# Validation and Test Protocol for the Universal Stochastic Predictor

Adaptive Meta-Prediction Development Consortium

February 21, 2026

# Contents

# Chapter 1

# Unit Tests: Kernels and Fundamental Algorithms

These tests verify isolated implementation of critical algorithms without relying on global system state.

## 1.1 Entropy and Random Variable Generation

### 1.1.1 CMS Algorithm

**Test Case 1.1** (Validation of $\alpha$-Stable Distributions)**.** *Validate that generating $\alpha$-stable variables via the Chambers-Mallows-Stuck method produces distributions with the desired parameters $(\alpha, \beta, \gamma, \delta)$.*

**Criterion 1.1.** *For a sample size $N \geq 10^4$, empirical moments must match the theoretical properties of the stable distribution within a 95% confidence interval.*

### 1.1.2 Pseudo-Random Generator Integrity

**Test Case 1.2** (Mersenne Twister/PCG64)**.** *Verify the absence of serial correlations and the long period guarantees of the pseudo-random number generator.*

**Criterion 1.2.** *Apply standard statistical test batteries (TestU01, Diehard) and verify that no randomness tests fail. The generator period must be $\geq 2^{127}$ for Mersenne Twister and $\geq 2^{128}$ for PCG64.*

## 1.2 Singularity Analysis (SIA)

### 1.2.1 Holder Exponent Detection

**Test Case 1.3** (WTMM Validation)**.** *Use synthetic signals with known Holder exponent $H$ to validate that the WTMM (Wavelet Transform Modulus Maxima) algorithm recovers the singularity spectrum $D(h)$ with error $< 5\%$.*

**Criterion 1.3.** *Let $f(t)$ be a synthetic signal with known $H = H_0$. The Holder exponent estimated by WTMM must satisfy:*

$$|\hat{H} - H_0| < 0.05 \cdot H_0$$

*where $\hat{H}$ is the estimate from multiscale analysis.*

### 1.2.2 Cone of Influence

**Test Case 1.4** (Besov Influence Radius). *Verify that maxima linking in the scale space respects the influence radius defined by $C_{besov}$.*

**Criterion 1.4.** *For two consecutive maxima at scales $s_1 < s_2$, their temporal positions $(t_1, t_2)$ must satisfy:*

$$|t_2 - t_1| \leq C_{besov} \cdot (s_2 - s_1)$$

*where $C_{besov}$ is the Besov-space constant associated with the chosen wavelet.*

### 1.2.3 Soft Nyquist Limit Validation

The I/O specification defines a minimum data injection frequency to preserve multifractal analysis integrity. This test explicitly validates that operational limit.

**Test Case 1.5** (Multifractal Aliasing). *Gradually reduce the input sampling frequency until the spectrum $D(h)$ collapses, validating that the system detects the undersampling condition before irreversible degradation occurs.*

**Criterion 1.5.** *Consider a reference multifractal signal with known singularity spectrum $D_0(h)$ and nominal sampling frequency $f_0$. Use the following protocol:*

1. *Reduce sampling frequency by decimation: $f_k = f_0/2^k$ with $k = 0, 1, 2, \ldots$*
2. *For each $f_k$, compute the estimated spectrum $\hat{D}_k(h)$ via WTMM*
3. *Compute relative error:*

$$\varepsilon_k = \frac{\|D_0(h) - \hat{D}_k(h)\|_{L^2}}{\|D_0(h)\|_{L^2}}$$

4. *Monitor the dominant Holder exponent: $\hat{H}_k = \arg\max_h D_k(h)$*

*The system must satisfy:*

- ***Preventive detection:*** *Emit `FreezingTopologicalBranchEvent` when $\varepsilon_k > 0.05$ (error $> 5\%$)*
- ***Acceptance criterion:*** *The signal must trigger before Holder error exceeds 10%:*

$$\frac{|\hat{H}_k - H_0|}{H_0} < 0.10 \quad \textit{at the moment of freezing}$$

- ***Corrective action:*** *After the signal, freeze the topological branch weight at its last reliable value: $w_C \to w_C^{frozen}$ with no dynamic updates*

**Note 1.1.** *The Nyquist limit in multifractal analysis is a soft boundary. Self-similarity allows some tolerance to undersampling, but below a threshold fine-scale structures collapse and $D(h)$ degrades. This test ensures the system operates in a safe resolution regime.*

**Note 1.2.** *The critical frequency $f_{critical}$ depends on:*

1. *The scale range $[s_{min}, s_{max}]$ of the wavelet transform*
2. *The temporal support of the mother wavelet $\psi(t)$*
3. *The expected minimum Holder regularity $H_{min}$*

*A practical heuristic:*

$$f_{min} \geq \frac{10}{s_{min}} \cdot (1 + H_{min}^{-1})$$

*This ensures at least 10 points per minimum scale, adjusted by roughness.*

## 1.3 Algebraic Structures (Branch D)

### 1.3.1 Chen Identity

**Test Case 1.6** (Signature Concatenation)**.** *Validate that concatenating signatures of two path segments via the tensor product (⊗) equals the signature of the full path.*

**Criterion 1.6.** *Let $\gamma_1 : [0, T_1] \to \mathbb{R}^d$ and $\gamma_2 : [0, T_2] \to \mathbb{R}^d$ be two paths. Chen's identity states:*

$$\mathit{Sig}(\gamma_1 \star \gamma_2) = \mathit{Sig}(\gamma_1) \otimes \mathit{Sig}(\gamma_2)$$

*where $\gamma_1 \star \gamma_2$ denotes concatenation. Numerical error must be $< 10^{-6}$ in Euclidean norm.*

### 1.3.2 Scale Invariance

**Test Case 1.7** (Temporal Reparametrization)**.** *Check that the level-$M$ truncated signature is invariant under strictly increasing time reparametrizations of the input path.*

**Criterion 1.7.** *For a strictly increasing reparametrization $\phi : [0, 1] \to [0, 1]$ with $\phi(0) = 0$ and $\phi(1) = 1$, we must have:*

$$\mathit{Sig}^{(M)}(\gamma) = \mathit{Sig}^{(M)}(\gamma \circ \phi)$$

*where $\mathit{Sig}^{(M)}$ is the level-$M$ truncated signature.*

# Chapter 2

# Integration Tests and Stochastic Convergence

Validate that component interactions respect continuous probability laws and numerical stability conditions.

