Benchmarking BCI Kernels: A Pipeline for Real-Time Performance Analysis

Team Members: Avi Kumar, Weston Voglesonger *Date: September 9, 2025*

1. Overview and Objectives

This project designs a reproducible pipeline for analyzing kernels that form the building blocks of brain–computer interface (BCI) workflows. We standardize measurement of latency, throughput, jitter, memory footprint, and energy under fixed input/output contracts so results are comparable across kernels and machines. Candidate kernels include FIR/IIR band-pass filters, spatial references (common average, Laplacian), and frequency-domain methods (e.g., Welch's power spectral density). The work is *PC-only*: we target commodity x86/Linux so the pipeline is broadly reproducible and we report energy as incremental joules per window (via RAPL) and derived compute power (milliwatts)—a proxy for on-implant feasibility near the ∼40 mW class budget discussed in the BCI architecture literature [1, 7]. In short, we complement accuracy-focused tools (e.g., MOABB [2]) with a compact, system-level harness that exposes timing and efficiency trade-offs under realistic deadlines.

2. Pipeline

A dataset *replayer* streams EEG at the true sampling rate F_s. A *scheduler* slices the stream into fixed windows (length W, hop H) and assigns each window a release time and a real-time deadline (H/F_s). Kernels are drop-in plugins (shared libraries) behind a tiny C ABI (init/process/teardown + JSON metadata); they compute on each window while the harness controls timing, fairness, and I/O. The harness pins the worker thread, sets optional real-time policy, invokes kernels, and records per-window telemetry—latency, throughput, p50/p95/p99 jitter, memory footprint, and *energy* (RAPL on x86). Runs are driven by YAML configs and emit reproducible CSV tables and plots; lightweight reference tests compare kernel outputs against SciPy/MNE oracles to verify correctness before timing. This separation lets us rank kernels objectively and identify which ones are worth accelerating and pushing closer to the brain.

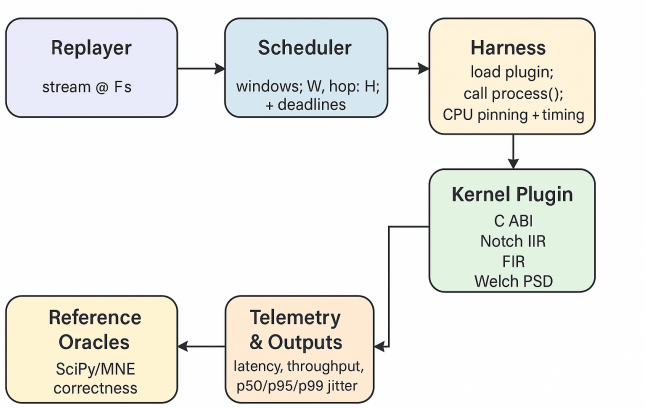

Figure 1: PC-only benchmarking pipeline. The harness controls timing and fairness; kernels are plugins behind a small C ABI; telemetry is captured per window and exported as CSV/plots.

3. Methods (What We Will Build)

3.1. Dataset Replayer & Scheduler

We use public EEG datasets (e.g., PhysioNet; MOABB loaders) or synthetic traces. The replayer maintains a stable timebase and can inject controlled gaps/dropouts for sensitivity checks. The scheduler forms overlapping windows (default F_s=160 Hz, W=160, H=80, C=64 channels) and stamps each window with release and deadline times. We adopt pinned threads and optional `SCHED_FIFO` to reduce OS-induced jitter.

3.2. Kernel Plugin ABI

Each kernel is a shared library exporting: `init(config)`, `process(in,out)`, `teardown()`, plus JSON metadata (I/O shape, state size, supported dtypes). We will implement *CAR*, *notch IIR* (biquad), *band-pass FIR* (8–30 Hz), and *bandpower via Goertzel*; *Welch PSD* is optional. Reference outputs are generated via SciPy; tolerances are set per-kernel.

3.3. Harness & Telemetry

For each window, the harness calls `process()` and records timestamps using a monotonic clock. Latency is measured from last-input-sample arrival to output-ready; jitter is defined as $p95-p50$ and $p99-p50$. Deadline misses are logged

rather than hidden. Memory footprint and live state are recorded from kernel metadata. Energy is sampled around each `process()` via Linux RAPL `energy_uj`, producing E_{window} (joules) and derived continuous compute power $P = E_{\text{window}} \cdot F_s/H$ (mW).

3.4. Fairness, Load Profiles, and Quantization

All kernels run over the same windows with standardized warm-up and identical background-load profiles (idle / medium / heavy) induced via `stress-ng`. We enforce CPU affinity and avoid per-window heap allocations inside kernels. The harness supports quantization sweeps (float32, Q15, Q7) to measure accuracy impact and potential fixed-point savings. We additionally log rough FLOPs and bytes-per-window (analytical counters) to estimate arithmetic intensity for accelerator targeting.

