Python Test Policies for the Universal Stochastic Predictor

Adaptive Meta-Prediction Development Consortium

February 21, 2026

Contents

Chapter 1

Purpose and Scope

1.1 Objective

The test suite must validate that the system is correct, stable, reproducible, and safe under the operational conditions defined by the specification. The objective is to prevent regressions, protect numerical integrity, and enforce strict quality policies across all layers.

1.2 Scope

Tests must cover the API, Core, Kernel, and I/O layers, including integration boundaries and cross-layer contracts. Coverage includes deterministic functionality, numerical invariants, error handling, and operational edge cases.

Chapter 2

Quality Expectations

2.1 Code Quality

All production and test code must follow consistent formatting, linting rules, and type safety constraints. The test policy requires deterministic behavior, explicit error handling, and no unused or dead code in critical paths.

2.2 Configuration Discipline

No hardcoded parameters are allowed in production logic. All tunable values must be driven by configuration. Tests must verify configuration loading, validation, and propagation, including defaults and boundary constraints.

2.3 No Fallback Policy

Silent fallbacks are forbidden. If a dependency, configuration, or invariant is missing or invalid, the system must fail fast with an explicit, actionable error. Tests must assert this behavior.

2.4 Security and Secrets

No secrets or credentials may be hardcoded. Tests must verify that secret material is injected only through authorized channels and never appears in logs, reports, or snapshots.

Chapter 3

Completeness Criteria

3.1 Coverage

Each layer must have unit tests for core logic, integration tests for inter-layer contracts, and robustness tests for stress conditions. The test suite must include negative cases for invalid inputs and explicit failure modes.

3.2 Public API Surface

All public callables must have at least one validation path that confirms correct execution and error handling. Public contracts must be verified for input validation, output structure, and invariants.

Chapter 4

Performance and Robustness

4.1 Performance Targets

Performance-sensitive components must meet documented latency and throughput thresholds under controlled inputs. Tests must report measured values and enforce explicit acceptance thresholds.

4.2 Robustness Targets

The system must remain stable under heavy tails, outliers, regime shifts, and numerical stress. Tests must quantify false-positive and false-negative rates for detectors and enforce stability criteria.

4.3 Determinism and Reproducibility

All tests must be deterministic under fixed seeds and configuration. Each run must record configuration, environment metadata, and timing so that results can be reproduced.

Chapter 5

Execution Policies

5.1 Isolation

Tests must run in an isolated environment with locked dependencies. No system-wide packages may influence results. The environment must be reproducible across supported platforms.

5.2 Tiered Execution

Unit, integration, robustness, I/O, hardware, validation, and edge-case tiers must be independently executable. Each tier must define clear entry and exit criteria.

Chapter 6

Reporting Requirements

Reports must summarize pass/fail status by tier, list missing dependencies, and provide lint and type statistics. Reports must be structured, versioned, and reproducible.

Chapter 7

Acceptance Criteria

The suite is accepted when:

- All required tiers pass in the baseline environment
- No blocking issues remain in formatting, linting, or type checks
- Dependency validation reports zero missing required packages
- Critical numerical invariants meet documented thresholds
- Performance targets meet or exceed acceptance thresholds