sddec25-01

User Testing Plan

1. Identifying Users

While our end product aims to serve handicap individuals with underlying conditions, the full picture of the development of that end product is years away from materializing to start hands-on testing. Thus, we have chosen that we should focus, for our user testing plan, on the advisor and future teams that will interact with the system and codebases in the coming years.

2. Testing Objective

Understandable knowledge transfer from team to team and verifiable information that the client approves of said usage for easy, and verifiable Result and Metrics. After setup of the system, the advisor should just need to run our validation program included into our petalinux build to verify and test our results.

- Ensure knowledge transfer and system reproducibility across development cycles.
- Validate integration accuracy and algorithm performance benchmarks (e.g., ≥200 FPS).
- Simplify system usability for technically skilled but non-author team members.
- Confirm system adaptability and documentation quality for future extension.

3. Testing Methodology

Testing Approach

Passed on repo will need to be accessible on the hardware and build successfully. From there on, detailed instructions of how and what to feed the algorithm to visually understand how it processes and passes information will be essential to future usage.

Selected Testing Methods

- Observation
- Task Analysis
- Prototype Evaluation

4. Test Plan Details

Test Environment

- Location: Coover Senior Lab
- Equipment needed: Kria Board KV260, Computer

Test Scenarios

1. **Scenario 1**: Benchmark Previous Code
    - Tasks: Benchmark accuracy of current algorithm making an easily reproducible result
    - Success criteria: Single script can be ran to receive the results on a setup system
2. **Scenario 2**: Benchmark Split Up Model
    - Tasks: Using the benchmarking scripts made in the previous step ensure the same or better accuracy of the model was achieved.
    - Success criteria: Comparisons against the previous scenario results.
3. **Scenario 3**: Benchmark Previous Code (On board)
    - Tasks: Log time of completion for each segment of current implementation.
    - Success criteria: View the visual logs written to an output file.
4. **Scenario 4**: Benchmark Pipelined Code
    - Tasks: Log time of completion for each segment of proposed implementation.
    - Success criteria: View and compare the visual logs written to another output file.

5. Data Collection

Metrics to Capture

- **Quantitative Metrics**:
    - Time to completely setup the system
    - FPS throughput
    - Accuracy/error rate
- **Qualitative Feedback**:
    - Suggestions for Improvement

Measurement Tools

- Accurate logging for reporting time of completion of each segment in the compared algorithms.
- Benchmarking scripts for performance.

CPU Usage Monitoring Tools

- **Mpstat (sysstat):** Part of the **sysstat** collection of tools, `mpstat` reports per-CPU usage statistics, including user/system time and idle time for each core. The sysstat suite is a collection of performance monitoring tools for Linux (xilinx wiki). Using `mpstat` or the broader `sar` utility, we can measure CPU utilization on each A53 core over time. `sar` can record CPU metrics periodically to log files for later analysis.

- **Xilinx System Debugger (XSDB) and Vitis Profiling (GUI):** For more advanced CPU profiling, Xilinx's Vitis IDE provides profiling tools that can connect to a running PetaLinux system (via JTAG). These allow function-level profiling and timeline analysis by instrumenting code or using hardware Performance Monitoring Unit (PMU) counters. In the Vitis analyzer (formerly SDK System Performance Analysis), we can trace function execution times and CPU usage over time on a timeline.

Memory Usage Monitoring Tools

- **Free and VMStat:** The `free` command (often built into BusyBox) provides a snapshot of total memory, used, free, buffers/caches, and swap. It's useful for a quick check of memory consumption. For more detail, `vmstat` (virtual memory statistics) shows a running summary of processes, memory, swap, I/O, and CPU in columns (linuxjournal monitoring tools). `vmstat` can update repeatedly (e.g., `vmstat 1` for per-second stats) to show how memory usage changes, and includes columns for swap in/out and memory paging which are crucial on memory-constrained systems.included via `procps` or `util-linux`.

Latency and Real-Time Performance Analysis Tools

- **Ftrace (Built-in Kernel Tracer) and Trace-cmd:** For deep analysis of system behavior and latencies, Linux's **ftrace** framework is invaluable. It can trace function calls, interrupts, scheduler events, and more with very fine granularity. Using **ftrace** directly means writing to files in `/sys/kernel/debug/tracing`. Can record scheduling events (`trace-cmd record -e sched_switch`), IRQ events, function execution times, and even generate per-CPU timelines.

