

Numerical and Algorithmic Implementation Treatise for Universal Stochastic Predictors

Adaptive Meta-Prediction Development Consortium

February 20, 2026

Contents

1 Discretization Fundamentals and Monte Carlo Simulations	3
1.1 Pseudo-Random Number Generation	3
1.1.1 Chambers-Mallows-Stuck (CMS) Algorithm	3
1.2 Stochastic Integration Schemes	3
1.2.1 Euler-Maruyama Scheme	3
1.2.2 Milstein Scheme	4
1.3 Jump Process Simulation (Branch C)	4
2 System Identification Engine (SIA) Implementation	5
2.1 Multifractal Estimation (WTMM)	5
2.2 Regime Change Detection (CUSUM Test)	5
2.3 Sensitivity Computation (Malliavin/AAD)	6
2.3.1 Tangential Processes and Bismut-Elworthy-Li	6
2.3.2 Delta-Malliavin Monte Carlo Algorithm	7
3 Numerical Solvers for Prediction Kernels	8
3.1 Branch A: Hilbert Projection and Wiener Filtering	8
3.1.1 Levinson-Durbin Recursive Algorithm	8
3.2 Branch B: HJB Equation and Viscosity Methods	8
3.2.1 Monotone Finite Difference Schemes	8
3.2.2 Deep Galerkin Method (DGM)	9
3.3 Branch C: Jump Integro-Differential Equation	9
3.3.1 Delta-Malliavin Algorithm on Poisson Spaces	9
3.3.2 IMEX (Implicit-Explicit) Scheme for PIDEs	9
3.4 Branch D: Signature Computation	10
3.4.1 Chen Identity and Truncation	10
3.4.2 Log-Signatures	10
4 Orchestrator: Regularized Optimal Transport	11
4.1 Robustness Circuit Breaker (Pre-Orchestrator)	11
4.2 Sinkhorn-Knopp Algorithm (Dual Space)	11
4.3 JKO Proximal Scheme	12
4.4 Dynamic Sinkhorn Regularization: Coupling to Local Volatility	12
5 Software Architecture and Parallelism	14
5.1 Object-Oriented Construction Patterns	14
5.1.1 Suggested Class Structure	14
5.2 Heterogeneous Computing and Acceleration	14
5.2.1 GPU (CUDA/OpenCL)	14
5.2.2 FPGA (Field-Programmable Gate Array)	15

6 Numerical Stability Considerations	16
6.1 CFL Condition (Courant-Friedrichs-Lowy)	16
6.2 Log-Signature Stability	16
7 Governance of Heuristic Metaparameters	17
7.1 Taxonomy and Analytical Bounds (Safe Harbors)	17
7.1.1 Discretization and Truncation Parameters	17
7.1.2 Regularization and Stability Parameters	17
7.1.3 Decision Thresholds (Hard Boundaries)	18
7.2 Causal Cross-Validation (Walk-Forward Validation)	18
7.3 Derivative-Free Meta-Optimization (Bayesian Optimization)	18
7.3.1 Tiered Search Space Architecture	18
7.3.2 Persistence and Resumability Protocol	20
7.3.3 Meta-Optimization Execution Workflow	22
7.3.4 Objective Function Computation	22
7.3.5 Computational Budget Management	22

Chapter 1

Discretization Fundamentals and Monte Carlo Simulations

1.1 Pseudo-Random Number Generation

The stochastic integrator relies on an entropy source $\xi \sim \mathcal{D}$.

- **Gaussian:** For Brownian motion $dW_t \approx \sqrt{\Delta t}Z$, with $Z \sim \mathcal{N}(0, 1)$. Recommended generators are Mersenne Twister or PCG64 for long periods.
- **Levy/Jumps:** Use the Chambers-Mallows-Stuck method to simulate stable variables $S(\alpha, \beta, \gamma, \delta)$.

1.1.1 Chambers-Mallows-Stuck (CMS) Algorithm

To generate a standard α -stable random variable $S(\alpha, \beta = 0, \gamma = 1, \delta = 0)$ with $\alpha \neq 1$:

1. Sample $U \sim \text{Uniform}(-\pi/2, \pi/2)$ and $W \sim \text{Exponential}(1)$.

2. Compute:

$$X = \frac{\sin(\alpha U)}{(\cos U)^{1/\alpha}} \cdot \left[\frac{\cos((1-\alpha)U)}{W} \right]^{(1-\alpha)/\alpha}$$

3. Return X . For the general case $Y \sim S(\alpha, \beta, \gamma, \delta)$, apply the corresponding affine transform.

1.2 Stochastic Integration Schemes

1.2.1 Euler-Maruyama Scheme

For the stochastic ODE $dX_t = b(X_t)dt + \sigma(X_t)dW_t$, the first-order discretization is:

Algorithm 1 Euler-Maruyama Integrator

- 1: **Input:** $X_0, T, N, b(\cdot), \sigma(\cdot)$
 - 2: $\Delta t \leftarrow T/N$
 - 3: $X \leftarrow \text{array of length } N + 1$
 - 4: **for** $k \leftarrow 0$ **to** $N - 1$ **do**
 - 5: $Z \sim \mathcal{N}(0, 1)$
 - 6: $X_{k+1} \leftarrow X_k + b(X_k)\Delta t + \sigma(X_k)\sqrt{\Delta t}Z$
 - 7: **end for**
 - 8: **Return** X
-

1.2.2 Milstein Scheme

Improves strong convergence to order 1.0. Requires the derivative of volatility $\sigma'(x)$.

$$\hat{X}_{k+1} = \hat{X}_k + b_k \Delta t + \sigma_k \Delta W_k + \frac{1}{2} \sigma_k \sigma'_k ((\Delta W_k)^2 - \Delta t)$$

Note: If $\sigma(x)$ is constant (additive volatility), Milstein reduces to Euler-Maruyama.

1.3 Jump Process Simulation (Branch C)

For $dX_t = b(X_t)dt + \sigma(X_t)dW_t + dJ_t$, where J_t is a compound Poisson process with intensity λ and jump size $Y \sim F_Y$:

1. Simulate the number of jumps in $[t, t + \Delta t]$: $N_{\text{jump}} \sim \text{Poisson}(\lambda \Delta t)$.
2. If $N_{\text{jump}} > 0$, generate sizes $Y_1, \dots, Y_{N_{\text{jump}}}$.
3. Update: $X_{k+1} = X_{k+1}^{\text{diff}} + \sum Y_i$.

