BEF Unified Architecture: State■Over■Time + HSSA (Hash■Shielded Streaming Accumulator)

A detailed architectural description of the BEF pipeline (BICEP → ENN■C++ → FusionAlpha) and the accompanying cryptographic accumulator for temporal traces.

Executive Summary

This document specifies the BEF pipeline from data generation to planning, then formalizes and implements a cryptographic accumulator (HSSA) that binds epoch traces and enables O(1) global checks with quantified error. We include math definitions, algorithms, encoding rules, GPU kernel design, JSON schemas, benchmarks, and verification strategies.

1. BICEP: State■Over■Time Engine

Purpose: generate temporal traces via SDE simulation with reproducible stochasticity and controlled numerical schemes.

1. Integrators: Euler–Maruyama (Itô), Heun midpoint (Stratonovich), Milstein (with derivative terms planned).
2. Noise: counter■based PRNG; Brownian increments scaled by sqrt(dt); variance reduction via antithetic pairs.
3. Models: GBM, OU, double■well; extendable to domain■specific drifts/diffusions.
4. State layout: dense vectors (nalgebra DVector), time stamps, path/ensemble containers.
5. I/O: structured exports to CSV/Parquet; for crypto, exports as flattened integer sequences (mod p) with chunk metadata.
6. Determinism knobs: seeds, dt, stride, chunk boundaries.

Dataflow: Each simulation produces a sequence of states x_t. We derive integers $v[i]$ from x_t (e.g., quantized features, recorded transitions) and partition into fixed■size chunks for downstream modules.

2. ENN■C++: Entangled Neural Network Encoder

Purpose: encode sequences/traces into latent summaries; learn committor functions, sequence tasks, or domain-specific features.

1. Cell dynamics: $\psi_{t+1} = \tanh(W_x x_t + W_h h_t + (E - \lambda I) \psi_t + b)$ with $E = L L^T$ (PSD).
2. Collapse/attention: $\alpha = \mathrm{softmax}(W_m \psi)$, $y = \alpha^T \psi$; temperature control; entropy as uncertainty.

3. Training: BPTT, Adam/AdamW, schedulers (cosine/linear), gradient clipping, regularization.
4. Validation: finite■difference grad checks (1e–10), PSD eigenvalue checks, softmax stability tests.
5. Performance: OpenMP + Eigen vectorization; zero■copy; small allocations; configurable batch/seq lengths.
6. I/O contracts: CSV/NPY readers; toy generators (parity, copy, double■well); can ingest per■chunk features (e.g., HSSA sketch_vec).

Integration: ENN can be used to learn summaries over chunks (e.g., map sketch_vec→committor or risk scores). It is optional for auditing: the cryptographic layer stands alone.

3. FusionAlpha: Graph Planning & Contradiction

Purpose: build state■time graphs and propagate priors with severity/risk controls; focus attention on contradiction■heavy regions.

1. Graph builders: HumanoidMaze (positions), AntSoccer (ball/agent), Puzzle (toggle dynamics).
2. Priors: committor/confidence from ENN; boundary conditions (goal nodes); density priors.
3. Propagation: severity■scaled diffusion/propagation steps; risk■sensitive variants; pick_next_node by argmax q.
4. Contradiction: per■node scores where local checks disagree with global trends; use to bias sampling/audits.
5. Bindings: Python + Rust (bindings crate); demos and integration tests.

Integration: HSSA per■chunk fingerprints can be node features; FusionAlpha can propose where to audit Merkle paths when contradiction is high.

4. Cryptographic Accumulator (HSSA)

Goal: bind a temporal epoch trace with a Merkle transcript and a k■dimensional polynomial fingerprint; enable O(1) global checks with explicit SZ error.

Definitions

Field p (default 2^61–1). Trace v ∈ ■^L. Chunks C_j of size B. Root_j = Merkle(SHA256(encode_u256(v[i] mod p)) over C_j). Transcript T = Root_0 || … || Root_{m–1} || encode_u256(L). Challenges r_j = SHA256(T || encode_u32(j)) mod p, j ∈ [0..k–1], r_j ≠ 0. Polynomial f(X)=Σ (v[i] mod p)·X^i. Fingerprints S_j = f(r_j) mod p.