## 2.1 Stochastic Differential Equation (SDE) Solvers

### 2.1.1 Convergence of Numerical Schemes

**Test Case 2.1** (Euler-Maruyama vs Milstein). *For diffusion with non-constant volatility, verify that Milstein achieves strong convergence order 1.0 versus order 0.5 for Euler-Maruyama.*

**Criterion 2.1.** *Consider the SDE:*

$$dX_t = \mu(X_t, t)\,dt + \sigma(X_t, t)\,dW_t$$

*with non-constant $\sigma(x,t)$. For a sequence of time steps $\Delta t_n = 2^{-n}\Delta t_0$, the strong error must satisfy:*

$$\begin{aligned} E[|X_T - X_T^{EM}|^2] &= O(\Delta t^{0.5}) \quad \textit{(Euler-Maruyama)} \\ E[|X_T - X_T^{M}|^2] &= O(\Delta t^{1.0}) \quad \textit{(Milstein)} \end{aligned}$$

*where $X_T^{EM}$ and $X_T^{M}$ are the numerical approximations.*

### 2.1.2 Mixed CFL Condition

**Test Case 2.2** (Stability Violation). *Force a time step $\Delta t$ that violates the Courant-Friedrichs-Lewy restriction and confirm numerical instability or NaNs as expected behavior to calibrate the safety monitor.*

**Criterion 2.2.** *For an explicit scheme, the CFL condition requires:*

$$\Delta t \leq \frac{C}{\sup_x |\mu'(x)| + \sigma^2(x)/2}$$

*Force $\Delta t > 10C$ and verify divergence detection via:*

- *NaN or Inf emergence in simulated trajectories*
- *Instability alert emitted by the safety module*

## 2.2 Transport Optimization (Orchestrator)

### 2.2.1 Sinkhorn Algorithm Stability

**Test Case 2.3** (Log-Domain Convergence)**.** *Evaluate Sinkhorn convergence in the log domain with decreasing regularization $\varepsilon$, ensuring no underflow down to $\varepsilon \geq 10^{-4}$.*

**Criterion 2.3.** *The log-domain Sinkhorn-Knopp algorithm must converge when:*

$$\varepsilon \geq 10^{-4}$$

*For $\varepsilon < 10^{-4}$, the system must detect underflow risk and emit a warning. Convergence is measured by:*

$$\|K\, diag(u) K^T diag(v) - \mu\|_1 < 10^{-6}$$

*where $K_{ij} = \exp(-C_{ij}/\varepsilon)$ is the Gibbs kernel.*

### 2.2.2 Simplex Normalization

**Test Case 2.4** (Probabilistic Mass Conservation)**.** *Confirm that after any JKO update the sum of kernel weights is strictly $\sum_i \rho_i = 1.0$.*

**Criterion 2.4.** *After each JKO iteration:*

$$\rho^{n+1} = \arg \min_{\rho \in \mathcal{P}(\mathcal{X})} \left\{ W_2^2(\rho, \rho^n) + \tau \mathcal{F}[\rho] \right\}$$

*verify that $\rho^{n+1}$ belongs to the probability simplex:*

$$\sum_{i=1}^{N} \rho_i^{n+1} = 1.0, \quad \rho_i^{n+1} \geq 0 \quad \forall i$$

*with numerical tolerance $|\sum_i \rho_i^{n+1} - 1.0| < 10^{-10}$.*

## 2.3 HJB Solution via DGM (Branch B)

Branch B solves the Hamilton-Jacobi-Bellman equation with Deep Galerkin Method (DGM). These tests validate convergence and stability.

### 2.3.1 Gradient Stability

**Test Case 2.5** (Gradient Explosion Under High Volatility)**.** *Monitor gradient norms during PDE training to detect gradient explosions in high-volatility regimes.*

**Criterion 2.5.** *The HJB equation:*

$$\frac{\partial V}{\partial t} + \sup_{u \in \mathcal{U}} \{\mathcal{L}^u V(x,t) + f(x,u,t)\} = 0$$

*is approximated by a neural network $V_\theta(x,t)$ trained with DGM. During training, monitor:*

1. *Loss gradient norm:*

$$\|\nabla_\theta \mathcal{L}_{DGM}(\theta)\|_2 = \left\| \frac{\partial}{\partial \theta} \mathbb{E}[|PDE[V_\theta]|^2 + |BC[V_\theta]|^2] \right\|_2$$

2. *Apply gradient clipping when $\|\nabla_\theta \mathcal{L}\|_2 > C_{clip}$ with $C_{clip} = 10.0$*

3. *In high-volatility scenarios ($\sigma(x,t) > 2\sigma_0$), the system must:*
    - *Detect if $\|\nabla_\theta \mathcal{L}\|_2$ exceeds $C_{clip}$ for more than 5 consecutive iterations*
    - *Emit `GradientInstabilityEvent`*
    - *Adaptively reduce learning rate: $\eta \to 0.5\eta$*
    - *Verify stabilization within the next 20 epochs*

**Note 2.1.** *Gradient explosions in DGM often arise from interactions between second-order derivatives and diffusion coefficients. Early detection and adaptive clipping are essential for convergence in high-volatility regimes.*

### 2.3.2 Viscosity Solution Validation

**Test Case 2.6** (Crandall-Lions Comparison Principle)**.** *Validate that the neural solution $V_\theta(x,t)$ respects the comparison principle for viscosity solutions, ensuring uniqueness and consistency for non-linear PDEs.*

**Criterion 2.6.** *The comparison principle states: if $u$ is a viscosity supersolution and $v$ is a viscosity subsolution of the same HJB equation with $u \geq v$ on the boundary, then $u \geq v$ on the full domain.*

*Validation procedure:*

1. *Construct a reference solution $V_{ref}(x,t)$ using a standard method (upwind finite differences, method of lines)*
2. *Train the DGM network to obtain $V_\theta(x,t)$*
3. *Verify time monotonicity (finite horizon):*
$$V_\theta(x,t_1) \geq V_\theta(x,t_2) \quad \forall t_1 < t_2, \forall x \in \mathcal{D}$$
*(assuming non-negative costs)*
4. *Validate sub/supersolution constraints on a test grid $\{(x_i,t_j)\}_{i,j}$:*
$$\begin{aligned} \textit{Subsolution:} \quad & PDE[V_\theta](x_i,t_j) \leq \epsilon_{tol} \\ \textit{Supersolution:} \quad & PDE[V_\theta](x_i,t_j) \geq -\epsilon_{tol} \end{aligned}$$
*with $\epsilon_{tol} = 10^{-3}$*
5. *Compare with reference solution:*
$$\|V_\theta - V_{ref}\|_{L^\infty(\mathcal{D})} < \delta_{viscosity}$$
*where $\delta_{viscosity} = 0.05 \cdot \|V_{ref}\|_{L^\infty}$ (relative error $< 5\%$)*

**Note 2.2.** *Crandall-Lions viscosity theory is the rigorous framework for HJB equations. DGM networks, being smooth functions, must approximate these generalized solutions. The comparison principle test prevents spurious oscillations and causality violations.*

### 2.3.3 Training Entropy Test (Mode Collapse)

During neural PDE training, the model can collapse to trivial constant solutions. This test detects and prevents that behavior.