Chapter 2

System Identification Engine (SIA) Implementation

2.1 Multifractal Estimation (WTMM)

The WTMM (Wavelet Transform Modulus Maxima) algorithm extracts the singularity spectrum $D(h)$ in quasi-real time.

Algorithm 2 Detailed Discrete WTMM - Maxima Tracking

```
1: Input: Time series  $X$ , scales  $a_i \in \{2^0, 2^{0.1}, \dots, 2^J\}$  (dense dyadic scales).
2: Step 1: CWT (FFT) and Local Maxima
3: For each scale  $a_j$ , extract the maxima set  $M_j = \{(b, |W_{a_j}(b)|)\}$ .
4: Step 2: Maxima Linking (Tracking)
5: Initialize lines  $\mathcal{L} = \{(b, |W_{a_J}(b)|)\}_{b \in M_J}$  (from coarse scale).
6: for  $j \leftarrow J - 1$  downto 1 do
7:   for each line  $L \in \mathcal{L}$  with last point  $(b_{\text{last}}, \text{mod})$  do
8:     Search  $(b_{\text{curr}}, \text{mod}_{\text{curr}}) \in M_j$  such that  $|b_{\text{curr}} - b_{\text{last}}| < C \cdot a_j$  (cone of influence).
9:     If multiple candidates, choose the one with highest modulus.
10:    Extend  $L \leftarrow L \cup \{(b_{\text{curr}}, \text{mod}_{\text{curr}})\}$ .
11:   end for
12: end for
13: Step 3: Partition Function For moments  $q \in [-5, 5]$ :
14:  $Z(q, a) = \sum_{L \in \mathcal{L}} (\sup_{(b, \text{mod}) \in L \cap \text{scale}(a)} \text{mod})^q$ 
15: Step 4: Exponents
16:  $\tau(q) \leftarrow$  slope of the linear regression  $\log Z(q, a)$  vs  $\log a$ .
17: Output: Legendre spectrum  $D(h) = \min_q (qh - \tau(q))$ .
```

2.2 Regime Change Detection (CUSUM Test)

The `RegimeChangedEvent` is emitted when the Page statistic of cumulative residuals exceeds an adaptive threshold. To improve robustness in heavy-tail regimes, we incorporate a kurtosis adjustment.

Implementation Note 2.1 (Kurtosis Adjustment Rationale) *The term $(1 + \ln(\kappa_t/3))$ adjusts the threshold based on tail heaviness:*

- For Gaussian distributions: $\kappa \approx 3 \Rightarrow \ln(\kappa/3) \approx 0$, threshold remains $h_t \approx k\sigma_t$

Algorithm 3 Discrete CUSUM with Kurtosis Adjustment

```

1: Input: Standardized residuals  $e_t$ , base factor  $k$ , rolling window  $W$ .
2:  $S_0 \leftarrow 0$ ,  $G_0^+ \leftarrow 0$ ,  $G_0^- \leftarrow 0$ 
3: Initialize buffer  $\mathcal{B} \leftarrow []$  (rolling residual window)
4: for  $t \leftarrow 1$  to  $N$  do
5:   Add  $e_t$  to  $\mathcal{B}$  and keep only the last  $W$  values
6:   Compute rolling statistics:
7:      $\mu_t \leftarrow \text{mean}(\mathcal{B})$ 
8:      $\sigma_t \leftarrow \text{std}(\mathcal{B})$ 
9:      $m_4 \leftarrow \frac{1}{W} \sum_{i \in \mathcal{B}} (e_i - \mu_t)^4$                                  $\triangleright$  Fourth moment
10:     $\kappa_t \leftarrow \frac{m_4}{\sigma_t^4}$                                                $\triangleright$  Kurtosis
11:   Compute adaptive threshold:
12:      $h_t \leftarrow k \cdot \sigma_t \cdot (1 + \ln(\kappa_t/3))$                                  $\triangleright$  Log tail adjustment
13:   Update CUSUM statistic:
14:      $G_t^+ \leftarrow \max(0, G_{t-1}^+ + e_t - k)$ 
15:      $G_t^- \leftarrow \max(0, G_{t-1}^- - e_t - k)$ 
16:   if  $G_t^+ > h_t$  or  $G_t^- > h_t$  then
17:     Emit RegimeChangedEvent
18:      $G_t^+, G_t^- \leftarrow 0$                                                          $\triangleright$  Reset CUSUM
19:   end if
20: end for

```

- For leptokurtic distributions (heavy tails): $\kappa > 3 \Rightarrow \ln(\kappa/3) > 0$, the threshold increases proportionally, reducing false alarms during high-volatility non-Gaussian periods without structural change
- The logarithmic adjustment avoids explosive growth for extreme κ

This mechanism is consistent with the Adaptive Threshold with Kurtosis Lemma in the theory document.

2.3 Sensitivity Computation (Malliavin/AAD)

Instead of perturbing inputs (finite differences), we compute the exact derivative of the computational graph.

2.3.1 Tangential Processes and Bismut-Elworthy-Li

For a general diffusion $dX_t = b(X_t)dt + \sigma(X_t)dW_t$, the Malliavin weight formula generalizes as:

$$\partial_{X_0} E[f(X_T)] = E \left[f(X_T) \int_0^T (\sigma^{-1}(X_s) Y_s \nabla b(X_s))^{\top} dW_s \right]$$

where $Y_t = \nabla_{X_0} X_t$ is the **first variation process**, satisfying the linearized ODE:

$$dY_t = \nabla b(X_t) Y_t dt + \sum_{k=1}^d \nabla \sigma_k(X_t) Y_t dW_t^k, \quad Y_0 = I$$

We must solve the coupled system (X_t, Y_t) or use automatic differentiation (forward-mode AD) to propagate the Jacobian along the trajectory.

2.3.2 Delta-Malliavin Monte Carlo Algorithm

To compute $\Delta = \partial_{X_0} E[f(X_T)]$ in the simplified case:

$$\Delta \approx E \left[f(X_T) \frac{W_T}{\sigma X_0 T} \right]$$

In computation graphs (TensorFlow/PyTorch):

1. Define the computational graph of the payoff $L = f(X_T)$.
2. Simulate forward paths $X_0 \rightarrow X_1 \cdots \rightarrow X_T$.
3. Run the backward pass to obtain $\nabla_{X_0} L$.
4. Average $\nabla_{X_0} L$ over M paths.

