The Journal of Open Source Software

PyTestLab: A Reproducible, Compliance-Aware Python Framework for Electronic Test & Measurement

Emmanuel A. Olowe [1¶] **and Danial Chitnis**[1]

1 The University of Edinburgh, UK ¶ Corresponding author

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor:

Submitted: 05 October 2025
Published: unpublished

Summary

Modern hardware characterization workflows involve coordinating multiple laboratory instruments (power supplies, oscilloscopes, multimeters, electronic loads, signal sources), executing structured sweeps or time-based acquisition routines, persisting results, and ensuring traceability. These activities are often implemented via ad-hoc Python scripts mixing raw SCPI commands with unstructured data handling and little or no provenance or compliance support. PyTestLab is an extensible Python framework that unifies:

1. Bench configuration (YAML (YAML Language Development Team, 2021)) with safety limits, automation hooks, calibration & DUT traceability.
2. A high-level `MeasurementSession` builder for declarative parameter sweeps or timed parallel acquisition loops with background stimulus tasks.
3. Pluggable instrument backends: VISA (*VPP-4.3*, 2014; *VPP-4.3.2*, 2022) and Lamb (Olowe & Chitnis, 2025), advanced deterministic simulation, session recording, and strict replay (sequence validation).
4. Structured experiment and measurement data stored in Polars DataFrames (Polars Developers, 2025) for efficient analytics and plotting.
5. A compliance layer providing cryptographic signing (ECDSA P-256 (*Digital Signature Standard (DSS)*, 2023; *SEC 2*, 2010)), linked timestamp chains (ISO 18014-3 style (Adams et al., 2001; Haber & Stornetta, 1991; Information Technology — Security Techniques — Time-Stamping Services — Part 3, 2009)), audit trail logging, and persistent signature envelopes.
6. Lightweight plotting and FFT / frequency-response helpers for rapid feedback.

The framework enables reproducible, testable, and audit-ready measurement workflows suitable for both exploratory R&D and regulated environments. Step-by-step tutorials from basic to advanced are linked from the README.

Statement of Need

PyTestLab addresses these gaps by coupling declarative bench specification (enabling infrastructural reproducibility) with a high-level acquisition API, while embedding compliance and integrity features by design rather than as afterthought plugins. Researchers, test engineers, and reliability or validation teams can therefore move from exploratory scripting to production-grade, provenance-rich pipelines without re-architecting code.

Existing Python tooling—e.g., PyVISA (Grecco et al., 2023) and vendor SDKs—focuses on transport-level communication, leaving users to implement experiment orchestration, safety enforcement, provenance capture, and deterministic regression testing manually. Higher-level packages such as PyMeasure (PyMeasure Developers, 2024) provide drivers and experiment scaffolding but typically do not offer deterministic command sequence replay coupled with

integrated compliance primitives. In regulated or quality-critical contexts (medical, aerospace, energy), requirements extend to tamper evidence, auditability of measurement changes, and controlled replay of prior sessions. No widely adopted open-source library in the instrumentation space currently integrates: (a) enforceable safety envelopes, (b) deterministic command sequence replay for validation, and (c) cryptographic measurement signing plus chained timestamp audit primitives — while also offering an ergonomic experiment builder and simulation layer.