4. State of the Art and Challenges

Prior work emphasizes accuracy and dataset reproducibility (e.g., MOABB [2]) or data standards (EEG-BIDS/iEEG-BIDS). BCIBench [3] profiled kernels on simulated low-power x86 but was not updated or made reusable. Architecture-facing surveys [1, 7] argue for real-time constraints and better systems evidence. There remains no small, open pipeline that reports latency, power-usage, and jitter in a reproducible, apples-to-apples manner for common BCI kernels. Our contribution is a compact PC-only pipeline that complements existing accuracy frameworks with practical system-level measurements and clear, per-kernel guidance for push-down (on-head/implant) decisions.

5. Significance and Impact

For researchers, the benchmark quickly answers whether kernels meet real-time budgets under realistic loads. For architects, it provides empirical data to prioritize which kernels deserve hardware acceleration [1, 7]. For our team, this is a realistic entry into BCI systems with a path to scale. The expected outcome is a defensible list of kernels whose timing/energy profiles justify early on-device placement, plus normalized metrics (J/window, mW, FLOPs/byte) that translate into accelerator or fixed-point design targets.

6. Executional Challenges and Risk Mitigation

Challenges include limited prior experience with BCI-specific kernels and obtaining stable energy/timing measurements on shared machines. We mitigate risk by fixing configurations (dataset, F_s, W, H, C) early; pinning threads and using RT policy; repeating runs to bound variance; validating kernels against SciPy/MNE before timing; and separating warm-up from measurement. We also version config manifests so results remain reproducible across environments.

7. Resources

One x86_64 laptop/desktop running Linux with user-space access to RAPL counters; Python 3 with MNE, NumPy, SciPy, pandas, matplotlib; build tools for C/C++ (and optional Rust). The repository will include minimal build scripts, example configs, and a manifest enabling one-command reproduction.

8. Timeline

Weeks 1–2: harness skeleton (replayer, scheduler, CSV/plots) and CAR plugin with reference tests. Weeks 3–4: notch IIR and FIR band-pass plugins; add warm-up and background-load profiles. Weeks 5–6: Goertzel bandpower; RAPL energy logging; FLOPs/bytes counters; quantization sweeps. Week 7: run matrix and draft tables/figures; midterm demo. Weeks 8–10: (optional) Welch PSD; finalize report and "push-down" recommendations.

9. Metrics and Deliverables

Midterm: end-to-end pipeline on two kernels and one dataset; CSV exports and at least one comparison figure. Final: benchmarks for two to four kernels on fixed dataset splits; comparison plots; reproducible code/data manifest. We report p50/p95/p99 latency, jitter (tail-minus-median), deadline-miss rate, E_{window}, mW, and structural metrics

(FLOPs, bytes, arithmetic intensity). The final report summarizes bottlenecks/trade-offs and provides at least one actionable insight about real-time viability and on-device prioritization.

10. Open Questions

Which kernels are most representative of invasive BCIs, what accuracy tolerances are acceptable under quantization, and which derived metrics (e.g., FLOPs/byte vs. data-reduction ratio) most usefully predict on-device payoff? Feedback on these will refine our evaluation order and thresholds.

References

[1] A. Bhattacharjee, "Computer Architecture for Brain-Computer Interfaces," *SIGARCH*, Sept. 26, 2019.

[2] S. Chevallier, et al., "The Largest EEG-Based BCI Reproducibility Study for Open Science: The MOABB Benchmark," *arXiv*, Apr. 3, 2024, doi:10.48550/arXiv.2404.15319.

[3] R. Jafari, O. Dehzangi, C. Zong, V. Nathan, "BCIBench: A Benchmarking Suite for EEG-Based Brain-Computer Interface," *ODES@CGO 2014*, pp. 19–24.

[4] N. D. Skomrock, et al., "A Characterization of Brain-Computer Interface Performance Trade-Offs Using SVMs and DNNs to Decode Movement Intent," *Frontiers in Neuroscience*, vol. 12, p. 763, 2018.

[5] F. Manzouri, et al., "A Comparison of Machine Learning Classifiers for Energy-Efficient Implementation of Seizure Detection," *Frontiers in Systems Neuroscience*, vol. 12, p. 43, 2018.

[6] A. Bhattacharjee, et al., "Dataflow-Specific Algorithms for Resource-Constrained Scheduling and Memory Design," in *Proceedings of the 37th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '25)*, ACM, 2025, doi:10.1145/3694906.3743342.

[7] R. Pothukuchi and A. Bhattacharjee, "The Brain-Computer Interfacing Landscape for Computer Architects," *SIGARCH*, Jan. 22, 2024.