Chapter 3

Numerical Solvers for Prediction Kernels

3.1 Branch A: Hilbert Projection and Wiener Filtering

3.1.1 Levinson-Durbin Recursive Algorithm

To solve the discrete Yule-Walker normal equations (the discrete equivalent of Wiener-Hopf) and obtain the optimal linear predictor of order p , $\hat{X}_{t+1} = \sum_{k=1}^p \phi_k X_{t-k+1}$: **Note:** For $O(N \log N)$ efficiency in

Algorithm 4 Levinson-Durbin Recursion

```
1: Input: Autocorrelations  $R_0, R_1, \dots, R_p$ .
2:  $E_0 \leftarrow R_0$ 
3: for  $k \leftarrow 1$  to  $p$  do
4:    $\lambda_k \leftarrow (R_k - \sum_{j=1}^{k-1} \phi_j^{(k-1)} R_{k-j})/E_{k-1}$ 
5:    $\phi_k^{(k)} \leftarrow \lambda_k$ 
6:   for  $j \leftarrow 1$  to  $k-1$  do
7:      $\phi_j^{(k)} \leftarrow \phi_j^{(k-1)} - \lambda_k \phi_{k-j}^{(k-1)}$ 
8:   end for
9:    $E_k \leftarrow E_{k-1}(1 - \lambda_k^2)$ 
10: end for
11: Output: Filter coefficients  $\phi^{(p)}$ .
```

long convolutions, use FFT (convolution theorem) instead of direct time recursion.

3.2 Branch B: HJB Equation and Viscosity Methods

3.2.1 Monotone Finite Difference Schemes

The Barles-Souganidis theorem (1991) establishes necessary conditions for convergence to viscosity solutions.

Numerical Scheme 3.1 (Generalized Upwind Scheme) *For the equation $H(u, u_x, u_{xx}) = 0$, we use:*

$$D_x^+ u_i = \frac{u_{i+1} - u_i}{\Delta x}, \quad D_x^- u_i = \frac{u_i - u_{i-1}}{\Delta x}$$
$$D_{xx} u_i = \frac{u_{i+1} - 2u_i + u_{i-1}}{(\Delta x)^2}$$

The time step is updated explicitly:

$$u_i^{n+1} = u_i^n - \Delta t \cdot H_{\text{num}}(u_i^n, D_x^+ u_i^n, D_x^- u_i^n, D_{xx} u_i^n)$$

Monotonicity Condition: The numerical Hamiltonian $H_{\text{num}}(u, p, q, r)$ must be non-decreasing in u , p , q , and r (depending on characteristic flow direction).

3.2.2 Deep Galerkin Method (DGM)

For high dimension ($d > 3$), where grids are infeasible (curse of dimensionality).

Algorithm 5 DGM Neural Network Training

- 1: **Input:** Network $f_\theta(t, x)$, PDE $\mathcal{L}u = 0$, domain Ω , steps M .
 - 2: **for** $i \leftarrow 1$ **to** M **do**
 - 3: Sample random points:
 4: $\{t_j, x_j\}_j \sim \text{Unif}([0, T] \times \Omega)$ (interior)
 5: $\{\tau_k, \xi_k\}_k \sim \text{Unif}(\{T\} \times \Omega)$ (terminal condition)
 6: $\{\zeta_l, \gamma_l\}_l \sim \text{Unif}([0, T] \times \partial\Omega)$ (boundary)
 - 7: Compute loss:
 8: $L_1 = \frac{1}{N} \sum (\partial_t f + \mathcal{L}f(t_j, x_j))^2$
 9: $L_2 = \frac{1}{K} \sum (f(T, \xi_k) - g(\xi_k))^2$
 10: $L_3 = \frac{1}{L} \sum (f(\zeta_l, \gamma_l) - h(\gamma_l))^2$
 11: $L(\theta) = L_1 + L_2 + L_3$
 12: Update $\theta \leftarrow \theta - \eta \nabla_\theta L(\theta)$ (Adam/SGD)
 - 13: **end for**
-

3.3 Branch C: Jump Integro-Differential Equation

3.3.1 Delta-Malliavin Algorithm on Poisson Spaces

For processes with jump component J_t , sensitivity is based on Malliavin integration by parts with probability weights:

$$\partial_{X_0} E[f(X_T)] \approx E \left[f(X_T) \left(\frac{W_T}{\sigma T} + \sum_{i=1}^{N_T} \frac{\partial_X \Delta X_{\tau_i}}{\Delta X_{\tau_i}} \right) \right]$$

Implementation requires tracking jump times τ_i and amplitudes ΔX_{τ_i} during the forward Monte Carlo step.

3.3.2 IMEX (Implicit-Explicit) Scheme for PIDEs

To solve the Fokker-Planck equation with integral term $\mathcal{I}[p](x) = \int p(y) \nu(dy)$:

$$\frac{p_i^{n+1} - p_i^n}{\Delta t} = \underbrace{\mathcal{L}_{\text{diff}} p_i^{n+1}}_{\text{Implicit}} + \underbrace{\mathcal{I}[p^n]_i}_{\text{Explicit}}$$

The diffusion part is solved by inverting a tridiagonal matrix (Thomas algorithm). The convolution integral is evaluated explicitly using FFT at each time step $O(N \log N)$.

3.4 Branch D: Signature Computation

3.4.1 Chen Identity and Truncation

The signature tensor $\mathbf{S}(X)_{0,t}$ up to level M lives in $T^{(M)}(\mathbb{R}^d)$. **Iterative Algorithm:** Given a discretized path with increments $\Delta X_k = X_{t_{k+1}} - X_{t_k}$: 1. Compute the signature of the linear segment $\mathbf{S}(\Delta X_k) = \exp(\Delta X_k)$ in the tensor algebra. - Level 1: ΔX_k - Level 2: $\frac{1}{2}\Delta X_k \otimes \Delta X_k$ 2. Concatenate using Chen multiplicativity:

$$\mathbf{S}(X)_{0,t_{k+1}} = \mathbf{S}(X)_{0,t_k} \otimes \mathbf{S}(\Delta X_k)$$

This tensor product is implemented efficiently by exploiting the triangular structure of tensors.

