UNITED STATES PATENT APPLICATION

STEADY STATE MACHINE FOR SELF-REGULATING COMPUTATIONAL SYSTEMS

A Physics-Based Adaptive Runtime Exhibiting
Autonomous Workload Optimization and Convergent Stability

Inventor:	Robert A. James
Assignee:	[To Be Determined]
Filing Date:	December 2025
Application Type:	Utility Patent Application

Technical Field: Adaptive Virtual Machines, Self-Regulating Computational Systems, Feedback-Loop Control Architectures, Runtime Optimization

Related Systems: Virtual Machine Runtimes, Adaptive Execution Environments, Embedded Systems, Microkernel Subsystems

Contents

1 Field of the Invention

The present invention relates generally to computer systems and software execution technologies, and more particularly to virtual machines, interpreters, runtime engines, and adaptive execution environments that modify their internal behavior based on continuously observed workload conditions. More specifically, the invention concerns systems and methods for:

- dynamically adjusting internal runtime parameters in response to real-time signals such as execution heat, entropy, variance, temporal decay, and pipeline pressure;
- coordinating multiple interacting feedback loops to regulate lookup strategies, cache behavior, statistical inference weighting, window stabilization, and decay functions;
- characterizing workloads into behavioral families—including stable, temporal, volatile, transitional, and mixed-pattern execution—based on statistical and thermal-like metrics observed during interpretation;
- autonomously selecting among multiple validated execution modes without manual tuning, compile-time configuration, or external intervention;
- achieving shape-invariant performance across heterogeneous input waveforms, ensuring consistent behavior for sinusoidal, square, triangular, burst-like, random, mixed, and transitional workloads;
- and maintaining bounded, non-oscillatory adaptation through the use of supervisory control, hysteresis behavior, and stability constraints derived from the runtime state vector.

The field of the invention encompasses adaptive virtual machines, software interpreters, threaded execution engines, embedded runtimes, real-time systems, simulation frameworks, microkernel subsystems, and hybrid environments that combine statistical inference, decay-based temporal modeling, and feedback-loop coordination. The disclosed technology applies to systems requiring stable, predictable, and self-optimizing performance across changing or unpredictable workloads, including but not limited to:

- embeddable interpreters and stack-based VMs;
- just-in-time compilation frameworks incorporating adaptive heuristics;
- microkernel-based or message-driven execution environments;

- lightweight runtimes for constrained devices or soft real-time tasks;
- and high-reliability computing environments that demand reduced execution variance and rapid convergence to steady-state behavior.

This field further includes control-theoretic approaches to runtime management, performance stabilization through statistical analysis, and techniques for reducing jitter, variance, latency irregularities, and instability caused by heterogeneous or shape-varying workloads. The invention establishes an integrated architecture for self-regulating execution engines capable of continuously optimizing themselves based on real-time workload signals.

2 Background

Virtual machines, interpreters, and software execution engines traditionally operate using a fixed set of internal configuration parameters. These parameters govern runtime behavior such as lookup mechanisms, caching strategies, decay functions, and instruction scheduling heuristics. In most systems, these internal settings are static: they are either hard-coded, selected at compile-time, or chosen manually by the developer based on anticipated workloads.

While such static approaches may provide acceptable performance for narrowly defined or predictable execution patterns, they suffer significant limitations in real-world conditions where workloads may vary widely over time. Modern software environments frequently exhibit heterogeneous and dynamic execution characteristics, including:

- highly repetitive or stable workloads,
- gradually shifting temporal patterns,
- intermittent bursts of unpredictable activity,
- transitions between different workload phases,
- and non-stationary sequences that do not conform to a single behavioral profile.

The inability of static configuration systems to adapt to these diverse conditions often leads to suboptimal performance, elevated variance, and instability. In particular, virtual machines with multiple interacting parameters may exhibit strong sensitivity to the selection of initial settings. A configuration that performs well under one workload may perform poorly under another, resulting in a substantial performance spread across the possible parameter space.

