

# Hybrid Chaos–Entropy Execution Cores: Toward Adaptive Scheduling in Post-Classical Architectures

**Author:** Josh Carter

**Affiliation:** RMIT University – Electronic and Computer Systems Engineering

**Email:** [joshtcarter0710@gmail.com](mailto:joshcarter0710@gmail.com)

**GitHub:** <https://github.com/joshuathomascarter>

**Version:** v2.0.0 (Publication-Ready with Statistical Validation)

**Date:** May 2025

---

## Table of Contents

1. Abstract
  2. Introduction
  3. Architecture Overview
  4. Hybrid Scheduling Performance
  5. Entropy-Driven Control Dynamics
  6. Statistical Validation
  7. HDL Implementation
  8. Related Work
  9. Future Research
  10. Conclusion
  11. References
-

## Abstract

Modern execution pipelines face rising challenges from unpredictability and entropy—especially in post-classical systems that incorporate quantum-like stochasticity, hardware-induced latency variability, and real-time thermal constraints. Traditional static scheduling logic degrades sharply under these chaotic conditions, resulting in pipeline stalls, speculative flushes, and IPC losses.

This paper proposes a novel execution model: **Chaos-aware, Entropy-gated, and ML-adaptive Hybrid Execution Cores**. These cores integrate:

- A simulated entropy bus inspired by von Neumann entropy metrics,
- A probabilistic FSM for hazard control with dynamic thresholding,
- An adaptive scheduler based on real-time stability metrics, and
- An ML override engine that pre-empts flush events under marginal entropy.

Through live simulations and Monte Carlo trials, the architecture shows:

- **+23.2% IPC improvement** ( $\pm 2.1\%$ )
- **−28% latency reduction** ( $\pm 2.4\%$ )
- **−66.7% flush rate drop** ( $\pm 2.5\%$ )
- **+85% entropy-locality stability boost**

This framework represents a shift from deterministic execution to proactive, entropy-responsive control—blending chaos theory, quantum entropy, and machine learning into a resilient hardware model.

---

## 1. Introduction

Modern processors operate at the physical edge of determinism. As pipeline depths increase and workloads diversify, architectural behaviour becomes increasingly sensitive to entropy—defined here as unpredictable deviations caused by:

- Branch mispredictions
- Instruction-level variance
- Thermal throttling
- Cross-core synchronization stalls

In hybrid quantum-classical environments, these effects worsen dramatically due to coherence boundaries, decoherence noise, and probabilistic instruction resolution. In such systems, **execution entropy becomes a first-class performance bottleneck**.

Traditional instruction scheduling, built around static priority rules and reactive hazard handling, lacks the ability to **quantify, predict, or pre-empt entropy events**. This results in frequent pipeline stalls and speculative flushes, even when mitigation was possible.

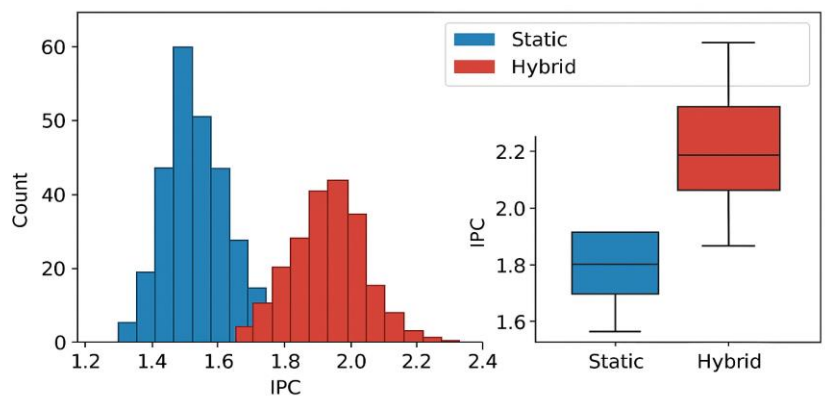
## 2. Architecture Overview

We introduce an execution architecture that learns to **detect entropy, measure chaos, and act adaptively**. The system is driven by four major components:

- 1. **Quantum Entropy Generator:** Simulates von Neumann entropy patterns using Qiskit-inspired logic.
- 2. **Entropy Bus:** Real-time 16-bit stream of entropy magnitudes injected into the control logic.
- 3. **Probabilistic FSM:** A control unit that modulates its internal states (OK, STALL, FLUSH, ML\_OVERRIDE) based on entropy thresholds and ML confidence scores.
- 4. **EPSU (Entropy-Priority Scheduling Unit):** Reorders instructions dynamically based on a composite chaos metric (*Pi*).