3.4.2 Log-Signatures

To reduce the feature vector dimension, we project the signature to the free Lie algebra via the Baker-Campbell-Hausdorff (BCH) formula. Recommended libraries: `iisignature` (Python/C++) or `signatory` (PyTorch, differentiable).

Chapter 4

Orchestrator: Regularized Optimal Transport

4.1 Robustness Circuit Breaker (Pre-Orchestrator)

Before Wasserstein weighting, apply strong conditional logic based on the Robustness Postulate for Singularities.

1. **Input:** SIA vector V_s and current weights w_t .
2. If $\alpha(t) < \alpha_{\text{threshold}}$ (critical roughness) or $d > 1.5$:
 - Force $w_D \leftarrow 1.0$ (Signature).
 - Switch Wasserstein cost function to Huber metric $\rho_\delta(x - y)$.
3. If **RegimeChangedEvent**:
 - Reset entropy: $w_t \leftarrow \text{Softmax}(\mathbf{0})$ (uniform).
4. **Output:** Adjusted weights to initialize Sinkhorn.

4.2 Sinkhorn-Knopp Algorithm (Dual Space)

The classic algorithm is numerically unstable for small ε . Implement via **LogSumExp** with dual potentials $f = \varepsilon \log u, g = \varepsilon \log v$.

Algorithm 6 Stabilized Sinkhorn Iterations (Log-Domain)

- 1: **Input:** Cost C , marginals a, b (in log: $\alpha = \log a, \beta = \log b$), ε .
 - 2: Initialize duals $f \leftarrow \mathbf{0}_N, g \leftarrow \mathbf{0}_N$
 - 3: **function** SMIN(M, ϵ)
 - 4: **Return** $-\epsilon \cdot \text{LogSumExp}(-M/\epsilon)$ row-wise.
 - 5: **end function**
 - 6: **while** not converged **do**
 - 7: $f \leftarrow \text{Smin}(C - g^\top, \varepsilon) + \alpha$
 - 8: $g \leftarrow \text{Smin}(C - f, \varepsilon) + \beta$
 - 9: **end while**
 - 10: Sinkhorn distance $W_\varepsilon \approx \langle \exp(f/\varepsilon), (K \odot C) \exp(g/\varepsilon) \rangle$
-

4.3 JKO Proximal Scheme

The weight update $w^{(k+1)} = \operatorname{argmin}_w \dots$ requires differentiation through the Sinkhorn loop. **Differentiable Implementation:** Use autodiff libraries (JAX/PyTorch) with `custom_vjp` (vector-Jacobian product) at the Sinkhorn fixed point, avoiding unrolling the loop to save memory:

$$\partial L / \partial C = P^* \quad (\text{Optimal Transport Plan})$$

This feeds the exact gradient $\nabla_{W_2} \mathcal{F}$ to the L-BFGS optimizer. The weight update $w^{(k)}$ is implemented as an implicit gradient step on the Wasserstein manifold:

$$w^{(k+1)} = \operatorname{Prox}_{\tau \mathcal{F}}^{W_2}(w^{(k)})$$

This is solved by nesting a Sinkhorn loop inside an L-BFGS optimizer or by projected gradient descent if entropic regularization is sufficient to smooth the energy landscape.

4.4 Dynamic Sinkhorn Regularization: Coupling to Local Volatility

Motivation: Static entropic annealing (doubling ε on failure) is robust but discrete. In highly turbulent markets, Wasserstein topology becomes rough gradually. The solution is to dynamically couple the entropic regularization parameter ε_t to local process volatility:

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

where:

- ε_0 : nominal base regularization (typically 10^{-2} or 10^{-1})
- ε_{\min} : lower bound for numerical precision (e.g., 10^{-6})
- σ_t : local realized volatility of contemporaneous prediction error
- $\alpha > 0$: sensitivity parameter (volatility-entropy coupling)

Theoretical Justification:

Under the Wasserstein flow model, the cost geometry C in the Kantorovich problem is proportional to the first variation of free energy $\delta \mathcal{F} / \delta \rho$. In high turbulence regimes:

1. The energy Hessian $\nabla^2 \mathcal{F}$ has Lipschitz constants scaling with $\|\sigma_t\|^2$ (amplified curvature).
2. The Sinkhorn operator contraction constant satisfies $\rho_{\text{contraction}} \leq 1 - c \cdot \varepsilon$ (where $c > 0$).
3. If ε is fixed and small while $\|\sigma_t\|$ is large, convergence slows exponentially.
4. Increasing ε proportionally to σ_t re-accelerates convergence without losing transport precision when volatility normalizes.

Implementation Algorithm:

Numerical Example:

Suppose $\varepsilon_0 = 0.1$, $\alpha = 0.5$, $\varepsilon_{\min} = 10^{-6}$.

- **Normal regime:** $\sigma_t = 0.02 \Rightarrow \varepsilon_t = \max(10^{-6}, 0.1 \times (1 + 0.5 \times 0.02)) = 0.101$
- **Moderate volatility:** $\sigma_t = 0.1 \Rightarrow \varepsilon_t = 0.1 \times (1 + 0.05) = 0.105$
- **Stress:** $\sigma_t = 0.5 \Rightarrow \varepsilon_t = 0.1 \times (1 + 0.25) = 0.125$
- **Crisis:** $\sigma_t = 2.0 \Rightarrow \varepsilon_t = 0.1 \times (1 + 1.0) = 0.2$ (full smoothing)

Algorithm 7 Adaptive Sinkhorn with Volatility-Based Regularization

- 1: **Input:** Cost C , marginals a, b , contemporaneous error e_t , EMA volatility σ_t
 - 2: Compute scaled volatility: $\sigma_t \leftarrow \sqrt{\text{EMA}(e_t^2, \lambda)}$
 - 3: Dynamic regularization: $\varepsilon_t \leftarrow \max(\varepsilon_{\min}, \varepsilon_0 \cdot (1 + \alpha\sigma_t))$
 - 4: Initialize duals $f, g \sim 0$
 - 5: **while** iteration < iter_max **and** not converged **do**
 - 6: $f \leftarrow \text{Smin}(C - g^\top, \varepsilon_t) + \log a$
 - 7: $g \leftarrow \text{Smin}(C - f, \varepsilon_t) + \log b$
 - 8: **end while**
 - 9: Sinkhorn distance: $W_{\varepsilon_t} = \langle \exp(f/\varepsilon_t), K_{\varepsilon_t} \exp(g/\varepsilon_t) \rangle$
 - 10: **Return** W_{ε_t}, f, g (duals for plan extraction)
-

Advantages:

1. **Continuous transition:** No discrete jumps. Sinkhorn convergence adapts smoothly to the current regime.
2. **Reduced failures:** Avoids uniform fallback (except in extreme cases) while preserving transport precision.
3. **Self-calibration:** The parameter α can be tuned via rolling validation (walk-forward) of cost vs precision.
4. **Autograd compatibility:** The dynamics $\varepsilon_t(\sigma_t)$ is differentiable, enabling end-to-end optimization of α if desired.