**Test Case 2.7** (Mode Collapse Detection)**.** *Verify that the neural network does not collapse to a trivial solution that satisfies the PDE but fails to capture structure.*

**Criterion 2.7.** *Mode collapse occurs when the network learns a minimum-variance solution that minimizes PDE loss without solving the boundary-value problem.*

*Detection protocol:*

1. *Compute spatial variance at time $t < T$:*

$$Var_x[V_\theta(x,t)] = \mathbb{E}_x[(V_\theta(x,t) - \bar{V}_t)^2]$$

   *where $\bar{V}_t = \mathbb{E}_x[V_\theta(x,t)]$*

2. *Compute variance of the terminal condition:*

$$Var[g(\xi)] = \mathbb{E}_\xi[(g(\xi) - \bar{g})^2]$$

3. *Verify proportionality:*

$$\kappa_{low} \leq \frac{Var_x[V_\theta(x,t)]}{Var[g(\xi)]} \leq \kappa_{high}$$

   *with typical thresholds $\kappa_{low} = 0.3$ and $\kappa_{high} = 1.2$*

4. *If the ratio falls below $\kappa_{low}$: detect mode collapse*

5. *Monitor differential entropy:*

$$H[V_\theta] = -\int p(v)\log p(v)\,dv$$

   *A collapsed solution yields $H[V_\theta] \to -\infty$ (delta distribution)*

*Acceptance criteria:*

- *Variance ratio in $[\kappa_{low}, \kappa_{high}]$ for at least 90% of $t \in [0,T]$*
- *Differential entropy satisfies $H[V_\theta] > H_{min}$*
- *Spatial gradient norm does not collapse:*

$$\mathbb{E}_x[\|\nabla_x V_\theta(x,t)\|_2] > \epsilon_{grad} > 0$$

**Note 2.3.** *Mode collapse is common in:*

1. *Uniform or symmetric boundary conditions*
2. *Excessive learning rates leading to minimum-norm solutions*
3. *Under-parameterized networks*
4. *Biased weight initialization*

*Early variance monitoring allows reinitialization or hyperparameter tuning before collapse is permanent.*

**Note 2.4.** *From a control theory perspective, a collapsed solution $V_\theta(x,t) \approx c$ yields a degenerate control policy $u^*(x,t)$ that does not respond to state changes, which is operationally useless.*

### 2.3.4 Mesh Refinement Convergence

**Test Case 2.8** (Consistency under Refinement)**.** *Verify that the DGM solution converges to the exact (or reference) solution as the density of collocation points increases.*

**Criterion 2.8.** *For nested meshes $\mathcal{M}_1 \subset \mathcal{M}_2 \subset \mathcal{M}_3$ with densities $N_1 < N_2 < N_3$:*

1. *Train DGM on each mesh to loss convergence: $\mathcal{L}_{DGM}^{(k)} < 10^{-4}$*
2. *Compute error on a fixed fine evaluation mesh:*
$$e_k = \|V_{\theta_k} - V_{ref}\|_{L^2(\mathcal{D}_{eval})}$$
3. *Verify monotone convergence:*
$$e_1 > e_2 > e_3$$
4. *Estimate empirical rate:*
$$r = \frac{\log(e_1/e_2)}{\log(N_2/N_1)}$$
*Expected $r \geq 0.5$ (at least order $N^{-0.5}$ convergence)*

### 2.3.5 Simplified Output Variance Test (Mode Collapse)

This test complements entropy monitoring with a direct variance ratio check.

**Test Case 2.9** (Minimum Variance Threshold)**.** *Compare output variance of the neural solution $V_\theta(x,t)$ against a reference solution $V_{ref}(x,t)$ and verify that the network captures at least 10% of the true variability.*

**Criterion 2.9.** *Let $V_{ref}(x,t)$ be a reference solution obtained with a standard method. For the trained DGM network $V_\theta(x,t)$:*

1. *Compute reference variance at time $t \in [0,T]$:*
$$Var_{ref}(t) = \frac{1}{|\mathcal{X}|} \sum_{x_i \in \mathcal{X}} (V_{ref}(x_i,t) - \bar{V}_{ref}(t))^2$$
2. *Compute neural variance:*
$$Var_\theta(t) = \frac{1}{|\mathcal{X}|} \sum_{x_i \in \mathcal{X}} (V_\theta(x_i,t) - \bar{V}_\theta(t))^2$$
3. *Variance ratio:*
$$R_{var}(t) = \frac{Var_\theta(t)}{Var_{ref}(t)}$$
4. ***Failure criterion:***
$$R_{var}(t) < 0.10 \quad \textit{for any } t \in [0, 0.9T]$$
5. ***Acceptance criterion:***
$$R_{var}(t) \geq 0.10 \quad \forall t \in [0, 0.9T]$$
*and*
$$median_{t\in[0,T]} R_{var}(t) \geq 0.50$$

**Note 2.5.** *If* $R_{var}(t) < 0.10$ *before convergence, interrupt training and adjust hyperparameters (learning rate, architecture, initialization). The 10% threshold is conservative; any solution below it is effectively constant. The* $t \in [0, 0.9T]$ *restriction excludes the terminal region where variance naturally contracts.*

**Note 2.6.** *This criterion complements the entropy test. Practical interpretation:*

- $R_{var} < 0.10$*: critical collapse (fail)*
- $0.10 \leq R_{var} < 0.30$*: low-variance warning*
- $R_{var} \geq 0.50$*: normal operation*

# Chapter 3

# Robustness Tests and Circuit Breakers

These tests verify system protection against market anomalies or data failures.

## 3.1 Outlier and Regime Handling

### 3.1.1 Outlier Injection

**Test Case 3.1** (Extreme Outlier Values). *Inject values $> 20\sigma$ in the input stream and verify that the system rejects the point, emits a validation alert, and preserves inertial weights.*

**Criterion 3.1.** *Given observations $\{y_t\}$ with rolling mean $\mu_t$ and standard deviation $\sigma_t$, inject:*

$$\tilde{y}_t = \mu_t + 20\sigma_t$$

*The system must:*

1. *Detect $|\tilde{y}_t - \mu_t| > \theta\sigma_t$ with $\theta = 10$*
2. *Reject the observation and NOT update the meta-state $\Xi_t$*
3. *Emit `OutlierDetectedEvent` with metadata for the rejected value*
4. *Keep weights $\{w_i\}_{i=A}^{D}$ unchanged*

### 3.1.2 CUSUM Trigger

**Test Case 3.2** (Structural Regime Change). *Simulate a regime change (structural drift) and validate that the change event is emitted exactly when $G_t^+$ exceeds the dynamic threshold $h$.*

**Criterion 3.2.** *CUSUM detects mean shifts via:*

$$G_t^+ = \max(0, G_{t-1}^+ + (y_t - \mu_0) - k)$$

*where $k$ is the slack and $h$ is the alarm threshold. Verify:*

1. *$G_t^+ > h \Rightarrow$ emit `RegimeChangedEvent`*
2. *Reset $G_t^+ = 0$ after detection*
3. *Detection delay is at most $\tau$ observations after the true change point*

## 3.2 Emergency Mode (Robustness Postulate)

### 3.2.1 Critical Singularity

**Test Case 3.3** (Extreme Roughness Regime)**.** *Artificially reduce the Holder exponent below $H_{min}$ and verify that the orchestrator forces $w_D \to 1.0$ and switches the cost to the Huber metric.*

**Criterion 3.3.** *Define a critical threshold $H_{min}$ (typically $H_{min} = 0.25$). When WTMM detects:*

$$\hat{H}_t < H_{min}$$

*The system must:*

1. *Activate emergency mode: $w_A = w_B = w_C = 0$, $w_D = 1.0$*
2. *Switch orchestrator cost from Wasserstein to Huber:*

$$C(x,y) = \begin{cases} \frac{1}{2}|x-y|^2 & \textit{if } |x-y| \leq \delta \\ \delta(|x-y| - \frac{\delta}{2}) & \textit{if } |x-y| > \delta \end{cases}$$

3. *Emit `CriticalSingularityEvent`*
4. *Maintain this state until $\hat{H}_t > H_{min} + \epsilon_{hysteresis}$*

# Chapter 4

# I/O and Persistence Tests

These tests guarantee operational continuity and latent state integrity.

