSV/JSON schema for telemetry: per-window latency, jitter (p95–p50 and p99–p50), deadline misses, memory footprint, live state bytes, energy per window (E_{window}), derived power ($P = E_{\text{window}} F_s / H$), and analytical counters for FLOPs and bytes per window (arithmetic intensity). Record this in `RUN_CONFIG.md` and `TELEMETRY.md`.

- **Dataset Replayer and Scheduler Skeleton (Person 1).** Implement a basic replayer that streams data from the selected dataset at the true sampling rate, optionally injecting controlled gaps or dropouts to measure kernel robustness. Build a scheduler that forms overlapping windows of length W with hop H, assigns each window a release time and a deadline (H/F_s), and enforces pinned thread execution with optional `SCHED_FIFO`. Leave hooks for injecting background load via stress-ng (idle, medium, heavy) and for warming up the kernels before measurement.

Weeks 3–4: Harness Skeleton and First Kernels

- **Harness Implementation (Person 1).** Build out the harness: integrate the dataset replayer and scheduler, implement the plugin loader, parse the run configuration and manage per-window buffers. Implement high-resolution timing using monotonic clocks, record latency and release/deadline times, and log telemetry to CSV. Add the ability to pin threads and select scheduling policies.

- **CAR Plugin (Person 1).** Implement the common average reference (CAR) plugin according to the specification. Use Person 2's reference code to verify correctness on the gold bundle. Measure per-window latency, jitter, memory and energy. Record results in the telemetry format.

- **Notch IIR Plugin (Person 1).** Implement the biquad notch filter (50/60 Hz) using the provided difference equation. Verify against the reference implementation, including initialization and state persistence across windows. Add support for float32. Prepare for quantized versions but defer implementation until Week 5.

- **Background Load and Warm-Up (Person 1).** Integrate stress-ng to generate idle, medium and heavy background load profiles. Ensure that each kernel run includes a warm-up period where data is processed without measurement to let caches and branch predictors stabilize. Log and compare results with and without background load.

- **Documentation (Both).** Write a usage guide for the harness and plugins, including example configuration files and instructions to run the CAR and notch kernels.

Weeks 5–6: Kernel Expansion and Quantization

- **Band-Pass FIR Plugin (Person 1).** Implement an 8–30 Hz linear-phase FIR band-pass filter with a Hamming window, fixed order and passband/stopband tolerances. Verify against Person 2's reference. Document group delay and adjust the harness if necessary.

- **Goertzel Bandpower Plugin (Person 1).** Implement a Goertzel algorithm to compute bandpower in specified frequency bins. Define the bins (e.g., alpha/beta bands) and verify against reference code. Include the FLOPs/byte analysis for this kernel and record energy consumption.
- **Quantization Support (Both).** Add support for Q15 and Q7 fixed-point formats. Define conversion routines and adjust the plugin ABI to advertise supported types. For each kernel, implement quantized versions and compare outputs to float32 references with predefined tolerances. Measure latency, energy and accuracy loss across data types.
- **Energy Measurement (Person 1).** Integrate Intel RAPL interfaces to read energy counters before and after each `process()` call. Compute energy per window and derive instantaneous compute power ($P = E_{\text{window}} F_s / H$). Validate the measurement by running microbenchmarks and comparing against theoretical expectations. Document limitations (e.g., sampling resolution and shared power domains).
- **Analytic Counters (Person 1).** Estimate the number of floating-point operations and memory bytes read per window for each kernel. Record these metrics in the telemetry and compute arithmetic intensity (FLOPs/byte). This helps identify which kernels may benefit from hardware acceleration.

Week 7: Experiments and Midterm Demo

- **Run Matrix (Both).** Execute a comprehensive set of experiments combining kernels (CAR, notch, band-pass FIR, Goertzel), data types (float32, Q15, Q7) and background loads (idle, medium, heavy). Use the run configuration system to reproduce each experiment. For each run, collect CSV telemetry and generate plots showing latency distributions (p50, p95, p99), jitter, deadline misses, energy per window, power consumption and arithmetic intensity. Perform multiple repeats to quantify variance.
- **Draft Figures and Tables (Person 1).** Produce preliminary comparison figures—e.g., bar charts of latency and energy across kernels; Pareto plots showing latency versus energy; tables summarizing memory footprint and FLOPs/byte. Identify outliers and trends. Highlight trade-offs such as energy savings from quantization versus increased error.
- **Optional Welch PSD (Person 1).** If time permits, begin implementing the Welch power spectral density estimator. Define the number of segments, overlap and window function. Verify correctness with SciPy's PSD implementation.
- **Midterm Demo (Both).** Prepare a live demonstration showing the harness running multiple kernels, generating telemetry in real time and producing comparison plots. Explain the methodology, report preliminary results and solicit feedback on evaluation order and thresholds.

Weeks 8–9: Polish and Final Deliverables

- **Finalize Kernels (Person 1).** Complete any remaining kernels (including optional Welch PSD if pursued) and ensure all implementations pass the reference tests. Fix bugs and optimize where possible without breaking the ABI.
- **Reproducibility and Packaging (Person 1).** Assemble a repository with build scripts, example configurations, a manifest listing all dataset splits and kernel versions, and a one command reproduction script. Ensure that results can be replicated on any x86/Linux machine with RAPL access.
- **Complete Documentation (Both).** Finish the project report detailing the dataset, kernel specifications, harness architecture, measurement methodology, results and lessons learned. Provide clear instructions for adding new kernels and interpreting the metrics.
- **Final Experiments (Both).** Run the final matrix of experiments on the fixed dataset splits and selected kernels (two to four kernels recommended). Generate final comparison plots and summary tables. Include metrics such as p50/p95/ p99 latency, jitter, deadline miss rate, energy per window, power consumption, FLOPs/byte and memory footprint. Make at least one recommendation about which kernels are worth further optimization or acceleration.
- **Presentation (Both).** Prepare presentation slides for the final demo. Summarize the motivation, methodology, key results and recommendations. Rehearse to ensure the demo fits within the allotted time.

Risk Management

- **Dataset mismatch.** Freeze F_s, W, H and C early and verify the selected dataset meets these requirements. Run a probe script to confirm channel count and sampling rate.
- **Specification drift.** Freeze kernel specifications and the plugin ABI by Week 2. Any changes thereafter require a change log with justification and impacts.
- **Measurement variance.** Use pinned threads and optional real-time policies; separate warm-up from measurement; run each configuration multiple times; record per-window telemetry instead of averages. Validate RAPL energy readings with microbenchmarks.
- **Workload imbalance.** If one participant falls behind, prioritize core kernels (CAR, notch, FIR) and defer optional ones (Goertzel, Welch). Clearly document any reduction in scope.

Summary Timeline

Weeks	Owner(s)	Key Deliverables and Milestones
1–2	Persons 1–2	Dataset selected and documented; kernel specs written; plugin ABI, run config and telemetry schema defined; replayer and scheduler skeleton drafted; gold bundles produced.
3–4	Person 1 (with input from Person 2)	Harness skeleton implemented; CAR and notch plugins written and verified; background load and warm-up support added; initial end-to-end runs and documentation.
5–6	Person 1	Band-pass FIR and Goertzel plugins implemented; quantization support added; energy measurement and FLOPs/byte counters integrated.
7	Persons 1–2	Full experiment matrix executed; draft plots and tables produced; midterm demo presented; optional Welch PSD started.
8–9	Persons 1–2	Kernels finalized; reproduction scripts packaged; final experiments run; report and presentation prepared; optional Welch PSD completed if feasible.