Suggested Parameters:

- $\varepsilon_0 \in [10^{-2}, 10^{-1}]$: Depends on cost scale; typically calibrated empirically.
- $\alpha \in [0.3, 1.0]$: Medium sensitivity. High values ($\alpha > 1$) may over-smooth; low values ($\alpha < 0.1$) reduce adaptation.
- Volatility estimator: $\sigma_t = \sqrt{\text{EMA}(e_t^2, \lambda)}$ with $\lambda \in [0.05, 0.1]$ (short memory, reactive to recent changes).

Chapter 5

Software Architecture and Parallelism

5.1 Object-Oriented Construction Patterns

The system follows SOLID principles to ensure modularity and extensibility of predictive kernels.

5.1.1 Suggested Class Structure

1. **AbstractStochasticProcess**: Base class defining the interface `simulate(dt, steps)`.
2. **ModelIdentifier (SIA)**: Singleton that consumes data streams and emits `RegimeChangedEvent`.
Uses the Strategy pattern to swap estimation methods (WTMM, DFA).
3. **PredictionKernel**: Abstract interface for predictors (A, B, C, D).
 - `fit(historical_data)`: Parameter calibration.
 - `predict(horizon)`: Future trajectory generation.
 - `compute_risk()`: VaR/ES computation.
4. **Orchestrator**: Implements the Mediator pattern. Owns a `WassersteinOptimizer` and coordinates kernel weighting.

5.2 Heterogeneous Computing and Acceleration

5.2.1 GPU (CUDA/OpenCL)

Neural network training (DGM) and large Monte Carlo simulations are delegated to the GPU.

- **Kernels**: Implement random number generation (coalesced memory access) and parallel reduction for expectation computation.
- **Sinkhorn**: Matrix operations ($K \cdot v$) are executed via optimized BLAS libraries (cuBLAS).

Implementation Note 5.1 (Shared Memory Optimization for Branch D (Signatures)) *Iterative signature computation involves tensor products of the form $\mathbf{S}_{0,t} \otimes \Delta X_k$ operating on high-dimensional tensors (d^M components for depth M). On GPU architectures, efficiency depends critically on memory hierarchy.*

CUDA Memory Management Strategy:

1. **Shared Memory (SMEM) as explicit cache:**

- *Split the discretized path into blocks of B consecutive increments*
- *Load each block $\{\Delta X_k, \Delta X_{k+1}, \dots, \Delta X_{k+B-1}\}$ into SMEM at kernel start*

- Compute the signature concatenation $\bigotimes_{i=k}^{k+B-1} \mathbf{S}(\Delta X_i)$ entirely in SMEM
- Write the partial result to global memory once per block

2. **Minimize Global <-> Shared transfers:**

- Avoid redundant reads of ΔX from global memory
- Reuse previously computed tensor components within the block
- Typical $B \in [16, 32]$ to balance occupancy and SMEM size (48-96 KB per SM depending on architecture)

3. **Coalesced access pattern:**

- Organize tensors with stride that enables warp-coalesced access
- For rank- M tensors, flatten indices in a consistent row-major or column-major order

Example Gain: For $d = 3$, $M = 4$, $B = 32$ on a V100 GPU:

- Without SMEM optimization: 15 GB/s effective bandwidth (global memory latency bound)
- With SMEM blocks: 120 GB/s (leveraging > 10 TB/s internal SMEM bandwidth)
- Speedup: 8x in signature concatenation kernel

5.2.2 FPGA (Field-Programmable Gate Array)

For ultra-low-latency applications (HFT), Branch D (Signatures) is synthesized in reconfigurable hardware.

- **Pipeline:** Iterative signature computation $S_{0,t} \otimes \Delta X$ is implemented as a systolic pipeline.
- **Fixed-Point Arithmetic:** Fixed-point arithmetic maximizes throughput after analyzing tensor dynamic ranges.

Chapter 6

Numerical Stability Considerations

6.1 CFL Condition (Courant-Friedrichs-Lowy)

For explicit finite difference schemes in the HJB equation (Branch B), the time step must satisfy:

$$\Delta t \leq \frac{(\Delta x)^2}{2 \max \sigma^2}$$

If volatility is high, the time step becomes prohibitively small. In that case, switch to an **Implicit** or **Semi-Lagrangian** scheme.

6.2 Log-Signature Stability

Log-signature computation involves the Baker-Campbell-Hausdorff series, which converges only if increments are small.

Algorithm 8 Adaptive Step Control for Signatures

```
1: Input: Path  $X$ , tolerance  $\epsilon$ .
2: function COMPUTESIG( $X$ )
3:   if  $\|\Delta X\| > \epsilon$  then
4:      $X_{\text{mid}} \leftarrow \text{Interpolate}(X)$  (midpoint)
5:      $S_1 \leftarrow \text{ComputeSig}(X_{\text{left}})$ 
6:      $S_2 \leftarrow \text{ComputeSig}(X_{\text{right}})$ 
7:     Return  $S_1 \otimes S_2$ 
8:   else
9:     Return  $\exp(\Delta X)$ 
10:  end if
11: end function
```

Chapter 7

Governance of Heuristic Metaparameters

Stochastic systems implemented on finite hardware require regularization and truncation parameters that do not exist in continuous probability theory. This chapter defines the **Control Taxonomy** to ensure numerical instantiation remains stable, reactive, and causal.

