Enhancing JAX Support in Awkward Array Library

L. J. Shikhman
Florida Institute of Technology

April 2025

1 Background & Motivation

`awkward-array` provides jagged (variable-length) data structures in NumPy, CuPy, and JAX. The JAX backend is considered stable (guide), but Python testing may miss unexecuted branches so code could be incorrect or missing. Extending test coverage will either uncover issues or confirm correctness, guiding subsequent development.

2 Objectives

1. **Test-Coverage Extension.** Augment the test suite to cover at ≥90% of JAX-related branches and edge cases.
2. **High-Level Method Verification.** Ensure `ak.sum`, `ak.sort`, `ak.combinations`, and similar functions behave correctly under `grad` and `jit`, on CPU and GPU.
3. **Realistic Physics Analysis.** In consultation with Lino, Peter, and other physicists, build a representative HEP pipeline (event selection, combinatorics, custom losses); evaluate JAX-backend correctness, performance penalty or advantage on CPU/GPU, JIT opportunities, and missing functionality.
4. **Environment Validation.** Install and run tests under both Linux and Windows/WSL2 (with GPU), via pip or conda on the master branch, to confirm developer and user workflows.
5. **Open-Ended Investigation.** Document answers to key questions:
   - Does JAX on CPU incur excessive overhead?
   - Where do JIT-compiled functions yield benefit?
   - Which operations remain unsupported or incomplete?

3 Scope

- Audit and extend wrappers to cover all dispatch paths.
- Develop comprehensive tests to target every JAX branch and edge case.
- Implement high-level API tests for core functions (`ak.sum`, `ak.sort`, `ak.combinations`, etc.) under gradient and JIT execution on both CPU and GPU.
- Create an example high-energy physics analysis pipeline for performance and feature evaluation.

- Document test coverage metrics, verify API correctness, include a physics case study, and detail environment setup.
- Benchmark JAX performance by measuring CPU overhead, GPU throughput, and JIT speedups.

4 Plan of Work

1. **Weeks 1–4: API Tests** Develop and verify tests in `tests/jax/high_level/` for `ak.sum`, `ak.sort`, `ak.combinations`, ensuring correct behavior under `grad/jit` on both CPU and GPU.
2. **Weeks 5–8: Physics Scenario** Collaborate with HEP colleagues to define and implement a representative analysis pipeline; collect performance and functionality data, identifying any missing features.
3. **Weeks 9–10: Analysis & Documentation** Analyze JAX CPU overhead, GPU benefits, JIT impacts, and unsupported operations; draft comprehensive findings in `docs/user-guide/jax.md`.
4. **Weeks 11–12: Review & Merge** Finalize tests and documentation; address maintainer feedback; prepare and submit upstream pull request with updated release notes.

5 Success Criteria

- $\geq$90% JAX-backend branch coverage.
- Verified correctness of key functions (`ak.sum`, `ak.sort`, `ak.combinations`) under `grad/jit`, CPU/GPU.
- Completed physics workflow evaluation with performance and feature gap report.