## 4.1 Snapshot Protocol

### 4.1.1 Hot-Start

**Test Case 4.1** (State Continuity). *Serialize the meta-state $\Xi_t$, restart the system, and load it. The first prediction after restart must match the prediction without interruption.*

**Criterion 4.1.** *Define the full meta-state:*

$$\Xi_t = \left\{ \{w_i\}_{i=A}^{D}, \{\theta_i^*\}_{i=A}^{D}, \mathcal{H}_t, Sig_t, G_t^{\pm}, \mu_t, \sigma_t^2 \right\}$$

*Validation procedure:*

1. *At time $t_0$, serialize $\Xi_{t_0}$ to a binary file*
2. *Generate prediction $\hat{y}_{t_0+1}^{original}$*
3. *Restart the system (release memory)*
4. *Load $\Xi_{t_0}$ from file*
5. *Generate prediction $\hat{y}_{t_0+1}^{restored}$*
6. *Verify: $|\hat{y}_{t_0+1}^{original} - \hat{y}_{t_0+1}^{restored}| < 10^{-12}$*

### 4.1.2 Checksum Validation

**Test Case 4.2** (Cryptographic Integrity). *Corrupt a single bit in the snapshot file and verify that the system rejects the load via SHA-256, forcing a cold start.*

**Criterion 4.2.** *Each snapshot must include a SHA-256 hash. Load must:*

1. *Read the binary file*
2. *Compute $H' = SHA256(content)$*
3. *Compare to stored hash $H$*
4. *If $H' \neq H$: reject load, emit `CorruptedSnapshotEvent`, initialize cold start*
5. *If $H' = H$: proceed with deserialization*

*Validate by flipping one random bit and verifying rejection.*

## 4.2 I/O Failure Recovery

Persisting $\Xi_t$ is critical for continuity. These tests validate robustness against write, read, and storage failures.

### 4.2.1 Write Interruption (Atomicity)

**Test Case 4.3** (Power Loss Simulation)**.** *Simulate a sudden power loss during snapshot serialization and verify that partially written files are handled safely.*

**Criterion 4.3.** *Snapshotting must guarantee atomicity via write-then-rename:*

1. *Serialize $\Xi_t$ to temporary file:* `snapshot_{timestamp}.tmp`
2. *Compute $H = SHA256(\Xi_t)$ and append to file*
3. *Perform* `fsync()` *to flush to disk*
4. *Atomically rename:* `snapshot_{timestamp}.tmp` $\rightarrow$ `snapshot_{timestamp}.bin`

*Validation:*

1. *Start snapshot at time $t_0$*
2. *Interrupt at random progress $p \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$*
3. *Simulate abrupt termination (*`SIGKILL` *or I/O cut)*
4. *Restart system*
5. *The system must:*
   - *Detect absence of main* `.bin` *file*
   - *Detect presence of corrupted* `.tmp`
   - *Ignore the temp file*
   - *Load the latest valid snapshot before $t_0$*
   - *If none exist: execute cold start*
   - *Avoid infinite restart loops*

*Success criterion:*

$$\text{Recovery time} < T_{recovery} = 30 \text{ seconds}$$

*No functional degradation after recovery.*

**Note 4.1.** *Atomic snapshot writes avoid the torn write problem. Most modern file systems (ext4, XFS, NTFS, APFS) guarantee atomic rename, which this protocol relies on.*

### 4.2.2 Silent Disk Corruption

**Test Case 4.4** (Bit Rot and Storage Errors)**.** *Detect and handle silent data corruption that can occur between snapshot write and read.*

**Criterion 4.4.** *The protocol must:*

1. *Store verification metadata with each snapshot:*
   - *SHA-256 hash*
   - *Creation timestamp*

- *Serialization format version*
- *Optional CRC32 pre-check*

2. *On load:*
    - *Verify CRC32 if available*
    - *Verify full SHA-256*
    - *On failure, mark snapshot corrupt and search for previous valid snapshot*
3. *Retention policy:*
    - *Keep last $N = 5$ valid snapshots*
    - *Allow recovery from $t_{-k}$ if $t_0$ is corrupt*

*Validation:*

1. *Create valid snapshot at $t_0$*
2. *Corrupt random bytes*
3. *Attempt load*
4. *Verify system:*
    - *Detects corruption*
    - *Emits `CorruptedSnapshotEvent`*
    - *Falls back to previous valid snapshot*
    - *Executes cold start if no valid snapshot exists*
    - *Avoids infinite retry loops*

### 4.2.3 Disk Space Exhaustion

**Test Case 4.5** (Insufficient Capacity Handling)**.** *Validate behavior when storage is full during snapshot write.*

**Criterion 4.5.** *During snapshotting, the system must:*

1. *Before writing, verify free space:*

$$\mathit{FreeSpace} \geq 2 \times \mathit{EstimatedSize}(\Xi_t)$$

    *(factor 2 for temp + rename)*

2. *If insufficient space:*
    - *Emit `InsufficientStorageEvent`*
    - *Do not attempt write*
    - *Keep last valid snapshot*
    - *Continue operation in memory until space is available*
3. *If failure occurs mid-write:*
    - *Catch I/O exception*
    - *Delete corrupted temp file*
    - *Emit `SnapshotWriteFailedEvent`*

- *Preserve last valid snapshot*

*Validation:*

1. *Create a limited storage volume*
2. *Fill it, leaving insufficient space*
3. *Attempt snapshot*
4. *Verify graceful handling without crash*

**Note 4.2.** *Disk exhaustion is common in production. Design priorities: (1) do not corrupt valid snapshots, (2) degrade gracefully in memory, (3) provide clear telemetry, (4) recover automatically when space frees.*

# Chapter 5

# Adaptive and Topological Robustness

This chapter validates the theoretical guarantees introduced by Level 4 Autonomy: adaptive architecture, dynamic regularization, kurtosis-coupled CUSUM, and meta-optimization determinism. These tests are implementation-agnostic and verify mathematical correctness before hardware-specific validation.