7.1 Taxonomy and Analytical Bounds (Safe Harbors)

The following mathematical limits are mandatory to avoid numerical collapse (NaNs), gradient explosions, or causal violations.

7.1.1 Discretization and Truncation Parameters

Define the resolution of the simulated world.

- **Time Step (Δt):** Not free. Must satisfy the generalized CFL condition for stochastic PIDEs.

$$\Delta t \leq \frac{C_{\text{safe}} \cdot (\Delta x)^2}{2 \cdot \sup |\sigma(x)|^2 + \sup |b(x)| \cdot \Delta x}$$

Where $C_{\text{safe}} \approx 0.9$. This is a mixed advective-diffusive CFL condition because the dynamics have both drift (advection) and volatility (diffusion) terms. Violating this limit induces spurious oscillations in the DGM/IMEX solver.

- **Signature Depth (M):** Truncation of the tensor algebra $T((\mathbb{R}^d))$ defines topological memory.
 - **Safe range:** $M \in [3, 5]$.
 - **Justification:** $M < 3$ loses non-commutativity (event ordering). $M > 5$ invokes the curse of dimensionality (feature growth as d^M), saturating RAM without marginal predictive gain.

7.1.2 Regularization and Stability Parameters

Control solution smoothness in ill-posed problems.

- **Sinkhorn Entropy (ε):** Turns hard Wasserstein transport into a smooth convex problem.
 - **Initialization:** $\varepsilon \approx 10^{-2}$.
 - **Lower bound:** $\varepsilon \geq 10^{-4}$ (for float32). Smaller values cause numerical underflow in $K = e^{-C/\varepsilon}$.

- **Impact:** $\varepsilon \rightarrow \infty$ yields uniform mixture (max uncertainty). $\varepsilon \rightarrow 0$ yields unstable winner-takes-all.
- **JKO Proximal Step** (τ): Controls the rate of change of weight distribution ρ on the Wasserstein manifold.

$$\rho_{k+1} = \text{Prox}_{\tau E}^W(\rho_k)$$

High τ allows fast but noisy adaptation. Low τ induces excessive inertia. Recommend τ adaptive and inversely proportional to prediction error volatility.

7.1.3 Decision Thresholds (Hard Boundaries)

Convert continuous probabilities into discrete actions (e.g., Circuit Breaker activation).

- **CUSUM Threshold** (h_t): Must not be a magic constant. Define dynamically with kurtosis adjustment:

$$h_t = k \cdot \sigma_{\text{resid}} \cdot (1 + \ln(\kappa_t/3))$$

where:

- σ_{resid} is the rolling standard deviation of prediction residuals
- $k \in [3, 5]$ is the base sensitivity factor (three-sigma rule)
- κ_t is kurtosis (fourth standardized moment) computed over a rolling window
- $\ln(\kappa_t/3)$ adjusts the threshold in heavy-tail regimes, reducing false positives during non-Gaussian high volatility

This adaptive threshold matches the Adaptive Threshold with Kurtosis Lemma in the theory document.

- **Singularity Tolerance** (H_{\min}): Holder exponent threshold to activate emergency mode (Signatures). Typically $H_{\min} \in [0.4, 0.5]$ to detect violent mean-reversion or market crash regimes.

7.2 Causal Cross-Validation (Walk-Forward Validation)

Static validation methods (traditional K-Fold) are prohibited as they violate the arrow of time and leak future information (look-ahead bias). The only acceptable validation scheme is rolling walk-forward with a sliding window to avoid dilution of recent regimes.

7.3 Derivative-Free Meta-Optimization (Bayesian Optimization)

Many hyperparameters are discrete (tree depth M , decision thresholds) or the error surface is noisy and non-convex, making gradient descent inapplicable.

We prescribe the use of **Tree-structured Parzen Estimator (TPE)** for efficient search of the next optimal candidate θ_{next} :

$$\theta_{\text{next}} = \arg \max_{\theta \in \Theta} \text{Expected Improvement}(\theta | \mathcal{D}_{\text{obs}})$$

The objective function is the negative return of Walk-Forward Validation ($-\mathcal{E}$). After N iterations, the estimated global optimum θ^* is the candidate that empirically minimized the error \mathcal{E} .

7.3.1 Tiered Search Space Architecture

The hyperparameter space is stratified into two optimization regimes with fundamentally different computational budgets and persistence requirements:

Algorithm 9 Strict Walk-Forward Validation Protocol (Rolling Window)

```
1: Input: Data stream  $\mathcal{D} = \{x_1, \dots, x_T\}$ , initial window  $L_{\text{train}}$ , horizon  $H$ , maximum memory  $W_{\max}$ .
2: Output: Aggregated generalization error  $\mathcal{E}$ .
3:  $t \leftarrow L_{\text{train}}$ 
4: errors  $\leftarrow []$ 
5: while  $t + H \leq T$  do
6:    $start\_idx \leftarrow \max(1, t - W_{\max})$ 
7:    $\mathcal{D}_{\text{train}} \leftarrow \{x_{start\_idx}, \dots, x_t\}$  ▷ Rolling window
8:    $\mathcal{D}_{\text{test}} \leftarrow \{x_{t+1}, \dots, x_{t+H}\}$  ▷ Immediate unknown future
9:   Training: Optimize meta-predictor ( $\theta$ ) on  $\mathcal{D}_{\text{train}}$ 
10:  Inference:  $\hat{y} \leftarrow \text{Predict}(\mathcal{D}_{\text{test}}, \theta)$ 
11:  Evaluate:  $e_t \leftarrow \text{Metric}(\hat{y}, \mathcal{D}_{\text{test}})$ 
12:  errors.append( $e_t$ )
13:   $t \leftarrow t + H$  ▷ Advance time step by step
14: end while
15: return Mean(errors)
```

Fast Tuning (Sensitivity Hyperparameters)

Objective: Rapid adaptation to regime shifts without structural topology changes.