- **Perf (Linux Performance Counters): perf** is a Linux profiling tool that can also trace events and measure latency to some extent. It uses hardware PMU counters and software events to profile the system. Perf is part of the kernel source (the `perf` utility can be built for the target). *S*ample CPU performance events (cycles, cache misses, branch misses, etc.), and also record software events like context switches or page faults (redhat per vs gprof). With `perf sched timehist` or `perf sched record`, we can get scheduler latency insights.

Graphical Tools

While many embedded systems rely on command-line tools, there are GUI or network-based monitoring solutions that can be used with PetaLinux for a more user-friendly overview, especially during development or testing:

- **SNMP Tools:** PetaLinux includes **net-snmp** in its package list (xilinx-wiki package groups), meaning the device can run an SNMP daemon. This allows exposing CPU load, memory, interface stats, etc., via SNMP protocol to any standard monitoring system (Nagios, Zabbix, custom scripts, etc.). Industry-standard protocol for monitoring; good if integrating the Zynq device into a larger IT infrastructure monitoring. *Integration:* Add and configure `snmpd` on the device with the desired MIBs.
- **GUI on Target (X11 based):** In cases where the Zynq device has a display (for example, a Zynq MPSoC running PetaLinux with GUI stack), standard Linux GUI system monitors will be used. These provide graphical views of CPU and memory usage and process lists. *Features:* User-friendly visuals, graphs of CPU/memory over time, point-and-click interface. *Integration:* This requires a full OS image with GUI libraries leveraging X11. The tooling is already built into the petalinux configuration for our board.

Tool / Package	Metrics Covered	Interface	Use Case
Dstat (`dool`)	Combined stats: CPU, memory, disk, network, etc. in one view	CLI (interactive streaming)	Real-time multi-resource monitoring during tuning or testing
Perf (Linux perf)	CPU performance counters, event sampling, software events (e.g. context switches)	CLI (commands with text output)	Profiling CPU and code performance; finding bottlenecks, measuring event counts (cache misses, interrupts, etc.)

Ftrace & Trace-cmd	Kernel function traces, scheduler and interrupt latency, event timelines	CLI (`trace-cmd` for capture); GUI (KernelShark viewer)	In-depth analysis of latency and timing at kernel level; debugging real-time performance issues by tracing execution.
Vitis Analyzer (Xilinx)	Function execution times, software trace (via instrumentation or PMU)	GUI (on host PC)	Profiling software on Zynq MPSoC during development; identify hotspots to optimize or offload. (Development-time tool)

Table: Comparison of performance observability tools, metrics, interfaces, and typical use cases on PetaLinux (Zynq UltraScale+).

6. Evaluation Criteria

Success Metrics

- As one develops and theorizes an input it should try to solve a matrix, as the cv::mat class that handles input takes a frame and formats it as that of a matrix. Thus, the user should be able to understand their own matrix and how it is solved. Success will depend on if the matrix is solved correctly, but also if the user themself fed the input correctly.

7. Testing Schedule

Activity	Date	Duration	Responsible Person
Splitting up Model	4/29/2025	1-2 Weeks	Tyler
Multi Threading	5/1/2025	2 months (before and after summer)	Aidan
Benchmarking	4/9/2025	(Previous Team Code to our Code)	Joey and Conner

8. Resources Required

- **Personnel**:

  Joseph Metzen – Kria Board Manager

  Tyler Schaefer – ML Algorithm Analyst

  Conner Ohnesorge– ML Integration HWE

  Aidan Perry – Multithreaded Program Developer

- **Equipment**: Kria Kv260 board

9. Reporting Plan

- Format of results: Google Docs and Repo
- Audience: JR
- Timeline: Two Months before the completion of CPRE492 to enable for future team onboarding.

10. Ethical Considerations

- **Privacy**: JR and members collects all the data
- **Consent**: Advisor and participants will receive a consent form
- **Recording**: Any screen recording will be anonymized
- **Compensation**: voluntary participation