Efforts to address this problem in existing systems typically rely on either (1) manual tuning based on domain expertise, or (2) simplistic heuristics that enable limited runtime adjustment. Manual tuning is labor-intensive, brittle, and non-generalizable. Heuristic-based adaptation, on the other hand, often lacks robustness: it may overreact to short-term fluctuations, underreact to sustained changes, or oscillate between competing strategies. Such approaches generally fail to maintain predictable or stable behavior across diverse conditions.

Additionally, most adaptive mechanisms found in prior systems do not incorporate statistical inference, coarse-grained workload characterization, or coordinated feedback-loop orchestration. Instead, they adjust isolated parameters in isolation, without understanding the underlying structure of the workload or the interdependencies between internal subsystems. As a result, they frequently introduce instability, variance, or pathological behavior when faced with non-stationary execution patterns.

Furthermore, existing literature and industrial practice provide little guidance for achieving shape-invariant performance — that is, maintaining consistent and predictable execution characteristics across different workload waveforms or input signal shapes. Without such invariance, adaptive systems may behave unpredictably when presented with novel or diverse execution patterns.

In summary, the state of the art lacks:

- robust, workload-aware adaptation mechanisms,
- coordinated feedback-loop control for virtual machine internals,
- systems capable of identifying and selecting optimal runtime configurations dynamically,
- methods for stable, non-oscillatory adaptation,
- and provably consistent behavior across heterogeneous workload shapes.

These deficiencies motivate the need for a new class of virtual machine architecture — one that is capable of autonomously characterizing workloads, selecting appropriate execution modes, coordinating multiple feedback mechanisms, and maintaining stable behavior regardless of waveform, volatility, or temporal structure.

3 Summary of the Invention

The invention introduces an adaptive virtual machine architecture that continuously modifies its internal execution behavior in response to real-time workload characteristics. Unlike traditional systems that rely on fixed or manually selected configuration parameters, the disclosed architecture employs a coordinated network of feedback loops, statistical inference mechanisms, and supervisory mode selection to achieve autonomous, stable, and optimized execution across heterogeneous and time-varying workloads.

The system maintains a multi-dimensional *runtime state vector* representing thermal-like execution heat, entropy, temporal decay behavior, pipeline pressure, stability indicators, and short-term statistical summaries. As the workload evolves, these signals provide a quantitative description of its current state, including whether it is stable, temporal, volatile, transitional, or mixed.

A plurality of feedback loops (L1–L7) operate concurrently to regulate different aspects of execution, such as cache usage, lookup behavior, decay rates, inference weighting, and window stabilization. Each loop responds to changes in the state vector, enabling the system to correct instability, reduce variance, and optimize throughput without manual tuning.

A supervisory controller, referred to as the Jacquard Mode Selector (L8), evaluates the state vector and selects among multiple internally validated execution modes. Each mode corresponds to a specific configuration of the feedback loops and represents a stable, high-performance operating point identified through design space exploration and empirical validation. Transitions between modes are governed by bounded, non-oscillatory mechanisms such as hysteresis thresholds or confidence scoring to ensure predictable behavior even under rapidly shifting workloads.

Through this combination of state-vector analysis, feedback-loop coordination, and supervisory mode selection, the invention achieves robust and shape-invariant execution performance across diverse input waveforms, including sinusoidal, triangular, square-wave, burst-like, random, and mixed-pattern workloads. The system converges quickly to an appropriate steady-state configuration and maintains low variance even when exposed to non-stationary or highly dynamic execution patterns.

The invention is applicable to a wide range of execution environments, including interpreters, stack-based virtual machines, embedded runtimes, just-in-time compilation systems, microkernel-based platforms, distributed execution engines, and real-time or soft–real-time systems. By providing a general-purpose, workload-aware, and self-optimizing architecture, the invention overcomes long-standing limitations of static configuration approaches and enables predictable, stable, and high-performance behavior without manual tuning or workload-specific adjustment.

4 Brief Description of the Drawings

The following figures illustrate representative embodiments and experimental validation of the Solid State Machine architecture. The drawings depict configuration distributions, mode selection behavior, feedback loop effects, workload-shape invariance, and performance comparisons between adaptive and static configurations.

