# Spring 2026 Project Proposal for CPSC 4900 - Senior Project: Optimized Translation of Matlab to RISC-V Architecture For Brain Computer Interfaces

Kenan Erol

Advisor: Professor Rajit Manohar

## Introduction

Brain computer interfaces (BCIs) represent a transformative technology that can revolutionize the treatment of neurological disorders, such as epilepsy and strokes, by enabling direct communication between the brain and external devices. The HALO (Hardware Architecture for LOw Power BCIs) project is developing implantable BCI chips that can perform complex real-time signal processing on-device while operating on extremely low power to maximize device longevity and minimize thermal effects on brain tissue [1]. HALO achieves these goals through a RISC-V microcontroller architecture that uses the RISC-V Vector extension (RVV) for power efficient parallel computation [1, 2].

However, a significant gap exists between the tools used by neuroscience researchers and the requirements of the BCI hardware. Neuroscientists predominantly use MATLAB to develop signal processing algorithms. HALO requires highly optimized and vectorized C code that uses RVV for efficient execution. Currently, no automated pipeline exists to translate MATLAB code into RVV-optimized C code suitable for deployment on RISC-V BCI hardware.

This project proposes to complete and integrate two existing partial solutions - a MATLAB to C translator (ts-traversal) and a C matrix library with RVV vectorization (RISCV-Matrix) - into a unified, end-to-end pipeline. The resulting system will enable neuroscientists to write algorithms in familiar MATLAB syntax that will automatically generate power-efficient code for the BCIs and bridge the aforementioned gap.

## Background

### RVV Relevance

RVV is critical for meeting HALO’s dual constraints on low power consumption and real time signal processing. RVV provides assembly instructions that perform operations on entire vectors in parallel rather than iterating through each element individually. As described in the RISCV-Matrix documentation, “rather than adding two vectors by iterating through each element of their elements, RVV is capable of loading in entire vectors (or, at least, large segments of them), and executing the operation on the entire vector at once.” This approach reduces the number of instruction fetches, loop iterations, and memory accesses required, overall reducing the energy consumption. By completing matrix computations faster (vectorized versions 5-6% faster according to RISCV-Matrix benchmarks), the processor can return to low power idle states sooner, extending the battery life for implanted BCIs.

Existing Translation Infrastructure

Two partial solutions exist within the research group that form the foundation for this project. Ts-traversal is a typescript-based source-to-source compiler developed by Danielle Gruber ‘25 that converts Matlab code to C. The system has three phases. The parse phase uses tree-sitter and tree-sitter-matlab to generate a concrete syntax tree (CST) from Matlab source code. The type inference phase infer variable types from the dynamically typed Matlab to statically typed C. Finally, the code generation phase traverses the CST and transforms each Matlab construct into equivalent C code. RISCV-Matrix is a C library implementing Matlab equivalent matrix operations with RVV vectorization, developed by Zach Taylor ‘23 and Xiayuan Wen. There is also a Matlab commercial tool called MATLAB CODER that can generate C and C++ code, but these do not have architecture-specific optimizations [4]. To our knowledge, no existing tool provides automated MATLAB-to-RVV translation for embedded signal processing applications.

## Problem Description

The deployment of signal processing algorithms on RISC-V BCIs faces a critical toolchain gap. Neuroscientists develop and validate algorithms in MATLAB, leveraging its intuitive syntax for matrix operations and extensive signal processing toolboxes. However, the HALO BCI platform requires highly optimized C code utilizing RISC-V Vector Extension intrinsics to meet power and real-time performance constraints. Currently, manual translation is required, which is a time-consuming, error-prone process that creates barriers between algorithm development and hardware deployment.  The existing partial solutions suffer from several deficiencies:

MATLAB-to-C Translation Gaps (ts-traversal):

- Signal processing functions (stft, hamming, hanning) produce incorrect output or dimension mismatches

- Determinant computation (det) fails on certain matrix inputs
- No support for sparse matrices