Features

- Bench System & Safety: YAML-defined instrument ensembles; per-channel limits (voltage, current, amplitude, frequency) enforced at runtime by a `SafeInstrumentWrapper`; pre/post automation hooks map directly to shell/Python commands and instrument "macros" executed by the Bench automation engine.
- Instrument Abstraction: Automatic profile-based instantiation; SCPI engine (*Standard Commands for Programmable Instruments (SCPI), Volume 1*, 1999); command logging; backend polymorphism (VISA (*VPP-4.3*, 2014; *VPP-4.3.2*, 2022), Lamb (Olowe & Chitnis, 2025), simulation, recording, replay).
- Simulation Backend: YAML-driven state machine with regex/glob dispatch, sandboxed expressions, error queue emulation, artificial timing, deterministic behavior for CI.
- Recording & Replay: Recording backend generates enriched simulation profile; replay backend enforces exact SCPI sequence (raises on divergence) ensuring regression fidelity.
- MeasurementSession Builder:
  - Declarative parameter registration
  - Decorator-based acquisition functions
  - Parallel background tasks (stimulus) + foreground acquisition loop
  - Automatic Experiment aggregation (Polars DataFrame)
- Data & Analysis: `MeasurementResult` objects (scalar, array, waveform, DataFrame) with FFT and plotting convenience; experiment-level plotting.
- Compliance Layer:
  - Measurement hashing and ECDSA signing
  - Linked timestamp authority (hash chain) (Haber & Stornetta, 1991; Information Technology — Security Techniques — Time-Stamping Services — Part 3, 2009)
  - Audit trail (SQLite (Hipp & SQLite Project, 2025)) binding envelopes to actions
  - Database persistence of envelopes for post-hoc verification
- Extensibility: Clear boundaries (config, backends, drivers, compliance patching); user override path for simulation profiles; minimal monkey patch surface.
- Reproducibility & CI: Deterministic simulation plus replay => hardware-free continuous integration, easier debugging of instrumentation logic.

Design & Architecture

PyTestLab's layered architecture separates "what to measure" from "how to communicate":

1. Configuration Layer: Pydantic-backed models (Pydantic Developers, 2025) parse bench YAML (YAML Language Development Team, 2021), ensuring structural validation and enabling rich metadata (traceability, measurement plan, calibration references). Bench-defined safety limits are applied via a `SafeInstrumentWrapper` proxy, and automation hooks correspond to shell/Python commands and instrument macros that the bench executes in defined pre/post phases.
2. Instrument Core: A generic `Instrument` base encapsulates SCPI operations (*Standard Commands for Programmable Instruments (SCPI), Volume 1*, 1999), error queue handling, logging, communication timeouts, and binary block parsing; driver instances are created via an `AutoInstrument` factory selecting appropriate backend.

3. Backend Layer:
   - Simulation backend compiles SCPI dispatch tables (O(1) exact match + ordered regex fallback) and executes sandboxed state mutations.
   - Recording backend appends interaction logs and produces reproducible simulation profiles.
   - Replay backend enforces strict sequence determinism, supporting regression and audit scenarios.
4. Measurement Layer: `MeasurementSession` orchestrates cartesian parameter sweeps or interval-timed loops with parallel threads for stimulus tasks (e.g. PSU ramping, load pulsing). Acquisition functions return mapping objects merged into a growing structured dataset.
5. Data & Compliance: `MeasurementResult` is monkey-patched at import to emit a signed envelope for every result. The envelope contains: the SHA-256 (*Secure Hash Standard (SHS)*, 2015) of a canonicalized payload (sorted-key JSON of instrument, measurement_type, units, values_sha256, timestamp), an ECDSA P-256 signature (*Digital Signature Standard (DSS)*, 2023; *SEC 2*, 2010) over that hash, the PEM-encoded public key, an algorithm identifier, and a signature timestamp. Envelopes are persisted as JSON sidecars and in the database. A minimal linked time-stamp authority maintains an append-only hash chain (tsa.json) where each token stores idx, ts, sha_prev, sha_data, and sha_cum, enabling ISO 18014-3–style verification (Haber & Stornetta, 1991; Information Technology — Security Techniques — Time-Stamping Services — Part 3, 2009). An append-only audit trail (SQLite (Hipp & SQLite Project, 2025)) binds actor/action to each envelope hash and token index.
6. Analytics & Plotting: Lightweight wrappers centralize plotting semantics while deferring heavy analytics to established libraries.