Search Space Dimension: $|\Theta_{\text{fast}}| = 6$ parameters:

1. `cusum_k`: CUSUM drift magnitude threshold $\in [0.3, 1.5]$
2. `cusum_grace_period`: Post-alarm suppression window $\in [5, 50]$ steps
3. `sinkhorn_epsilon`: Entropic regularization $\in [10^{-4}, 10^{-1}]$ (log-uniform)
4. `ema_variance_alpha`: EWMA smoothing factor $\in [0.05, 0.5]$
5. `entropy_window`: Temporal horizon for entropy estimation $\in [20, 200]$ steps
6. `learning_rate`: JKO flow step size $\in [10^{-4}, 10^{-1}]$ (log-uniform)

Iteration Budget: $N_{\text{fast}} \approx 50$ evaluations.

Execution Time: $\sim 2\text{--}5$ hours on CPU (4 cores, walk-forward validation with $T_{\text{train}} = 5000$ samples).

Triggering Conditions:

- Deployment to a new asset class or market regime
- Significant degradation in out-of-sample performance ($> 20\%$ increase in prediction RMSE)
- Manual operator override for rapid recalibration

Persistence: Results stored in `config.toml` under `[sensitivity]` section. No intermediate state required (search completes in single session).

Deep Tuning (Structural Hyperparameters)

Objective: Optimize architectural topology and solver strategy for long-term deployment.

Search Space Dimension: $|\Theta_{\text{deep}}| \geq 20$ parameters, including:

1. **Kernel B (DGM Neural Architecture):**

- `dgm_width_size`: Network width $\in [32, 256]$ (discrete, powers of 2)

- `dgm_depth`: Number of hidden layers $\in [3, 8]$ (integer)
- `dgm_activation`: Activation function $\in \{\text{tanh}, \text{relu}, \text{swish}\}$ (categorical)
- `dgm_learning_rate`: Adam optimizer learning rate $\in [10^{-5}, 10^{-2}]$ (log-uniform)

2. Kernel C (SDE Solver Strategy):

- `stiffness_low`: Low-stiffness threshold $\in [50, 500]$
- `stiffness_high`: High-stiffness threshold $\in [500, 5000]$
- `sde_pid_rtol`: Relative tolerance for adaptive stepping $\in [10^{-6}, 10^{-3}]$ (log-uniform)
- `sde_pid_atol`: Absolute tolerance $\in [10^{-8}, 10^{-4}]$ (log-uniform)

3. Kernel A (WTMM Configuration):

- `wtmm_num_scales`: Number of wavelet scales $\in [8, 32]$ (integer)
- `wtmm_scale_min`: Minimum wavelet scale > 0
- `wtmm_sigma`: Morlet wavelet parameter $\in [0.5, 2.0]$
- `wtmm_fc`: Morlet central frequency > 0
- `wtmm_modulus_threshold`: Maxima detection threshold $\in [0.01, 0.5]$
- `wtmm_max_link_distance`: Maxima chain link distance > 0
- `wtmm_q_min`, `wtmm_q_max`, `wtmm_q_steps`: Partition function grid
- `wtmm_h_min`, `wtmm_h_max`, `wtmm_h_steps`: Singularity spectrum grid

4. Orchestrator Meta-Strategy:

- `weight_decay_rate`: Exponential decay for dormant kernels $\in [0.9, 0.999]$
- `modeCollapse_variance_threshold`: Minimum variance ratio $\in [0.1, 0.8]$
- `frozen_signal_recovery_ratio`: Threshold for stuck predictions $\in [0.5, 0.95]$

5. Global Numerical Precision:

- `signature_depth`: Truncation level $M \in [3, 5]$ (integer)
- `max_sinkhorn_iterations`: Convergence budget $\in [50, 500]$ (integer)
- `sinkhorn_inner_iterations`: Inner log-domain iterations $\in [5, 50]$ (integer)
- `numerical_epsilon`: Machine epsilon guard $\in [10^{-12}, 10^{-8}]$ (log-uniform)

Iteration Budget: $N_{\text{deep}} \approx 500$ evaluations.

Execution Time: $\sim 50\text{--}200$ hours on CPU (estimated 10–30 days of wall-clock time with interruptions).

Triggering Conditions:

- Initial system deployment (bootstrap calibration)
- Quarterly recalibration for production systems
- After major software version upgrades (e.g., JAX 0.4.20 \rightarrow 0.5.x)
- Systematic failure of Fast Tuning to restore performance

7.3.2 Persistence and Resumability Protocol

Due to the prohibitive computational cost of Deep Tuning, the optimization process must support **interruption and resumption without loss of explored search space**.

TPE State Serialization Algorithm

The Tree-structured Parzen Estimator maintains internal state consisting of:

- **Trial Database:** $\mathcal{D}_{\text{trials}} = \{(\theta_i, f(\theta_i), \text{status}_i)\}_{i=1}^N$ where $f(\theta_i)$ is the objective (walk-forward validation error) and $\text{status}_i \in \{\text{complete}, \text{failed}, \text{running}\}$.
- **Parzen Estimators:** Two kernel density estimators (KDE) for "good" and "bad" trials:

$$p_{\text{good}}(\theta) = \frac{1}{|L|} \sum_{i \in L} \mathcal{K}(\theta, \theta_i), \quad p_{\text{bad}}(\theta) = \frac{1}{|G|} \sum_{i \in G} \mathcal{K}(\theta, \theta_i)$$

where L is the set of top- γ trials (typically $\gamma = 0.2$) and G is the remainder.

- **Random Seed State:** PRNG state to ensure deterministic reproducibility.

Algorithm 10 TPE State Persistence Protocol

```

1: Serialization (Checkpoint):
2:   Input: Current TPE study object  $S$ , checkpoint path  $P_{\text{ckpt}}$ 
3:   Extract trial database:  $\mathcal{D} \leftarrow S.\text{trials}$ 
4:   Extract hyperparameter search space:  $\Theta \leftarrow S.\text{search\_space}$ 
5:   Extract best objective:  $f^* \leftarrow \min_i f(\theta_i)$ 
6:   Serialize to disk:
7:     pickle.dump( $\{\mathcal{D}, \Theta, f^*, \text{rng\_state}\}$ ,  $P_{\text{ckpt}}$ )
8:   Compute SHA-256 hash:  $h \leftarrow \text{hash}(P_{\text{ckpt}})$ 
9:   Save hash to metadata:  $P_{\text{ckpt}}.\text{sha256} \leftarrow h$ 
10:
11: Deserialization (Resume):
12:   Input: Checkpoint path  $P_{\text{ckpt}}$ 
13:   Output: Restored TPE study  $S'$ 
14:   Verify integrity:  $h_{\text{disk}} \leftarrow \text{hash}(P_{\text{ckpt}})$ 
15:   if  $h_{\text{disk}} \neq P_{\text{ckpt}}.\text{sha256}$  then
16:     Error: "Checkpoint corrupted, aborting resume"
17:   end if
18:   Deserialize:  $\{\mathcal{D}, \Theta, f^*, \text{rng\_state}\} \leftarrow \text{pickle.load}(P_{\text{ckpt}})$ 
19:   Reconstruct TPE study:  $S' \leftarrow \text{TPE}(\Theta)$ 
20:   Replay trials:  $\forall (\theta_i, f_i) \in \mathcal{D} : S'.\text{add\_trial}(\theta_i, f_i)$ 
21:   Restore RNG state:  $\text{set\_seed}(\text{rng\_state})$ 
22:   return  $S'$ 