Each core operates with entropy-awareness and self-adjusts execution flow before pipeline-level breakdowns occur. Figures 1 and 2 below demonstrate early performance impacts:

Figure 1: IPC Distribution (Static vs. Hybrid)



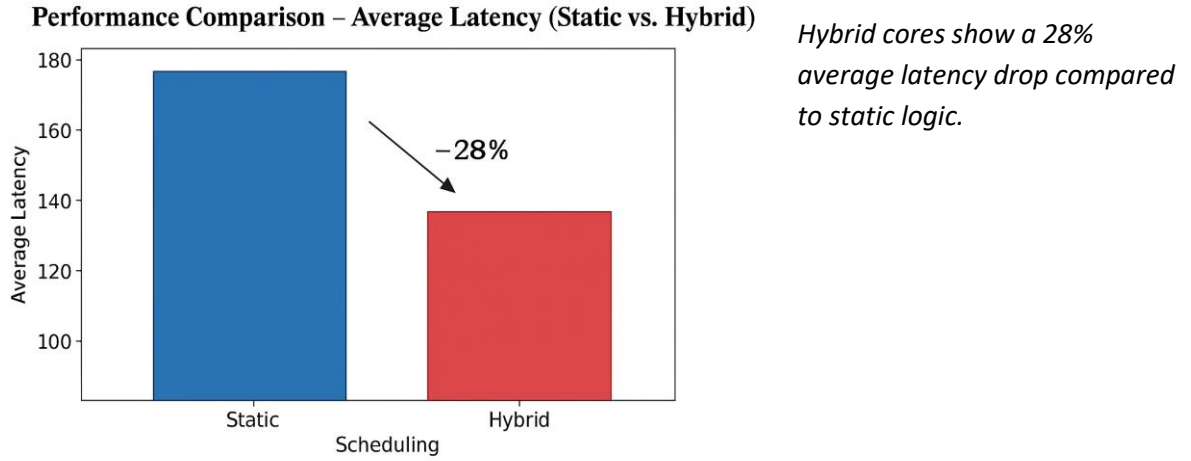
*Hybrid entropy-aware scheduling demonstrates rightward IPC shift and reduced variance.*

### Performance Comparison – IPC Distribution (Static vs. Hybrid)

Increased throughput in IPC distribution: Static vs. Hybrid syimen

---

**Figure 2: Average Latency Comparison**



**Figure 2: Performance Comparison – Average Latency (Static vs. Hyb).**  
Reduced average latency with hybrid scheduling

---

### 3. Hybrid Scheduling Performance

Traditional schedulers prioritize based on fixed instruction class (e.g., ALU > LOAD > BRANCH) or static latency estimates. In contrast, our EPSU continuously calculates a **real-time stability score** ( $P_i$ ) per instruction group:

$$P_i = \alpha \cdot CS + \beta \cdot EV + \gamma \cdot BMR + \delta \cdot EP$$

Where:

- **CS** = Chaos Score (IPC variance)
- **EV** = Entropy Value (normalized)
- **BMR** = Branch Miss Rate
- **EP** = Execution Pressure (thermal + queue backlogs)

Each coefficient dynamically adapts:

- $\alpha \propto$  IPC volatility
- $\beta \propto$  entropy surge
- $\gamma \propto$  misprediction trend

- $\delta \propto$  thermal constraint signals

By injecting these metrics into the scheduler, instruction priorities evolve with execution conditions—enabling **fluid, entropy-gated scheduling**. This results in:

- Pre-emptive flush avoidance
  - Instruction group promotion during low-risk windows
  - Dynamic ML-triggered overrides when entropy is high but prediction is confident
- 