Example Usage (Conceptual)

```
with Bench.open("bench.yaml") as bench:
    with Measurement(bench=bench) as session:
        session.parameter("voltage", [1.0, 2.0, 3.3], unit="V")
        session.parameter("frequency", [10e3, 50e3], unit="Hz")

        @session.acquire
        def capture(psu, scope, voltage, frequency):
            psu.channel(1).set(voltage=voltage)
            scope.set_acquisition_time(1e-3)
            waveform = scope.read_channels(1)
            vpp = scope.measure_voltage_peak_to_peak(1)
            return {"vpp": vpp.values, "waveform": waveform.values}

        experiment = session.run()
        experiment.plot(title="Vpp vs Parameters")
```

Replay-mode regression test (abbreviated):

```
psu = AutoInstrument.from_config("keysight/EDU36311A",
                                 backend_override=ReplayBackend(session_log, "psu"))
psu.connect_backend()
psu.set_voltage(1, 5.0)  # Raises if sequence diverges.
```

Reproducibility & Integrity

- Deterministic Replay: Ensures that test scripts remain protocol-stable; any deviation indicates unintentional behavioral drift.

- Cryptographic Envelopes: Each measurement result is hashed (SHA-256 (*Secure Hash Standard (SHS)*, 2015)) and signed (ECDSA P-256 (*Digital Signature Standard (DSS)*, 2023; *SEC 2*, 2010)); envelopes include public key, signature, algorithm, and timestamp.
- Linked Time-Stamp Chain: Hash chaining of envelope digests produces a tamper-evident chronological ledger (Adams et al., 2001; Haber & Stornetta, 1991; Information Technology — Security Techniques — Time-Stamping Services — Part 3, 2009).
- Audit Trail: Append-only records (SQLite (Hipp & SQLite Project, 2025)) link actor, action, and envelope hash; chain verification supports compliance audits.

Quality Assurance

PyTestLab emphasizes reliability and maintainability. Deterministic simulation and strict replay enable hardware-free continuous integration and regression testing of instrumentation logic. The codebase is type-annotated and checked with a static type checker; a linter enforces style and common bug patterns. Unit and smoke tests exercise configuration parsing, backends (simulation, recording, replay), core drivers, the measurement session builder, and compliance primitives. The minimum supported Python version is 3.11. Examples in the README run against the simulation backend and the replay harness to ensure they remain executable across releases. Releases are tagged with a changelog, and optional plotting dependencies are isolated behind an extra to keep the core lightweight.

Comparison to Existing Tools

Aspect	PyTestLab	PyVISA / Low-level Drivers	Ad-hoc Scripts	Proprietary Lab Suites
Declarative Bench Config	Yes	No	No	Partial
Deterministic Replay	Yes	No	No	Rare
Cryptographic Signing	Built-in	No	No	Rare / add-on
Linked Timestamping	Built-in	No	No	Rare
Simulation (Declarative)	Advanced YAML state machine	Minimal	Manual mocks	Varies
Parallel Acquisition Tasks	Yes	Manual threads	Manual threads	Varies
Structured Data (Polars)	Native	External	Optional	Varies

Related Work

Transport libraries such as PyVISA (Grecco et al., 2023) focus on low-level communication with instruments, leaving orchestration and provenance to users. Higher-level ecosystems like PyMeasure (PyMeasure Developers, 2024) provide drivers and experiment scaffolding; however, deterministic command sequence replay and integrated compliance primitives (cryptographic signing plus a linked timestamp audit chain) remain uncommon. PyTestLab complements this landscape by combining declarative bench configuration, an ergonomic acquisition API, and compliance-by-design features with CI-friendly simulation and replay.

Availability

PyTestLab is released under the Apache-2.0 license and targets Python 3.11. For reproducible installs at submission time, install from source: pip install -e . Optional plotting extras: pip install -e '.[plot]' When a published package is available, it can be installed via: pip install pytestlab Optional extras: pip install 'pytestlab[plot]' The command-line entry point (pytestlab) exposes utilities for profile inspection and record/replay workflows. Source code, issue tracking, and documentation are referenced from the project README.

Citations

Inline citations refer to instrumentation, scientific Python, progress bars, uncertainty propagation, cryptography, DataFrame processing, and instrumentation tooling (Costa-Luis, 2019; PyMeasure Developers, 2024; Grecco et al., 2023; Harris et al., 2020; Hunter, 2007; Lebigot, 2025; Polars Developers, 2025; (PyCA), 2025).