Commit/Open/Verify

1. Commit(v): output (p,L,Roots,r■,S■) and optionally per■chunk sketch_vec.
2. Open(i): return (v[i], j, Merkle path to Root_j).
3. Verify: check Merkle path; optional audit: recompute Σ v[i]·r_j^i for a sample or entire vector; compare to S_j.

Security: Binding

If v≠v' produce same (Roots,L,S■), then either Merkle collision or δ(X)=f–f' vanishes at all r_j. Schwartz–Zippel: Pr[δ(r_j)=0 ∀j] ≤ (deg δ/p)^k ≤ (min{t,L}/p)^k. Overall: Adv_forge ≤ Adv_crh + (min{t,L}/p)^k.

5. Encoding and Chunking

Deterministic transcript: fixed B; stable hashing; big■endian encodings; contiguous root list; length as 32■byte big■endian. Last chunk shorter allowed; order bound by root ordering.

6. Verification Strategies

1. Equality: compare S■ across parties for identical transcript (O(1)).
2. Full audit: recompute S_j in O(L).
3. Partial audit: sample indices; verify Merkle paths; add contributions; combine with SZ error.

7. CUDA Engineering

1. Mod prime 2^61–1: 128■bit products, fold reduction.
2. r^i builder: anchors×base tiling; device■resident r^i vector; O(L).

3. Per■chunk dot: coalesced loads, shared memory reductions per block.
4. Multi■challenge: loop over k; reuse builder; option to fuse later.
5. Memory: r^i length L; values streamed by chunk; per■chunk outputs sketch_vec.

Benchmarks: GTX1650 L=12.5M → r^i≈44.9ms; accumulate≈33.6ms; CPU≈69.1s. A100 L=12.5–20.5M → r^i≈4–7ms; accumulate≈2–4ms; CPU≈53–92s.

8. JSON Schemas and Tools

bef_trace_v1: {schema, trace_id, field_modulus, vector_length, chunk_length, chunks:[{chunk_index, offset, values}]}

bef_sketch_v1: {schema, trace_id, field_modulus, seed, length, challenges:[…], global_sketch_vec:[…], chunks:[{chunk_index, offset, length, root_hex, sketch_vec:[…]}], timing_ms:{cuda_rpow, cuda_chunks}}

Tool: sketch_trace.py takes bef_trace_v1 and writes bef_sketch_v1 with chosen (p,k).

9. Threat and Update Model

Adversary may attempt (a) change v but keep T (Merkle collision), or (b) change both v and T and hope sketches collide (SZ). Epoch discipline: freeze T per batch; derive fresh r■; do not reuse r across epochs; cache r^i within epoch.

10. Comparison Axes (Plain Terms)

1. Merkle tries: O(log N) index proofs; no O(1) global check; HSSA adds S■.
2. Vector commitments: succinct per■index; HSSA offers global SZ checks under CRH/RO (no setup).
3. Polynomial commitments: evaluation proofs; HSSA publishes evaluations for equality/audits (no setup).
4. Streaming fingerprints: unauthenticated; HSSA authenticates via transcript.

11. Colab/Runtime Notes

Use GPU runtime; install ninja; clear Torch extension cache if needed. demo_cuda(num_chunks, chunk_len, seed, num_challenges). sketch_trace.py --modulus p --num-challenges k. Recommended $p=2^{61}-1$, $k=4$.

12. Roadmap

1. Formalize partial audit detection probabilities (sampling + SZ).
2. Adaptive/multi■epoch analysis in QROM.
3. Fused multi■r kernel for higher k.
4. Alternative fast mod primes (e.g., $2^{64}-2^{32}+1$), quantify constants.

This document unifies system and cryptography views with explicit interfaces, parameters, and performance, enabling reproducible evaluation and fair comparison.