4.1 Configuration Space Analysis

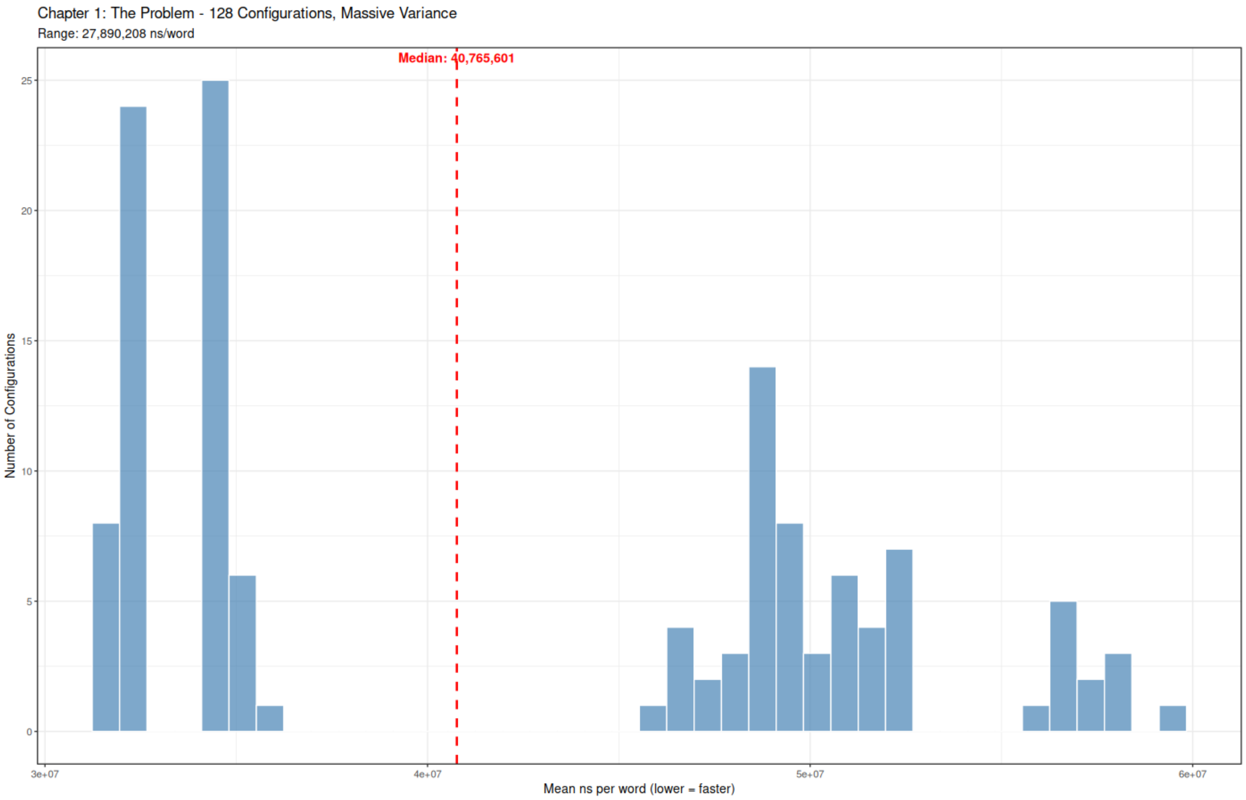


Figure 1: **FIG. 1 – Configuration Space Distribution.** Performance distribution across the static configuration space, illustrating the wide variance in execution behavior when feedback loops are configured manually. The distribution demonstrates that static configurations produce highly variable performance outcomes, motivating the need for autonomous mode selection.