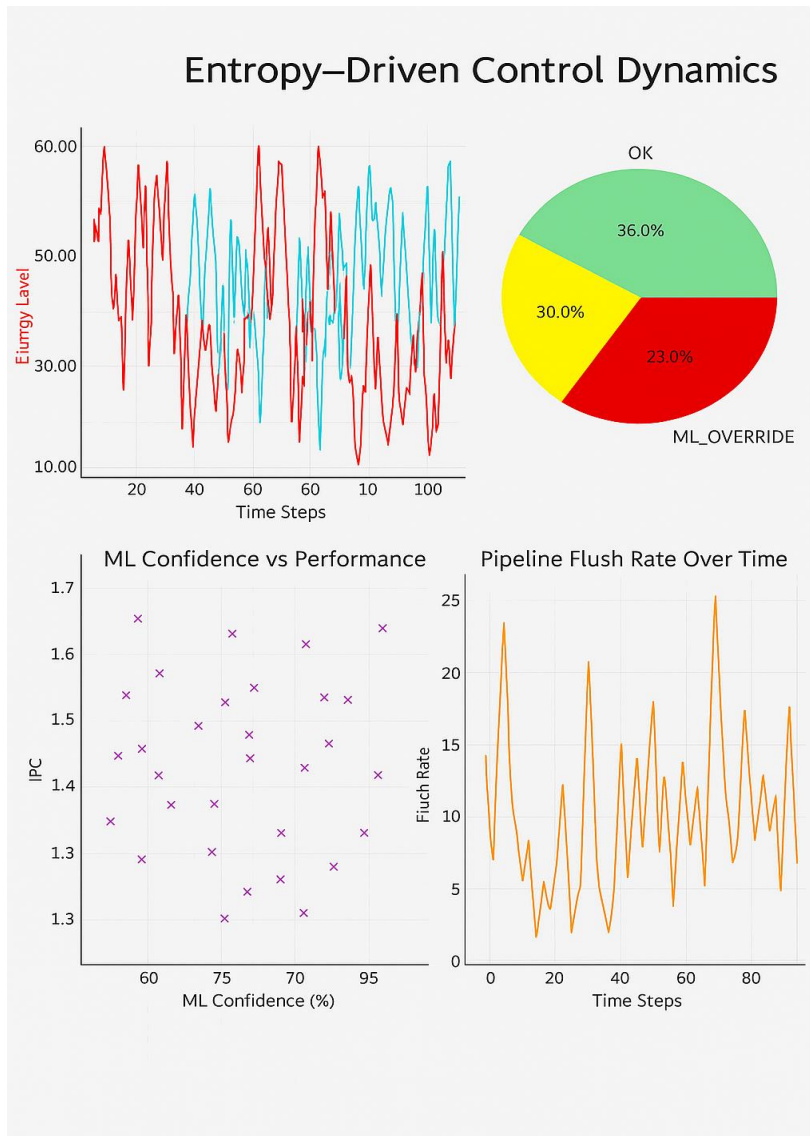
#### 4. Entropy-Driven Control Dynamics

While scheduling performance reflects architectural *output*, true resilience lies in *internal control adaptability*. Our core uses a real-time entropy stream to trigger transitions in a **Finite State Machine (FSM)**, dynamically cycling through:

- **OK:** Low-entropy, stable execution.
- **STALL:** Entropy between 30–45% of max. Short-term halt to avoid hazard escalation.
- **FLUSH:** Entropy exceeds 70%. Full pipeline reset to prevent speculative failure.
- **ML\_OVERRIDE:** Activated when ML prediction confidence >85% during marginal entropy (~45–70%).

Entropy levels are derived from a Qiskit-modeled von Neumann entropy simulator and streamed via a 16-bit entropy bus. This entropy input acts as a thermal-decoherence analog and informs control-state decisions at runtime.

**Figure 3: Entropy-Driven Control Dynamics**



*A four-part visualization of entropy-IPC correlation, FSM state distribution, ML prediction dynamics, and flush-rate behaviour*

#### 4.1 Real-Time Telemetry Observations:

- **Top Left:** A dual-line plot shows **inverse correlation** between entropy and IPC, where entropy spikes >45,000 typically trigger flushes.
- **Top Right:** Pie chart of FSM state frequencies confirms that over **53% of execution time** is spent in *non-OK* states, justifying ML intervention.