## 5.1 Optimal Transport Resilience (Dynamic Sinkhorn Coupling)

### 5.1.1 Volatility-Coupled Regularization

**Test Case 5.1** (Dynamic Regularization Under Volatility Shocks)**.** *Validate the equation for dynamic regularization:*

$$\varepsilon_t = \max\left(\varepsilon_{\min}, \varepsilon_0 \left(1 + \alpha\sigma_t\right)\right)$$

*where $\sigma_t$ is the current volatility estimate and $\alpha$ is the coupling coefficient.*

**Criterion 5.1.** ***Setup:***

1. *Initialize Sinkhorn algorithm with baseline volatility $\sigma_0 = 0.01$*
2. *At iteration $t = 500$, inject a volatility shock: $\sigma_t = 100 \cdot \sigma_0$*
3. *Continue Sinkhorn iterations with dynamic $\varepsilon_t$ adjustment*

***Acceptance Criteria:***

1. ***Convergence Guarantee:*** *Algorithm must converge within `max_iterations` without divergence to `NaN` or `Inf`*
2. ***Wasserstein Distance Stability:*** *Final Wasserstein distance $W_2(\mu, \nu)$ must remain finite and satisfy:*

   $$W_2(\mu, \nu) \leq C \cdot (1 + \sigma_t)$$

   *where $C$ is a problem-dependent constant*
3. ***Log-Domain Arithmetic:*** *All internal computations must use log-domain representation to prevent numerical underflow*
4. ***Regularization Lower Bound:*** *Verify $\varepsilon_t \geq \varepsilon_{\min}$ at all iterations to prevent ill-conditioning*

***Edge Case:*** *Test with extreme shock $\sigma_t = 10^4$. System must either (a) converge with degraded tolerance, or (b) emit `SinkhornDivergenceWarning` and fall back to unregularized transport.*

**Note 5.1.** *Dynamic regularization is critical for regime transitions in financial time series where volatility can spike by 100x within minutes (e.g., flash crashes, central bank announcements). Fixed $\varepsilon$ causes either: (1) over-smoothing in calm regimes (loss of signal), or (2) numerical instability in volatile regimes (NaN propagation).*

## 5.2 Kurtosis-Coupled CUSUM (Adaptive Change Detection)

### 5.2.1 Heavy-Tailed Robustness

**Test Case 5.2** (Adaptive Threshold Under Kurtosis Variation). *Validate the Adaptive Threshold Lemma:*

$$h_t = h_0 \cdot \left(1 + \beta \cdot \frac{\kappa_t - 3}{\kappa_0 - 3}\right)$$

*where $\kappa_t$ is the empirical kurtosis, $\kappa_0 = 3$ (Gaussian baseline), and $\beta \in [0.3, 0.7]$ is the coupling strength.*

**Criterion 5.2.** ***Setup - Phase 1 (Gaussian Regime):***

1. *Generate white Gaussian noise: $x_t \sim \mathcal{N}(0,1)$, $t \in [1, 1000]$*
2. *Expected kurtosis: $\kappa \approx 3$*
3. *Expected threshold: $h_t \approx h_0$*

***Setup - Phase 2 (Heavy-Tailed Regime):***

1. *At $t = 1001$, switch to Student-t distribution: $x_t \sim t_{\nu=3}$ (heavy tails)*
2. *Expected kurtosis: $\kappa \approx 9$ for $\nu = 3$ (infinite for $\nu \leq 4$)*
3. *Expected threshold scaling: $h_t \approx h_0 \cdot (1 + \beta \cdot 2) \approx 2h_0$*

***Acceptance Criteria:***

1. ***Type I Error Constancy**: False positive rate (FPR) must remain approximately constant:*

   $$|FPR_{Phase\ 1} - FPR_{Phase\ 2}| < 0.05$$

   *where FPR is measured over 10 independent runs*

2. ***Threshold Scaling**: Verify threshold adapts according to kurtosis:*

   $$\frac{h_{t=1500}}{h_{t=500}} \in [1.5, 2.5]$$

   *for $\beta = 0.5$*

3. ***No Spurious Alarms**: CUSUM must NOT trigger regime change alarm during Phase 2 transition purely due to outliers. Only sustained mean shifts should trigger alarms.*

4. ***Grace Period Enforcement**: After kurtosis-induced threshold adaptation, system must observe grace period of $N_{grace}$ steps before accepting new regime*

***Failure Mode**: If threshold remains fixed ($h_t = h_0$), heavy-tailed regime will cause excessive false alarms (Type I Error $> 20\%$), violating statistical control.*

**Note 5.2.** *Kurtosis-coupled CUSUM is essential for financial markets where distribution tails fatten during crises (VIX spikes, credit events). Fixed thresholds calibrated for Gaussian regimes produce alarm floods when kurtosis increases, rendering change detection useless.*

## 5.3 Topological Adaptation (DGM Architecture Scaling)

### 5.3.1 Entropy-Driven Capacity Expansion

**Test Case 5.3** (Mode Collapse Prevention via Architecture Scaling)**.** *Validate the Entropy-Topology Coupling Theorem:*

$$\log(W \cdot D) \geq \log(W_0 \cdot D_0) + \beta \cdot \log(\kappa)$$

*where $W$ is network width, $D$ is depth, $\kappa = H_{current}/H_{baseline}$ is the entropy ratio, and $\beta \in [0.5, 1.0]$ is the coupling exponent.*

**Criterion 5.3.** ***Setup:***

1. *Initialize DGM with baseline architecture: $W_0 = 64$, $D_0 = 4$ (capacity $= 256$)*
2. *Train on terminal condition with baseline entropy: $H[g] = H_0$*
3. *At iteration $t = 100$, inject high-entropy terminal condition: $H[g'] = 5 \cdot H_0$ (entropy ratio $\kappa = 5$)*
4. ***Scenario A (Fixed Architecture)****: Keep $W = 64$, $D = 4$ (no adaptation)*
5. ***Scenario B (Adaptive Architecture)****: Scale to $W \cdot D \geq 256 \cdot 5^{0.7} \approx 770$ (e.g., $W = 128$, $D = 6$)*

***Acceptance Criteria:***

1. ***Mode Collapse Detection (Scenario A)****: With fixed architecture, solution entropy must degrade within 10 training steps:*

$$H_{solution}(t = 110) < \gamma \cdot H[g']$$

*where $\gamma = 0.5$ (50% entropy loss threshold). System must emit `ModeCollapseWarning`.*

2. ***Entropy Preservation (Scenario B)****: With adaptive architecture, solution entropy must remain high:*

$$H_{solution}(t = 110) \geq 0.9 \cdot H[g']$$

*(90% entropy retention)*

3. ***Capacity Scaling Formula****: Verify architecture expansion follows:*

$$W_{new} \cdot D_{new} \geq (W_0 \cdot D_0) \cdot \kappa^{\beta}$$

*with $\beta \in [0.5, 1.0]$*

4. ***XLA Recompilation Budget****: Architecture scaling must trigger at most 1 XLA recompilation per regime transition. Verify via JAX compilation cache hit rate $> 95\%$ after warmup.*

***Mathematical Justification****: From universal approximation theory, approximating a function with entropy $H$ in function space requires network capacity $\propto \exp(H)$. In log-space, this becomes linear: $\log(\textit{capacity}) \propto H$.*

**Note 5.3.** *Mode collapse in DGM is catastrophic because the solver converges to a low-dimensional manifold that ignores the true solution's complexity. Adaptive architecture prevents this by expanding representational capacity when entropy increases. This is analogous to adaptive mesh refinement in numerical PDEs.*

## 5.4 Meta-Optimization Determinism (TPE Checkpoint Persistence)

### 5.4.1 Resumption Bit-Exactness

**Test Case 5.4** (Checkpoint Save/Load Roundtrip Determinism). *Validate the TPE State Persistence Protocol ensuring bit-exact resumption after interruption.*