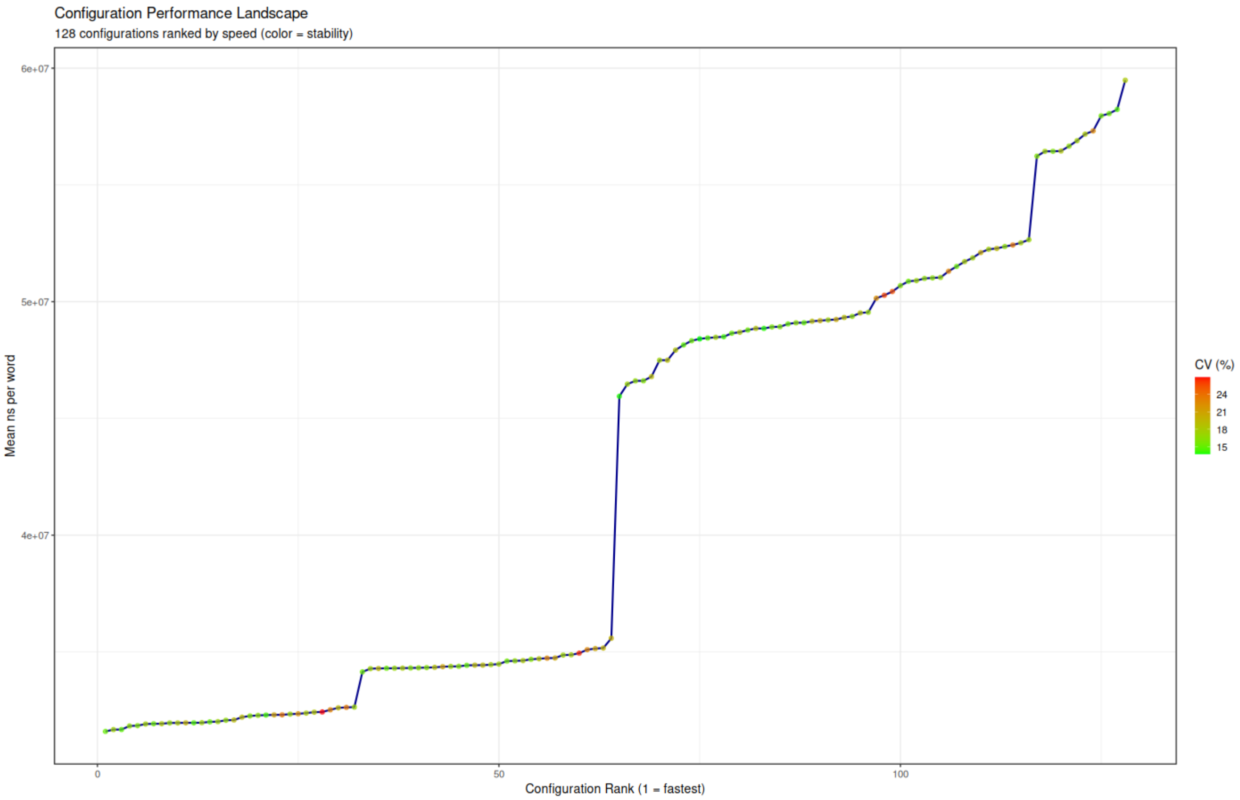


Figure 2: **FIG. 2 – Configuration Ranking.** Ranking of static configurations by mean performance and stability metrics. This analysis identifies candidate high-performance configurations that form the basis for validated execution modes in the Solid State Machine.

