VORTEX Roofline Lab: An Evidence-Aware Architecture, Prototype Evaluation, and Integration Roadmap

Aaron Singh

***Abstract*—This paper presents VORTEX Roofline Lab, a project in a twenty-project AI infrastructure portfolio. The current repository is an active prototype with implemented behavior, automated correctness tests, and explicit boundaries around unverified hardware or production claims. We describe the system model, current implementation, goals, and evaluation method, then propose five integrations that advance the project toward reproducible system-level validation. A local audit executed 8 project tests successfully. No accelerator, deployment, or performance conclusion is inferred unless a corresponding committed benchmark or profiler artifact exists.**

***Index Terms*—VORTEX, AI infrastructure, reproducibility, prototype validation, systems evaluation, hardware–software co-design**

I. Introduction

VORTEX Roofline Lab (P01) addresses a bounded problem within the portfolio's track-01-gpu-kernels track. Its declared status is *Active Prototype* [1]. The project validates its central mechanism before adding device-specific acceleration, production orchestration, or real-world hardware. This ordering matters because optimization without a trusted reference can make incorrect behavior appear successful.

The project goals are to establish deterministic behavior, encode correctness as tests, create machine-readable evidence, identify limiting resources through measurement, and integrate results with adjacent portfolio systems without losing provenance. The repository contains one paper named exactly after its singular folder: `P01-vortex-roofline-lab.tex`.

II. Technical Context

GPU kernel development requires a correct numerical reference before optimization. Reference equivalence, workload shape, warm-up policy, and profiler evidence are therefore first-class design concerns. The portfolio benchmark standard requires environment, workload, method, metric, artifact, reproduction, and limitation fields; unknown values remain explicitly unmeasured [3].

III. System Model and Architecture

The prototype is organized around the following domain model:

$$I = F/B, \quad P_{attainable} = \min(P_{peak}, BW\, I) \tag{1}$$

The equation is a design and test abstraction rather than a claimed empirical law. It supports invariants and expected-value checks while later implementations replace synthetic inputs with representative workloads or devices.

The software architecture contains an input/configuration layer, a deterministic core, validation and evidence output, and a local visualization. The principal inspected source artifacts are `python/compare_python_numpy.py`, `python/run_baseline.py`. Unsupported real-world conditions are surfaced as limitations rather than silently simulated.

IV. Detailed Script Operation and Rationale

The baseline script creates deterministic vectors, computes element-wise addition with a readable Python reference, checks every output, measures repeated execution, and appends environment-aware JSONL evidence. The companion comparison script runs the same workload with Python lists and NumPy so implementation overhead can be studied before an XPU port.

The execution path is:

1) Validate size, repetitions, warm-ups, and deterministic seed.
2) Generate two repeatable input vectors and compute the Python reference output.
3) Check output correctness before accepting any timing sample.
4) Measure repeated runs and summarize latency and throughput.
5) Attach Git/system metadata and append a JSONL record for review.

V. Implemented Prototype

The metadata and source audit found these completed features [2]:

- Beginner-readable Python CPU vector-add reference
- Deterministic synthetic input generation
- Correctness validation and unit tests
- JSONL benchmark evidence writer with Git metadata
- Optional Python-list versus NumPy comparison

`python3 -m unittest discover -s tests -p 'test_*.py'` completed successfully with 8 tests. The inspected test artifacts are `tests/unit/test_python_numpy_comparison.py`, `tests/unit/test_vector_add_reference.py`. This is evidence of local correctness for encoded cases, not production scale or hardware performance.

TABLE I
IMPLEMENTATION ARTIFACTS AND WHY THEY EXIST

Artifact	Observed responsibility	Engineering rationale
python/compare_python_numpy.py	make_inputs, measure, summarize, run_comparison, build_record, main	Implements the inspectable, unit-tested project core.
python/run_baseline.py	vector_add_reference, run_vector_add, validate_settings, output_is_correct, git_value, git_worktree_is_dirty, system_memory_gb, build_record, main	Implements the inspectable, unit-tested project core.
scripts/reproduce.sh	Fixed test and demonstration entry point	Gives another developer one command for local reproduction.
PROJECT.yaml and ANALYSIS.md	Status, completed work, planned work, and claim boundaries	Separates declared intent from evidence-backed implementation.
streamlit_app.py	Local evidence and status visualization	Makes outputs inspectable without upgrading simulation into a hardware claim.

VI. TESTING METHODOLOGY AND OBSERVED RESULTS

Testing uses Python's standard `unittest` discovery and exercises the public behavior of the reference implementation. The audit reran the suite from the project folder with `python3 -m unittest discover -s tests -p 'test_*.py'`. All 8 discovered tests passed. The result establishes correctness only for the encoded local cases; it does not establish accelerator correctness, real-device behavior, production reliability, or benchmark completion.