Limitations & Future Work

- Expansion to more instrument support is planned for future releases.

Acknowledgements

We acknowledge the open-source communities behind NumPy, Polars, Matplotlib, tqdm, uncertainties, and cryptography libraries whose foundational work enabled this project.

This work was produced under work funded by Keysight Technologies.

References

Adams, C., Cain, P., Pinkas, D., & Zuccherato, R. (2001). *Internet x.509 public key infrastructure time-stamp protocol (TSP)* (RFC 3161). IETF. https://doi.org/10.17487/RFC3161

Costa-Luis, C. O. da. (2019). tqdm: A fast, extensible progress meter for python and CLI. *Journal of Open Source Software*, *4*(37), 1277. https://doi.org/10.21105/joss.01277

Developers, PyMeasure. (2024). *PyMeasure*. https://doi.org/10.5281/zenodo.11241567. https://doi.org/10.5281/zenodo.11241567

Developers, Pydantic. (2025). *Pydantic: Data validation using python type hints*. https://docs.pydantic.dev/. https://docs.pydantic.dev/

Digital signature standard (DSS) (FIPS 186-5). (2023). NIST. https://doi.org/10.6028/NIST.FIPS.186-5

Grecco, H. E., Dartiailh, M. C., Thalhammer-Thurner, G., Bronger, T., & Bauer, F. (2023). PyVISA: The python instrumentation package. *Journal of Open Source Software*, *8*(84), 5304. https://doi.org/10.21105/joss.05304

Haber, S., & Stornetta, W. S. (1991). How to time-stamp a digital document. *Journal of Cryptology*, *3*, 99–111. https://doi.org/10.1007/BF00196791

Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M. H. van, Brett, M., Haldane, A., Fernández del Río, J., Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Hipp, D. R., & SQLite Project, the. (2025). *SQLite*. https://www.sqlite.org/.

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, *9*(3), 90–95. https://doi.org/10.1109/MCSE.2007.55

Information technology — security techniques — time-stamping services — part 3: Mechanisms producing linked tokens, Pub. L. No. ISO/IEC 18014-3:2009 (2009). https://www.iso.org/standard/50457.html

Lebigot, E. O. (2025). *Uncertainties: Python package for calculations with uncertainties*. https://uncertainties.readthedocs.io/. https://uncertainties.readthedocs.io/

Olowe, E. A., & Chitnis, D. (2025). LABIIUM: AI-enhanced zero-configuration measurement automation system. *2025 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*. https://doi.org/10.1109/I2MTC62753.2025.11079200

Polars Developers. (2025). *Polars: Lightning-fast DataFrame library for Rust and Python*. https://pola.rs/. https://pola.rs/

(PyCA), P. C. A. (2025). *Cryptography: Python library for cryptographic recipes and primitives*. https://cryptography.io/. https://cryptography.io/

SEC 2: Recommended elliptic curve domain parameters, version 2.0. (2010). Standards for Efficient Cryptography Group (SECG). https://www.secg.org/sec2-v2.pdf

Secure hash standard (SHS) (FIPS 180-4). (2015). NIST. https://doi.org/10.6028/NIST.FIPS.180-4

Standard commands for programmable instruments (SCPI), volume 1: Syntax and style. (1999). SCPI Consortium. https://www.ivifoundation.org/downloads/SCPI/scpi-99.pdf

VPP-4.3: The VISA library specification. (2014). IVI Foundation. https://people.ece.ubc.ca/~edc/3525.fall2014/datasheets/VISALibrarySpecification-vpp43_2014-06-19.pdf

VPP-4.3.2: VISA implementation specification for textual languages. (2022). IVI Foundation. https://www.ivifoundation.org/downloads/VISA/vpp432_2022-05-19.pdf

YAML Language Development Team. (2021). *YAML ain't markup language (YAML) version 1.2.2*. https://yaml.org/spec/1.2.2/.