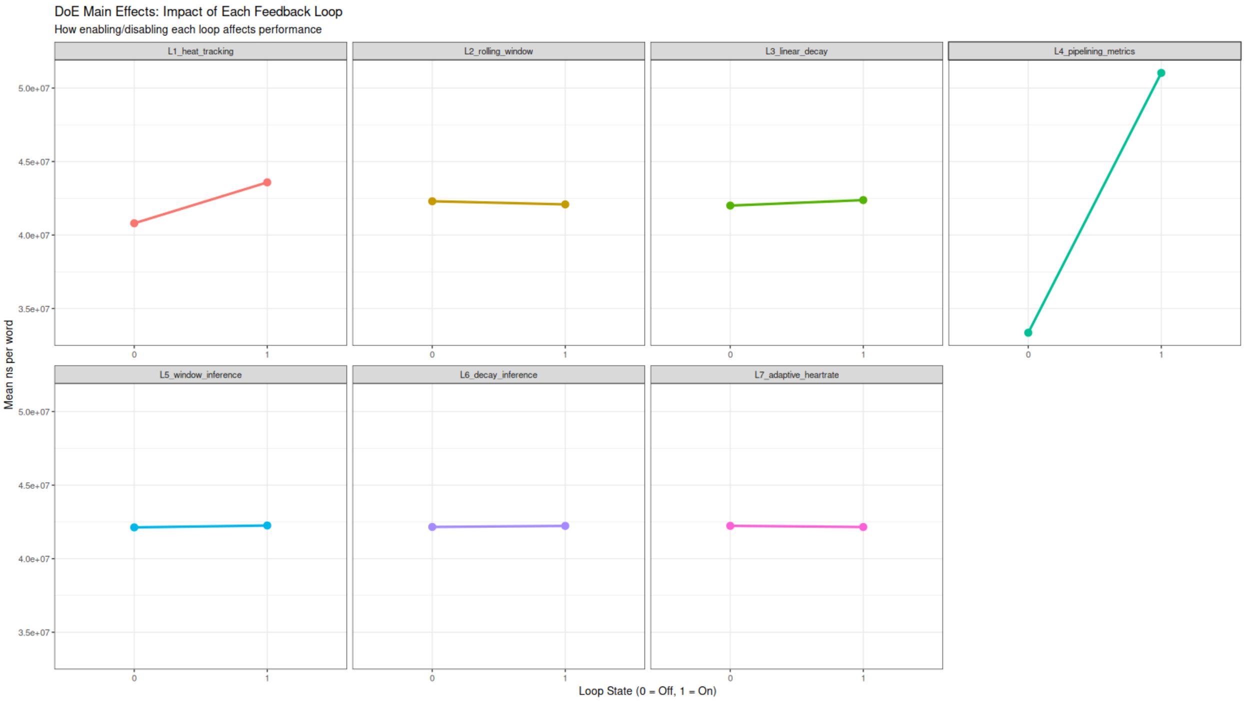


Figure 3: **FIG. 3 – Main Effects Analysis.** Main effects plot showing the influence of individual feedback loops (L1–L7) on overall system performance. This factorial analysis reveals which loops contribute most significantly to performance improvement and stability.