**Criterion 5.4.** ***Setup:***

1. *Initialize Deep Tuning meta-optimizer with fixed PRNG seed:* `seed = 42`
2. *Configure search space: 14 structural hyperparameters (DGM width/depth, Sinkhorn iterations, etc.)*
3. *Set iteration budget: 50 trials*

***Execution - Scenario A (Uninterrupted):***

1. *Run 50 consecutive trials without interruption*
2. *Record final best parameters $\theta^*_A$ and best objective value $f^*_A$*
3. *Record complete trial history: $\{(\theta_i, f_i)\}_{i=1}^{50}$*

***Execution - Scenario B (Interrupted):***

1. *Run first 25 trials*
2. *Save checkpoint with SHA-256 integrity hash:* `study_checkpoint_trial_25.pkl.sha256`
3. *Simulate process termination (*`SIGTERM`*)*
4. *Load checkpoint and resume: run remaining 25 trials*
5. *Record final best parameters $\theta^*_B$ and best objective value $f^*_B$*

***Acceptance Criteria:***

1. ***Bit-Exact Equivalence****: Final parameters must be identical:*

$$\theta^*_A = \theta^*_B \quad (\textit{all } 14 \textit{ hyperparameters})$$

   *Verify with:* `numpy.array_equal(theta_A, theta_B) == True`

2. ***Objective Value Parity****:*

$$|f^*_A - f^*_B| < 10^{-12}$$

   *(floating-point precision tolerance)*

3. ***Trial History Consistency****: All 50 trials must have identical parameters and objective values:*

$$\forall i \in [1, 50] : \quad \theta_i^A = \theta_i^B, \quad f_i^A = f_i^B$$

4. ***PRNG State Preservation****: After resumption, next sampled trial must be identical to what would have been sampled without interruption. Verify by seeding both scenarios and comparing trial 26 parameters.*
5. ***Checkpoint Integrity****: SHA-256 hash verification must pass on load. Corrupted checkpoint (modified bytes) must be rejected with* `IntegrityError`*.*

***Failure Modes to Test:***

1. ***Hash Mismatch**: Manually corrupt 1 byte in checkpoint file. Load must fail with integrity error.*
2. ***Missing Sidecar**: Delete* `.sha256` *file. Load must fail with* `FileNotFoundError`*.*
3. ***Study Name Mismatch**: Create checkpoint for study "A", attempt to load into study "B". Must fail with* `ValueError`*.*

**Note 5.4.** *Bit-exact determinism in meta-optimization is non-negotiable for reproducibility and debugging. Deep Tuning campaigns run for weeks; any checkpoint corruption or non-determinism forces full restart, wasting thousands of GPU-hours. SHA-256 integrity verification prevents silent corruption from disk errors or transmission failures.*

# Chapter 6

# Hardware Parity and Fidelity Tests (Cross-Platform)

Given heterogeneous targets (CPU, GPU, FPGA), add bit-consistency tests to ensure equivalent outputs across architectures.

## 6.1 Bit-Consistency Tests

**Test Case 6.1** (Multi-Architecture Equivalence)**.** *Verify that critical algorithms produce consistent numerical results across hardware platforms (CPU, GPU, FPGA), within precision limits of each architecture.*

**Criterion 6.1.** *For each critical component (random generation, signatures, SDE integration), execute identical inputs on:*

1. *CPU with IEEE 754 floating-point (64-bit)*
2. *GPU with floating-point (32 or 64-bit)*
3. *FPGA with fixed-point arithmetic (configurable precision)*

*Relative difference must satisfy:*

$$\frac{\|x^{CPU} - x^{GPU}\|_2}{\|x^{CPU}\|_2} < \epsilon_{GPU}, \quad \frac{\|x^{CPU} - x^{FPGA}\|_2}{\|x^{CPU}\|_2} < \epsilon_{FPGA}$$

*where $\epsilon_{GPU} = 10^{-6}$ for 32-bit arithmetic and $\epsilon_{FPGA}$ is the quantization error of the lower-precision hardware.*

## 6.2 Numerical Drift Test

### 6.2.1 Branch D: Signatures on FPGA vs CPU

**Test Case 6.2** (Fixed-Point Error Accumulation)**.** *Compare Branch D signatures on FPGA (fixed-point) against CPU (64-bit floating-point).*

**Criterion 6.2** (Drift Acceptance Criteria)**.** *For a path $\gamma : [0,T] \to \mathbb{R}^d$:*

$$Sig^{(M)}(\gamma) = \left(1, \int_0^T d\gamma_{t_1}, \int_0^T \int_0^{t_1} d\gamma_{t_2} \otimes d\gamma_{t_1}, \dots\right)$$

*For 10,000 iterations:*

1. *Compute CPU signature:* $Sig_{10000}^{CPU}$
2. *Compute FPGA signature:* $Sig_{10000}^{FPGA}$
3. *Compute accumulated divergence:*
$$\Delta_{acum} = \|Sig_{10000}^{CPU} - Sig_{10000}^{FPGA}\|_\infty$$
4. ***Primary criterion:***
$$\Delta_{acum} \leq N \cdot \epsilon_{quant}$$
*where* $N = 10000$ *and* $\epsilon_{quant} = 2^{-n+1}$ *for n-bit fixed point*
5. ***Secondary topological preservation:***
   - *Norm preservation:*
   $$\frac{|\|Sig_{10000}^{FPGA}\|_2 - \|Sig_{10000}^{CPU}\|_2|}{\|Sig_{10000}^{CPU}\|_2} < \tau_{norm}$$
   *with* $\tau_{norm} = 0.01$
   - *Sign preservation:*
   $$sgn(s_i^{(k),CPU}) = sgn(s_i^{(k),FPGA}) \quad \forall i, k$$
   - *Angular distance:*
   $$\cos(\theta) = \frac{\langle Sig_{10000}^{CPU}, Sig_{10000}^{FPGA}\rangle}{\|Sig_{10000}^{CPU}\|_2 \cdot \|Sig_{10000}^{FPGA}\|_2}$$
   *Require* $\cos(\theta) > 0.9999$ *(angular deviation* $< 0.81°$*)*
   - *Relative error per level:*
   $$\frac{\|Sig_{10000}^{CPU,(k)} - Sig_{10000}^{FPGA,(k)}\|_2}{\|Sig_{10000}^{CPU,(k)}\|_2} < \tau_k$$
   *where* $\tau_k = 0.05 \cdot k$

**Note 6.1.** *Fixed-point drift is inevitable due to repeated truncation. The primary criterion bounds cumulative error by the worst-case independent error accumulation. For Branch D, qualitative properties must also be preserved: orientation, angular similarity, and low-level tensor accuracy.*

**Note 6.2.** *For FPGA with* $n = 32$ *bits (Q16.16),* $\epsilon_{quant} = 2^{-15} \approx 3.05 \times 10^{-5}$*. After 10,000 iterations, the primary criterion allows* $\Delta_{acum} \leq 0.305$*, while topological criteria enforce tighter constraints:*

- *Relative norm error* $< 1\%$
- *Angular deviation* $< 1°$
- *Level-1 error* $< 5\%$