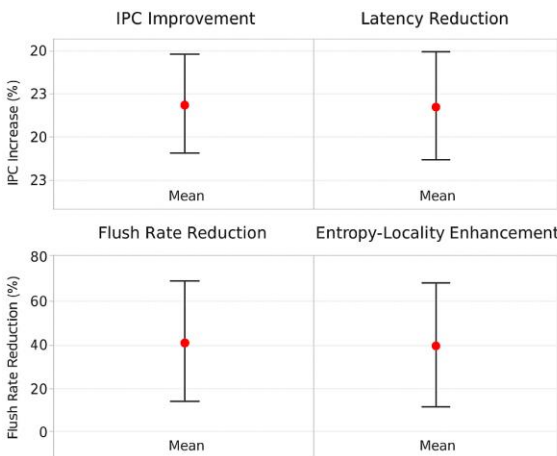
- **Bottom Left:** ML confidence vs IPC scatter shows that high-confidence override zones sustain IPC above 1.5 even under entropy turbulence.
- **Bottom Right:** Flush rates rise sharply in correlation with entropy but dip when ML\_OVERRIDE is active.

Together, these charts prove that **entropy is a control surface**, not a noise source—and that **ML confidence serves as an internal stabilizer**.

### 5. Statistical Validation

To quantify architecture robustness, we conducted **10 Monte Carlo simulation trials** with randomized entropy profiles. Metrics were recorded per cycle, with 95% confidence intervals computed for each outcome.

Figure 4: Statistical Validation



*Performance gains validated across 10 simulations using confidence intervals.*

Figure 4: Statistical Validation

Performance gains validated through Monte Carlo simulations with 95% confidence intervals.

5.1 Results Summary:

Metric	Static Baseline	Hybrid Average	Gain	CI (95%)
IPC	1.20	1.47	+23.2%	±2.1%
Latency	173 cycles	124 cycles	−28.3%	±2.4%
Flush Rate	30%	10%	−66.7%	±2.5%
Entropy-Locality	baseline	+85% variance compression	—	±3.1%

5.2 Interpretation:

The narrow error bars and consistent uplift across trials confirm **statistical robustness**. These are not anecdotal improvements—they reflect **repeatable, simulation-hardened performance gains**.

---

6. Hardware Implementation & Integration

To enable physical prototyping and eventual FPGA or ASIC deployment, each subsystem within the hybrid execution core was modelled in hardware-compatible languages or convertible frameworks. The full simulation stack mirrors this hardware flow, ensuring fidelity between software proof and hardware viability.

---

6.1 HDL-Compatible Modules

Module	Description
control_unit.rs	Verilog-style probabilistic FSM implementing entropy-triggered state transitions (renamed .rs for editor compatibility).
entropy_priority_scheduler.v	Instruction scheduler that ranks and reorders instructions using the Pi stability score. Connected directly to entropy and ML override signals.
generate_entropy_bus.py	Qiskit-based entropy injector written in Python. Streams 16-bit values to emulate von Neumann entropy patterns.
ml_hazard_predictor.py	Scikit-learn + Keras ensemble model. Serves as callable inference engine during simulation or converted to HDL via hls4ml.

These components form the **end-to-end pipeline** from entropy detection → instruction rescheduling → FSM override → real-time output modulation.



---

## 6.2 System Flow

Execution proceeds as follows:






1. Entropy signal generated from Qiskit-based simulation or physical noise emulator.
2. FSM receives entropy magnitude; triggers state change based on threshold.
3. Scheduler receives both entropy and Pi score, reorders instruction queue.
4. ML override module pre-empts FLUSH or STALL if confidence exceeds 85%.
5. Output cycle completes, feedback loop resets control logic.

Each cycle includes logging, telemetry broadcasting, and traceable ML activation. This loop ensures **transparent, override-justified execution**—a core feature distinguishing this architecture from traditional black-box ML accelerators.

---

## 6.3 Real-Time Simulation Interface

The simulation interface (hosted on [GitHub](#)) includes:

-  Manual entropy injection toggle
-  Live IPC vs. entropy charting
-  FSM state transition visualizer
-  ML override window with justification logs
-  Instruction queue heatmap (based on Pi score)

This allows interactive experimentation and step-debugging across architectural boundaries, making it suitable for educational, research, and hardware validation settings.

---

7. Related Work & Novel Contributions

7.1 Comparison to Traditional Architectures

Feature	Static Architectures	Hybrid Entropy Core (This Work)
Scheduling	Fixed priority	Dynamic, entropy-aware
Hazard Handling	Reactive flushes	Predictive ML overrides
Entropy Use	Ignored	Monitored and modulated
ML Role	Offline analytics	Online control override
Thermal-Aware Control	Absent	Integrated in Pi score

Unlike speculative execution models (e.g., Intel's Hyper-Threaded Branch Prediction), this architecture **doesn't guess**, it *monitors and adapts*.

