Numerical and Algorithmic Implementation Treatise for Universal Stochastic Predictors

Adaptive Meta-Prediction Development Consortium

February 19, 2026

Contents

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

Input: $X_0, T, N, b(\cdot), \sigma(\cdot)$
$\Delta t \leftarrow T/N$
$X \leftarrow$ array of length $N+1$
for $k \leftarrow 0$ **to** $N-1$ **do**
    $Z \sim \mathcal{N}(0,1)$
    $X_{k+1} \leftarrow X_k + b(X_k)\Delta t + \sigma(X_k)\sqrt{\Delta t} Z$
end for
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 $Y_1, \ldots, Y_{N_{\text{jump}}}$.
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

Input: Time series X, scales $a_i \in \{2^0, 2^{0.1}, \ldots, 2^J\}$ (dense dyadic scales).
Step 1: CWT (FFT) and Local Maxima
For each scale a_j, extract the maxima set $M_j = \{(b, |W_{a_j}(b)|)\}$.
Step 2: Maxima Linking (Tracking)
Initialize lines $\mathcal{L} = \{(b, |W_{a_J}(b)|)\}_{b \in M_J}$ (from coarse scale).
for $j \leftarrow J - 1$ **downto** 1 **do**
    for each line $L \in \mathcal{L}$ with last point $(b_{\text{last}}, \text{mod})$ **do**
        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).
        If multiple candidates, choose the one with highest modulus.
        Extend $L \leftarrow L \cup \{(b_{\text{curr}}, \text{mod}_{\text{curr}})\}$.
    end for
end for
Step 3: Partition Function For moments $q \in [-5, 5]$:
$Z(q, a) = \sum_{L \in \mathcal{L}} (\sup_{(b,\text{mod}) \in L \cap \text{scale}(a)} \text{mod})^q$
Step 4: Exponents
$\tau(q) \leftarrow$ slope of the linear regression $\log Z(q, a)$ vs $\log a$.
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

Input: Standardized residuals e_t, base factor k, rolling window W.
$S_0 \leftarrow 0$, $G_0^+ \leftarrow 0$, $G_0^- \leftarrow 0$
Initialize buffer $\mathcal{B} \leftarrow []$ (rolling residual window)
for $t \leftarrow 1$ **to** N **do**
    Add e_t to $\mathcal{B}$ and keep only the last W values
    Compute rolling statistics:
        $\mu_t \leftarrow \text{mean}(\mathcal{B})$
        $\sigma_t \leftarrow \text{std}(\mathcal{B})$
        $m_4 \leftarrow \frac{1}{W}\sum_{i\in\mathcal{B}}(e_i - \mu_t)^4$ ▷ Fourth moment
        $\kappa_t \leftarrow \frac{m_4}{\sigma_t^4}$ ▷ Kurtosis
    Compute adaptive threshold:
        $h_t \leftarrow k \cdot \sigma_t \cdot (1 + \ln(\kappa_t/3))$ ▷ Log tail adjustment
    Update CUSUM statistic:
        $G_t^+ \leftarrow \max(0, G_{t-1}^+ + e_t - k)$
        $G_t^- \leftarrow \max(0, G_{t-1}^- - e_t - k)$
    if $G_t^+ > h_t$ **or** $G_t^- > h_t$ **then**
        Emit `RegimeChangedEvent`
        $G_t^+, G_t^- \leftarrow 0$ ▷ Reset CUSUM
    end if
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 $X_0 \to X_1 \cdots \to X_T$.
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 long convolutions, use FFT (convolution theorem) instead of direct time recursion.

Algorithm 4 Levinson-Durbin Recursion

Input: Autocorrelations $R_0, R_1, \ldots, R_p$.
$E_0 \leftarrow R_0$
for $k \leftarrow 1$ **to** p **do**
  $\lambda_k \leftarrow (R_k - \sum_{j=1}^{k-1} \phi_j^{(k-1)} R_{k-j})/E_{k-1}$
  $\phi_k^{(k)} \leftarrow \lambda_k$
  for $j \leftarrow 1$ **to** $k-1$ **do**
    $\phi_j^{(k)} \leftarrow \phi_j^{(k-1)} - \lambda_k \phi_{k-j}^{(k-1)}$
  end for
  $E_k \leftarrow E_{k-1}(1 - \lambda_k^2)$
end for
Output: Filter coefficients $\phi^{(p)}$.

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:

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

Monotonicity Condition: *The numerical Hamiltonian $H_{num}(u, p, q, r)$ 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

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

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

4.3 JKO Proximal Scheme

The weight update $w^{(k+1)} = \text{argmin}_w \dots$ 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:

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

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:

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

where:

- ε_0: nominal base regularization (typically 10^{-2} or 10^{-1})
- $\varepsilon_{\min}$: 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

Input: Cost C, marginals a, b, contemporaneous error e_t, EMA volatility σ_t
Compute scaled volatility: $\sigma_t \leftarrow \sqrt{\text{EMA}(e_t^2, \lambda)}$
Dynamic regularization: $\varepsilon_t \leftarrow \max(\varepsilon_{\min}, \varepsilon_0 \cdot (1 + \alpha\sigma_t))$
Initialize duals $f, g \sim 0$
while iteration $<$ iter_max **and** not converged **do**
    $f \leftarrow \text{Smin}(C - g^\top, \varepsilon_t) + \log a$
    $g \leftarrow \text{Smin}(C - f, \varepsilon_t) + \log b$
end while
Sinkhorn distance: $W_{\varepsilon_t} = \langle \exp(f/\varepsilon_t), K_{\varepsilon_t} \exp(g/\varepsilon_t) \rangle$
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 $\{\Delta X_k, \Delta X_{k+1}, \ldots, \Delta X_{k+B-1}\}$ 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-Lewy)

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

Input: Path X, tolerance ϵ.
function ComputeSig(X)
    if $\|\Delta X\| > \epsilon$ **then**
        $X_{\text{mid}} \leftarrow \text{Interpolate}(X)$ (midpoint)
        $S_1 \leftarrow \text{ComputeSig}(X_{\text{left}})$
        $S_2 \leftarrow \text{ComputeSig}(X_{\text{right}})$
        Return $S_1 \otimes S_2$
    else
        Return $\exp(\Delta X)$
    end if
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:** $\varepsilon \to \infty$ yields uniform mixture (max uncertainty). $\varepsilon \to 0$ yields unstable winner-takes-all.

- **JKO Proximal Step** (τ): Controls the rate of change of weight distribution ρ on the Wasserstein manifold.

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

  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 **Gaussian Processes (GP)** 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 $\mathcal{E}$.

1. **Prior:** Define safe ranges (Section 7.1) for each hyperparameter.
2. **Surrogate Model:** Train a GP on observed pairs $(\theta_i, \text{Performance}_i)$.
3. **Acquisition Function:** Select θ_{next} that maximizes the probability of improving the current best (balancing exploration vs exploitation).

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

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

4. **Costly Evaluation:** Run the full walk-forward protocol only for θ_{next}.

This approach drastically reduces computational cost compared to grid search or random search in high-dimensional spaces, converging to the predictor's optimal "personality" in a few iterations.