### 6.2.2 Deterministic Reproducibility

**Test Case 6.3** (Controlled Seed Initialization)**.** *Guarantee that, given the same pseudo-random seed, all platforms (CPU, GPU, FPGA) produce the same state sequence.*

**Criterion 6.3.** *Set deterministic seed* $s_0$ *and run 1,000 simulation steps on each platform. Verify:*
$$\{X_t^{CPU}\}_{t=1}^{1000} = \{X_t^{GPU}\}_{t=1}^{1000} = \{X_t^{FPGA}\}_{t=1}^{1000}$$
*Equality must be bit-for-bit for platforms with the same representation (CPU and GPU floating-point). For FPGA, compare after converting fixed-point to floating-point.*

## 6.3 Latency and Throughput Validation

**Test Case 6.4** (Cross-Platform Performance Benchmark)**.** *Measure execution time and throughput for each predictor branch on all architectures to identify bottlenecks and validate heterogeneous acceleration.*

**Criterion 6.4.** *For a batch of* $N = 1000$ *predictions:*

$$T_{CPU} = \textit{total time on CPU}$$
$$T_{GPU} = \textit{total time on GPU}$$
$$T_{FPGA} = \textit{total time on FPGA}$$

*Expected:*

- *GPU outperforms CPU for massively parallel operations:* $T_{GPU} < 0.3 \cdot T_{CPU}$
- *FPGA outperforms CPU for deterministic low-latency operations:* $T_{FPGA} < 0.1 \cdot T_{CPU}$

*Throughput:*

$$\textit{Throughput} = \frac{N}{T} \quad \textit{[predictions/second]}$$

# Chapter 7

# Final Validation Protocol (Causality)

This is the definitive predictive test before any deployment.

## 7.1 Generalization

**Test Case 7.1** (Rolling Walk-Forward)**.** *Zero look-ahead bias. Training uses only data strictly prior to the test horizon.*

**Criterion 7.1.** *Split the dataset into rolling windows:*

$$\mathcal{D} = \{(t_1, y_1), \ldots, (t_N, y_N)\}$$

*For each test window $\mathcal{T}_k = \{t_{n_k}, \ldots, t_{n_k+w}\}$:*

1. *Train only with $\mathcal{D}^k_{train} = \{(t_i, y_i) : t_i < t_{n_k}\}$*
2. *Predict on $\mathcal{T}_k$*
3. *Advance window: $k \to k + 1$*
4. *Aggregate out-of-sample metrics (RMSE, MAE, Sharpe ratio)*

*The system must ensure that no future data is used to train the model at time $t$.*

## 7.2 Meta-Optimization Efficiency

**Test Case 7.2** (Bayesian Optimization)**.** *Iterative improvement of Expected Improvement on generalization error compared to random search.*

**Criterion 7.2.** *Use a Gaussian Process surrogate model for the hyperparameter space $\Theta = \{\alpha, \beta, \gamma, \ldots\}$. For $n$ optimization iterations:*

1. *Random search: $\theta_i \sim Uniform(\Theta)$*
2. *Bayesian optimization: $\theta_i = \arg\max_\theta EI(\theta|\mathcal{D}_{1:i-1})$*

*Acceptance criterion:*

$$\min_{i \le n} \mathcal{L}(\theta_i^{BO}) < \min_{i \le n} \mathcal{L}(\theta_i^{random})$$

*where $\mathcal{L}$ is validation loss. Improvement must be significant (p-value $< 0.05$ in Mann-Whitney test).*

## 7.3 Temporal Integrity

**Test Case 7.3** (Staleness Metric (TTL))**.** *Cancel JKO update if target signal delay $y_{target}$ exceeds $\Delta_{max}$.*

**Criterion 7.3.** *Define time-to-live:*

$$TTL(y_t) = t_{current} - t$$

*If:*

$$TTL(y_t) > \Delta_{max}$$

*The system must:*

1. *Discard the signal*
2. *NOT perform JKO update*
3. *Emit* `StaleDataEvent`
4. *Log the rejected timestamp*

*Typical value: $\Delta_{max} = 5$ seconds for high-frequency systems.*

### 7.3.1 Lag Injection Test

The staleness policy prevents operating with obsolete weights. This test validates behavior under extreme latency.

**Test Case 7.4** (Artificial Signal Delay)**.** *Artificially delay $y_{target}$ beyond $\Delta_{max}$ and validate that degraded inference mode is activated and optimal transport is suspended.*

**Criterion 7.4.** *Validation procedure:*

1. *Configure system with $\Delta_{max} = 5$ seconds*
2. *Generate real-time data stream with correct timestamps*
3. *Inject delayed signal $\tilde{y}_t$:*
$$TTL(\tilde{y}_t) = t_{current} - t = \Delta_{max} + \delta$$
*where $\delta > 0$ (typically $\delta = 1$ second)*
4. *Verify the sequence:*
    - *Detect $TTL(\tilde{y}_t) > \Delta_{max}$*
    - *Activate* `DegradedInferenceMode` = `True`
    - *Suspend JKO transport:*
    $$\textit{JKO\_update}(\rho^{n+1}) \rightarrow \textit{SUSPENDED}$$
    - *Freeze orchestrator weights at last valid value: $\{w_i\}_{frozen}$*
    - *Emit events:*
        - `StaleDataEvent` *with metadata $(t, t_{current}, TTL)$*
        - `DegradedInferenceModeActivated`
    - *Log lag duration for post-mortem analysis*
    - *Continue predictions using frozen weights only*

5. *Verify recovery when fresh signals satisfy* $TTL(y_t) < 0.8 \cdot \Delta_{max}$*:*
    - *Deactivate* `DegradedInferenceMode`
    - *Resume JKO transport*
    - *Emit* `NormalOperationRestoredEvent`

*Acceptance criteria:*

- *Detection time* $< 100$ *ms from receiving* $\tilde{y}_t$
- *No JKO iteration executed with stale data*
- *No crash or undefined state*
- *Predictions continue (degraded) with last valid configuration*

**Note 7.1.** *Degraded inference mode is critical in high-frequency systems. Operating with stale weights is equivalent to optimizing the wrong objective. It is better to operate with consistent static weights than with weights optimized on obsolete data.*

# Chapter 8

# Edge Cases and Operational Limits

This chapter documents boundary conditions for theoretical and operational behavior.