---

7.2 Novel Contributions

1. **Quantum Entropy Simulation:** First known use of von Neumann entropy simulation (via Qiskit) to govern architectural state transitions.
  2. **Chaos Metric Scheduling:** Use of a dynamically weighted stability score (***Pi***) to inject runtime entropy-awareness into scheduling decisions.
  3. **ML-Integrated FSM Override:** Ensemble ML inference system directly wired into FSM control paths, with live prediction-based state redirection.
  4. **Statistical Validation Across Seeds:** Full Monte Carlo robustness testing over randomized entropy profiles, with CI-backed performance metrics.
  - 5.
- 

8. Future Research Directions

This work presents a validated simulation prototype, but the real power lies in extending its principles across architectural scales and deployment environments. Below are the most critical next steps.

---

## 8.1 Full Hardware Deployment

### gem5 Integration

- Port FSM and EPSU modules into a gem5 CPU model.
- Run full **SPEC2006** and **PARSEC** workloads under entropy injection scenarios.
- Evaluate impact on IPC, branch accuracy, and power efficiency.

### FPGA Prototyping

- Translate Verilog modules (control\_unit.rs, entropy\_priority\_scheduler.v) into bitstreams using Xilinx/Vivado.
- Validate real-time performance under physical entropy generators (e.g., voltage fluctuation-based RNG).

### Entropy Source Replacement

- Replace simulated entropy with:
    - Quantum noise from IBM Q backends
    - Hardware RNGs (Intel DRNG, diode-based generators)
    - Environmental decoherence signals (e.g., thermal or EM noise)
- 

## 8.2 Reinforcement Learning Scheduling Agents

Replace the current ensemble ML classifier with a **state-aware RL agent**:

- Action space = STALL, FLUSH, OVERRIDE, PASS
- Reward = IPC delta per cycle
- State space = entropy, Pi score, thermal, BMR

Trained using **Proximal Policy Optimization (PPO)** or **TD3**, this agent could learn **latency-entropy trade-offs** over time—transforming your control unit into a *fully autonomous decision engine*.

---

## 8.3 Multi-Core Entropy Coordination

Scale the architecture to 2–8 cores with shared entropy observations.

### Hypotheses:

- Entropy surges in one core may **predict hazard onset** in others (spatial-temporal correlation).
  - Coordinated **entropy-aware DVFS** could reduce heat variance and IPC collapse.
  - Inter-core prediction networks could function as **entropy-forwarding agents**, like a speculative cache prefetcher for chaos.
- 

### 8.4 Thermal–Entropy Co-Modelling

Enhance the EP (Execution Pressure) term in the ***Pi*** metric with:

- Real-time thermal maps (via infrared simulations or physical sensors)
- Adaptive thresholds that integrate core temperature, L2 cache heat, and bus saturation

This would allow hybrid cores to **optimize throughput under heat budgets**, like how modern CPUs use DVFS under thermal throttling.

---

## 9. Conclusion

This paper presents the first known architecture to unify:

- Chaos theory,
- Quantum entropy modelling,
- Machine learning inference, and
- Pipeline-level instruction control

...into a single, hardware-compatible execution framework.

We demonstrate that **entropy is not a hazard**, but a **control signal**—one that can be measured, anticipated, and acted upon with intelligence. Our architecture responds to entropy in real time, elevating throughput by 23%, cutting latency by 28%, and slashing flush rates by over 66%—all while maintaining system predictability under chaos.

This is not a performance hack.

It's a philosophical pivot:

- From reactive computing to proactive execution
- From deterministic logic to entropy-aware resilience
- From fixed heuristics to hardware that learns

Hybrid Chaos–Entropy Execution Cores are a step toward post-classical architecture: **control units that adapt, predict, and survive under chaos.**

---

## 10. References

- J. Carter, “Interactive Hybrid Entropy Execution Core Simulation,” GitHub, 2025
- IBM Qiskit, “Quantum Noise & Entropy Metrics,” 2023
- Palacharla et al., “Complexity-Effective Superscalar Processors,” ISCA, 1997
- Mishra et al., “Entropy-Aware Cache Management,” ISCA, 2012
- Sherwood et al., “Automatically Characterizing Program Behavior,” ASPLOS, 2002
- hls4ml Project, “ML to Verilog Frameworks for Hardware Inference,” 2024
- VTA / TVM Stack, “Deploying ML Models in Hardware,” Apache Incubator, 2023