<u>C-to-RVV Vectorization Gaps (RISCV-Matrix):</u>

- FFT and signal processing functions execute in scalar mode only, as the underlying FFTW library has not been vectorized for RVV
- Matrix operations including transpose, eigendecomposition, and matrix inverse lack vectorized implementations
- BLAS/LAPACK routines used for linear algebra execute without RVV optimization
- Trigonometric functions lack vectorized implementations due to the absence of native RVV trigonometry instructions
- Statistical functions (mean, standard deviation, variance) remain scalar-only

<u>Integration Gaps:</u>

- No unified build system connecting both translation stages
- No automated testing framework exists to validate end-to-end correctness from MATLAB input to RVV execution

These gaps prevent the pipeline from supporting real BCI signal processing workloads, which typically involve filtering, spectral analysis (FFT), and statistical feature extraction - precisely the functions that are incomplete or unvectorized.

## Planned Approaches

The project will proceed in three phases: assessment, improvement, and integration.

The first phase establishes a baseline understanding of the current system capabilities. A comprehensive test suite will be developed containing BCI-relevant MATLAB programs that exercise filtering, FFT, and statistics. Each test case will include expected outputs derived from MATLAB execution. ts-traversal will be evaluated by running each test through the translator and categorizing failures by root cause. Similarly, RISCV-Matrix will be baselined by comparing vectorized and reference implementations of the Spike RISC-V ISA simulator [3].

The second phase will target and fix the highest impact deficiencies identified in phase 1. For ts-traversal, debugging will focus on the signal processing and matrix function translations that produce incorrect output. For RISCV-Matrix, vectorization efforts will primarily focus on functions by order of their importance to BCI workloads. For example, it would be ideal to vectorize some matrix transpose functions that have not yet been vectorized as transposition will be heavily used.

The focus of the final phase will be to create the cohesive end-to-end system. A unified build system will accept Matlab source files and produce executable RISC-V binaries in a single command. This system will use ts-traversal for translation, then compile the generated C code with the RISCV-Matrix library using the RISC-V GNU compiler toolchain. An automated test harness will execute both the original matlab code and the generated RISC-V binary and compare the outputs to verify correctness.

## Timeline

- Week 3-4: Getting started with the currently existing code and benchmarking
- Week 5-6: Debug failure cases in ts-traversal, such as hamming and matrix determinant
- Week 7-9: Vectorize matrix and signal processing functions that have not been vectorized yet in RISCV-Matrix
- Week 10-12: integrate and test the matlab=>C=>RVV pipeline
- Week 13: End-to-end testing with BCI signal processing workloads and performance benchmarking
- Week 14-16: Final results documentation, final report, final poster, submission

## Deliverables

- Benchmark the capabilities of the two repositories (ts-traversal for matlab to C conversion, RISC-V-Matrix for C to RVV)
- Vectorize the remaining non-vectorized BCI-critical functions for signal processing and matrix operations, such as…
- Debug the limitations of ts-traversal on matrix and signal processing related functions
- Provide a single repository for the integration of the conversion of matlab to RVV
- Final report to discuss the findings, results, and present the full pipeline
- Poster to summarize results
- Upload the final report and repository to the CPSC 4900 website

## References

[1] I. Karageorgos et al., "Hardware-Software Co-Design for Brain-Computer Interfaces," 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA), Valencia, Spain, 2020, pp. 391-404, doi: 10.1109/ISCA45697.2020.00041.

[2] Nelson, Phil. "Introducing HAL Riscv-Rvv: Unleashing the Power of RISC-V CPUs with RVV 1.0." *OpenCV*, 9 July 2025, opencv.org/blog/introducing-hal-riscv-rvv-unleashing-the-power-of-risc-v-cpus-with-rvv-1-0/. Accessed 25 Jan. 2026.

[3] Spike, https://github.com/riscv-software-src/riscv-isa-sim.

[4] MathWorks, "MATLAB Coder: Generate C and C++ code from MATLAB code." [Online]. Available: https://www.mathworks.com/products/matlab-coder.html