No numerical performance result is promoted by this test run. Where scripts emit JSON or JSONL, those outputs remain raw or simulation-specific until a reviewed summary includes hardware, software, workload, warm-up, repetition, correctness threshold, Git revision, and limitations.

VII. CLAIM BOUNDARIES AND RISKS

The project records these unverified or excluded claims:

- No XPU speedup or roofline result has been demonstrated yet

The main risk is confusing synthetic or modeled behavior with deployed-system behavior. Other risks include incomplete workloads, platform-dependent timing, missing failure injection, and interfaces not yet exercised across device boundaries. Performance claims require a reviewed record meeting the portfolio standard.

VIII. EVALUATION PLAN

Evaluation proceeds through correctness tests, deterministic reproduction with Git and environment metadata, repeated benchmarks reporting latency/throughput/memory/error metrics, and a named profiler capture tied to exact hardware and source revision. Success requires reference equivalence within a documented tolerance, preservation of safety and resource invariants, clear failures, and evidence reproducible from a clean environment.

IX. GOALS, MILESTONES, AND SUCCESS CRITERIA

The project goals are staged so that correctness precedes performance and integration. A goal is complete only when its proof artifact is committed or otherwise reviewable; prose or a simulated number alone is insufficient.

X. FUTURE WORK AND INTEGRATIONS

The five project-specific next steps are:

1) Intel XPU implementation
2) Profiler capture and roofline analysis
3) Implement and validate equivalent SYCL and PyTorch XPU kernels.
4) Publish a reviewed roofline report linked to a named profiler capture.
5) Feed normalized benchmark summaries into P19 and P20.

The early items complete declared evidence; the later items connect downstream portfolio consumers. Each integration should add tests and a reviewable artifact such as JSONL evidence, a report, profiler capture, deployment manifest, trace, or labeled data set.

XI. CONCLUSION

VORTEX Roofline Lab is an evidence-aware active prototype: its implemented behavior and tests are real, while unbuilt hardware, deployment, and performance goals remain labeled. Completing the five integrations in dependency order will advance it from a learning artifact toward a credible portfolio component.

REFERENCES

[1] Aaron Singh, "VORTEX Roofline Lab PROJECT.yaml," local portfolio repository.
[2] Aaron Singh, "VORTEX Roofline Lab: README, ANALYSIS, source, and test artifacts," local portfolio repository.
[3] Aaron Singh, "AI Infrastructure Portfolio Benchmark Standard," local portfolio repository.

TABLE II
AUDITED TEST MATRIX

Test artifact and case	Behavior being checked	Observed result
tests/unit/test_python_numpy_comparison.py:test_inputs_repeat_with_same_seed	Inputs repeat with same seed.	Pass (local)
tests/unit/test_python_numpy_comparison.py:test_comparison_passes_correctness	Comparison passes correctness.	Pass (local)
tests/unit/test_python_numpy_comparison.py:test_summary_contains_throughput	Summary contains throughput.	Pass (local)
tests/unit/test_vector_add_reference.py:test_vector_add_reference	Vector add reference.	Pass (local)
tests/unit/test_vector_add_reference.py:test_vector_lengths_must_match	Vector lengths must match.	Pass (local)
tests/unit/test_vector_add_reference.py:test_generated_data_is_repeatable	Generated data is repeatable.	Pass (local)
tests/unit/test_vector_add_reference.py:test_output_correctness_check_detects_bad_value	Output correctness check detects bad value.	Pass (local)
tests/unit/test_vector_add_reference.py:test_invalid_benchmark_settings	Invalid benchmark settings.	Pass (local)

TABLE III
DECLARED STATUS VERSUS AUDITED EVIDENCE

Audit field	Finding	Evidence source
Declared status	Active Prototype	PROJECT.yaml
Evidence-backed status	Active local prototype; 8 tests passed	Source plus local unittest run
Accepted measured results	None recorded in measured_results	PROJECT.yaml
Mismatch / claim boundary	No XPU speedup or roofline result has been demonstrated yet	PROJECT.yaml and ANALYSIS.md
Next proof required	Intel XPU implementation; Profiler capture and roofline analysis	Planned features

TABLE IV
PROJECT GOALS AND REQUIRED PROOF

ID	Goal	Completion evidence
G1	Intel XPU implementation	Passing tests and a reviewed source artifact
G2	Profiler capture and roofline analysis	Machine-readable result with reproduction metadata
G3	Publish a reviewed CPU baseline summary	Reference-equivalence or domain-correctness report
G4	Implement reference-equivalent XPU execution	Named profiler, deployment, or integration artifact