## 8.1 CUSUM: Adaptive Dynamic Threshold

**Test Case 8.1** (Volatility Regime Adaptation)**.** *Validate that the CUSUM threshold adapts correctly to low and high volatility via $h = k \cdot \sigma_{resid}$.*

**Criterion 8.1.** *CUSUM uses dynamic threshold:*

$$h_t = k \cdot \sigma_{resid,t}$$

*where $k \in [3, 5]$ and $\sigma_{resid,t}$ is rolling residual standard deviation.*

*Validation:*

1. ***Low volatility:***
   - *Generate signal with $\sigma_{true} = 0.01$*
   - *Estimate $\hat{\sigma}_{resid} \approx 0.01$*
   - *Verify $h_t = k \cdot 0.01$*
   - *Inject small drift $\Delta\mu = 0.05$*
   - *Detector should trigger when $G_t^+ > h_t$*
2. ***High volatility:***
   - *Generate signal with $\sigma_{true} = 0.50$*
   - *Estimate $\hat{\sigma}_{resid} \approx 0.50$*
   - *Verify $h_t = k \cdot 0.50$*
   - *Inject the same drift $\Delta\mu = 0.05$*
   - *Detector should NOT trigger*
   - *Inject larger drift $\Delta\mu = 2.0$*
   - *Detector should trigger*
3. ***Transition:***
   - *Simulate transition from low to high volatility*
   - *Verify $h_t$ updates smoothly (rolling window)*
   - *No spurious activations during transition*

*Acceptance criterion:*

$$\frac{h_{high}}{h_{low}} = \frac{\sigma_{high}}{\sigma_{low}} \pm 0.1$$

*Threshold ratio must match volatility ratio within 10% tolerance.*

## 8.2 Orchestrator: Maximum Entropy Convergence

**Test Case 8.2** (Uniform Weights under Total Uncertainty)**.** *Confirm that the system converges to uniform weights $w = [0.25, 0.25, 0.25, 0.25]$ when Sinkhorn regularization tends to infinity ($\varepsilon \to \infty$), representing maximum uncertainty.*

**Criterion 8.2.** *In Sinkhorn, entropy regularization $\varepsilon$ controls smoothing:*

$$\min_{\pi \in \Pi(\mu,\nu)} \{\langle C, \pi \rangle - \varepsilon H(\pi)\}$$

*with $H(\pi) = -\sum_{ij} \pi_{ij} \log \pi_{ij}$.*

*Limit behavior:*

1. *$\varepsilon \to 0$: deterministic optimal transport*
2. *$\varepsilon \to \infty$: maximum entropy dispersion*

*Validation:*

1. *Configure four branches $(A, B, C, D)$*
2. *Run with $\varepsilon_k = 10^k$ for $k \in \{0, 1, 2, 3, 4\}$*
3. *Record weights $\{w_A^k, w_B^k, w_C^k, w_D^k\}$*
4. *Verify convergence:*
$$\lim_{\varepsilon \to \infty} \{w_i\} = \left\{\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}\right\}$$
5. *Numerical criterion for $\varepsilon = 10^4$:*
$$\max_{i \in \{A,B,C,D\}} |w_i - 0.25| < 0.01$$

*Interpretation:*

- *High $\varepsilon$ implies indistinguishable branches, thus uniform weights*
- *Maximum entropy principle (Jaynes)*

**Note 8.1.** *This test validates that the orchestrator respects fundamental information theory. Under total uncertainty it must not favor any branch.*

## 8.3 Branch D: Time Reparametrization Invariance

**Test Case 8.3** (Signature Invariance to Time Warping)**.** *Test a signal and its time-stretched variants; the rough path signature must be identical under strictly increasing reparametrizations.*

**Criterion 8.3.** *Signature invariance:*

$$Sig(\gamma) = Sig(\gamma \circ \phi)$$

*for any strictly increasing $\phi : [0, 1] \to [0, 1]$ with $\phi(0) = 0$, $\phi(1) = 1$.*

*Validation:*

1. *Generate reference signal $\gamma(t)$ for $t \in [0, 1]$*
2. *Compute truncated signature $S_0 = Sig^{(M)}(\gamma)$*

3. *Apply nonlinear reparametrizations:*

$$\phi_1(t) = t^2$$
$$\phi_2(t) = \sqrt{t}$$
$$\phi_3(t) = \frac{1}{2}(1 - \cos(\pi t))$$

4. *For each, compute:*

$$\gamma_i(t) = \gamma(\phi_i(t)), \quad S_i = Sig^{(M)}(\gamma_i)$$

5. *Verify:*

$$\|S_i - S_0\|_2 < \epsilon_{inv} \quad \forall i \in \{1, 2, 3\}$$

*with* $\epsilon_{inv} = 10^{-8}$

*Negative control:*

1. *Apply a non-monotone reparametrization* $\psi(t) = t^2 - 0.5t$
2. *Verify* $Sig(\gamma \circ \psi) \neq Sig(\gamma)$

**Note 8.2.** *Signature invariance to time warping captures intrinsic path geometry regardless of execution speed. In finance, this means the signature captures the shape of a price move whether it occurred in 1 minute or 1 hour.*

## 8.4 Edge Case Summary Table

Table 8.1: Limit Scenarios and Edge Cases

| **Module** | **Test Scenario** | **Purpose** |
|---|---|---|
| CUSUM | Dynamic threshold | Validate $h = k \cdot \sigma_{resid}$ adapts to low/high volatility |
| Orchestrator | Maximum entropy ($\varepsilon \to \infty$) | Confirm convergence to uniform weights $[0.25, 0.25, 0.25, 0.25]$ |
| Branch D | Time reparametrization invariance | Signature remains identical under time warping ($< 10^{-8}$ error) |

# Chapter 9

# Acceptance Criteria Summary

Table 9.1: Global Validation Table

| **Test** | **Method** | **Acceptance Criterion** |
|---|---|---|
| Generalization | Rolling walk-forward | No look-ahead bias; training uses $t < t_{test}$ |
| Meta-optimization efficiency | Bayesian optimization (GP) | Expected Improvement beats random search |
| Temporal integrity | TTL staleness metric | Cancel JKO update if TTL$(y) > \Delta_{max}$ |
| Lag injection | Artificial delay $> \Delta_{max}$ | Immediate degraded mode; JKO suspended |
| Nyquist soft limit | Multifractal aliasing (SIA) | Freeze topological branch before $H$ error exceeds 1 |
| HJB-DGM stability | Gradient monitoring | Clip and reduce $\eta$ under sustained explosion |
| Viscosity solutions | Comparison principle | Error < 5% vs reference; sub/supersolution verified |
| Mode collapse | Training entropy | Variance ratio in $[0.3, 1.2]$ and proportional to term |
| I/O atomicity | Write interruption | Safe cold start; recovery < 30 s |
| Silent corruption | Bit rot detection | SHA-256 check with fallback to previous snapshot |
| Dynamic CUSUM | Volatility adaptation | Threshold ratio follows volatility ratio within 10% |
| Maximum entropy | Sinkhorn $\varepsilon \to \infty$ | Uniform weights within 1% |
| Time invariance | Path reparametrization | Signature error $< 10^{-8}$ |
| Bit consistency | Cross-platform tests | CPU/GPU/FPGA equivalence within quantization |
| Numerical drift | Branch D fixed-point | Cumulative divergence $\leq N \cdot \epsilon_{quant}$ after 10k iterati |

## 9.1 Final Considerations

All protocols here are language-agnostic and based solely on the mathematical and algorithmic foundations of the Universal Stochastic Predictor.

**Note 9.1.** *The full test suite must run before any production deployment. Results must be documented in a validation report including:*

- *Executive summary of all tests executed*
- *Numeric metrics for each criterion*
- *Convergence plots and distributions*

- *Failure case analysis (if any)*
- *Parameter calibration recommendations*

**Note 9.2.** *This protocol is a living document that must evolve with the system. Any architectural change in theory must be reflected in new test cases or updates to existing ones.*