```

Checkpoint Strategy

To balance disk I/O overhead with fault tolerance, checkpoints are emitted according to the following policy:

- **Periodic Checkpoints:** Every $\Delta N = 10$ completed trials
- **Best-Value Checkpoints:** Immediately upon discovering a new global optimum ($f(\theta_{\text{new}}) < f^*$)
- **Manual Checkpoints:** On reception of signal `SIGUSR1` (allows operator-triggered save)
- **Graceful Shutdown:** On `SIGTERM` or `SIGINT`, emit final checkpoint before termination

Checkpoint files are stored in `io/snapshots/` with naming convention:

`deep_tuning_study_<timestamp>_iter<N>.pkl`

where `<timestamp>` is ISO 8601 format and `<N>` is the iteration count.

Recovery and Validation

Upon resumption from checkpoint:

1. **Integrity Check:** Verify SHA-256 hash matches to detect disk corruption
2. **Trial Replay:** Reconstruct the TPE surrogate model by replaying all completed trials
3. **Duplicate Prevention:** Check that no trials are duplicated (same θ sampled twice)
4. **Warm Start:** Continue from iteration $N + 1$ without re-evaluating previous configurations
5. **Convergence Test:** If the improvement over the last $M_{\text{patience}} = 50$ trials is below $\delta = 10^{-4}$, declare convergence and terminate

This protocol ensures that Deep Tuning can span weeks or months of interrupted computation (e.g., system reboots, power failures, manual intervention) without catastrophic loss of optimization progress.

7.3.3 Meta-Optimization Execution Workflow

The complete optimization lifecycle integrates both tiers:

7.3.4 Objective Function Computation

The objective function for both tiers is the **mean absolute percentage error (MAPE)** aggregated over walk-forward validation folds:

$$f(\theta) = \frac{1}{K} \sum_{k=1}^K \frac{1}{H} \sum_{h=1}^H \left| \frac{\hat{X}_{t_k+h}(\theta) - X_{t_k+h}}{X_{t_k+h}} \right|$$

where K is the number of walk-forward folds, H is the prediction horizon, and $\hat{X}(\theta)$ is the predictor output under configuration θ .

Alternative objectives for specific use cases:

- **Sharpe Ratio:** For financial trading systems, maximize risk-adjusted return
- **Directional Accuracy:** For classification of trend direction (up/down)
- **99th Percentile Error:** For safety-critical applications (penalize worst-case outliers)

7.3.5 Computational Budget Management

To prevent runaway optimization costs:

- **Maximum Wall-Clock Time:** Deep Tuning terminates after 7 days regardless of iteration count
- **Early Stopping:** If no improvement over 50 consecutive trials, declare local optimum
- **Resource Limits:** Cap memory usage at 16 GB RAM per trial (prevents VRAM exhaustion on large DGM networks)

Algorithm 11 Unified Meta-Optimization Workflow

```
1: Phase 1: Deep Tuning (Structural Calibration)
2: if checkpoint exists in io/snapshots/ then
3:    $S_{\text{deep}} \leftarrow \text{Resume}(\text{checkpoint})$ 
4: else
5:    $S_{\text{deep}} \leftarrow \text{NewTPEStudy}(\Theta_{\text{deep}})$ 
6: end if
7: for  $i = 1$  to  $N_{\text{deep}}$  do
8:    $\theta_i \leftarrow S_{\text{deep}}.\text{suggest}()$ 
9:    $f_i \leftarrow \text{WalkForwardValidation}(\theta_i)$ 
10:   $S_{\text{deep}}.\text{report}(\theta_i, f_i)$ 
11:  if  $i \bmod 10 = 0$  or  $f_i < f^*$  then
12:     $\text{Checkpoint}(S_{\text{deep}})$ 
13:  end if
14: end for
15:  $\theta_{\text{deep}}^* \leftarrow \arg \min_i f_i$ 
16:  $\text{Save}(\theta_{\text{deep}}^*, \text{config.toml}[\text{structural}])$ 
17:
18: Phase 2: Fast Tuning (Sensitivity Refinement)
19: Fix structural parameters to  $\theta_{\text{deep}}^*$ 
20:  $S_{\text{fast}} \leftarrow \text{NewTPEStudy}(\Theta_{\text{fast}})$ 
21: for  $j = 1$  to  $N_{\text{fast}}$  do
22:    $\theta_j \leftarrow S_{\text{fast}}.\text{suggest}()$ 
23:    $f_j \leftarrow \text{WalkForwardValidation}(\theta_{\text{deep}}^* \cup \theta_j)$ 
24:    $S_{\text{fast}}.\text{report}(\theta_j, f_j)$ 
25: end for
26:  $\theta_{\text{fast}}^* \leftarrow \arg \min_j f_j$ 
27:  $\text{Save}(\theta_{\text{fast}}^*, \text{config.toml}[\text{sensitivity}])$ 
28:
29: Final Configuration:  $\theta^* = \theta_{\text{deep}}^* \cup \theta_{\text{fast}}^*$ 
30: return  $\theta^*$ 
```

- **Parallel Evaluation:** For systems with GPU clusters, evaluate up to 4 trials in parallel (requires synchronization of checkpoint writes)

This tiered architecture ensures that the Universal Stochastic Predictor can achieve autonomous Level 4 operation: structural topology adapts to asset characteristics via Deep Tuning (quarterly), while sensitivity parameters continuously adapt to regime shifts via Fast Tuning (on-demand).