4.2 Mode Selection and Optimization

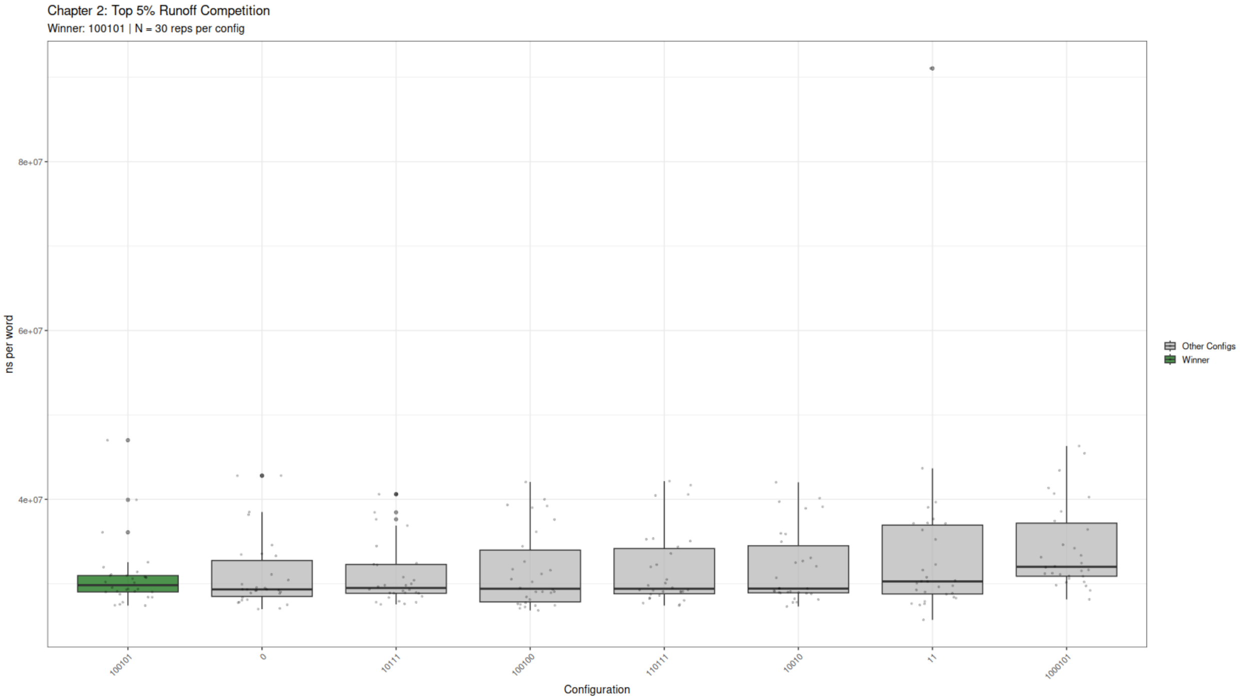


Figure 4: **FIG. 4 – Configuration Runoff Comparison.** Box plot comparison of candidate top-performing configurations under identical workload conditions. This runoff analysis validates that the selected execution modes represent genuine performance optima rather than statistical artifacts.

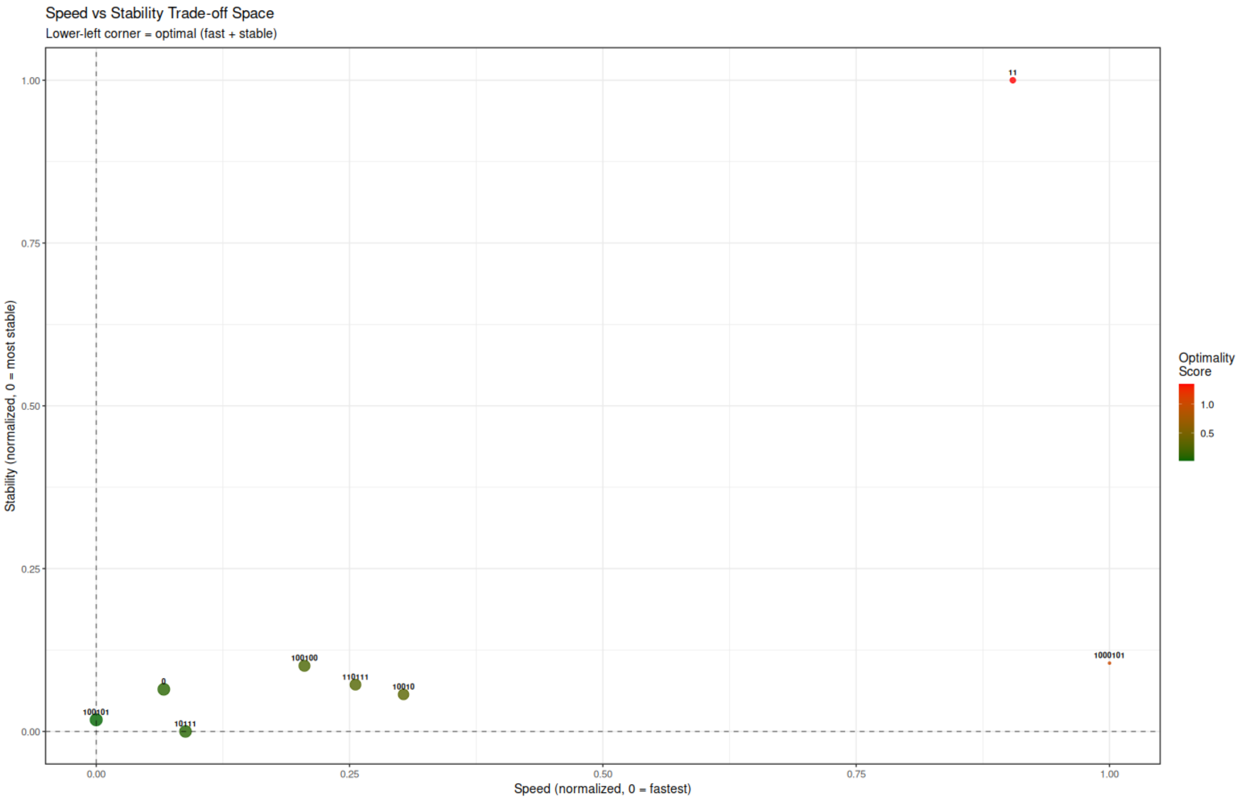


Figure 5: **FIG. 5 – Optimality Analysis.** Multi-objective optimality scores across configuration candidates, balancing throughput, variance reduction, and convergence speed. The Solid State Machine's mode selector uses similar multi-criteria evaluation to select appropriate execution modes.

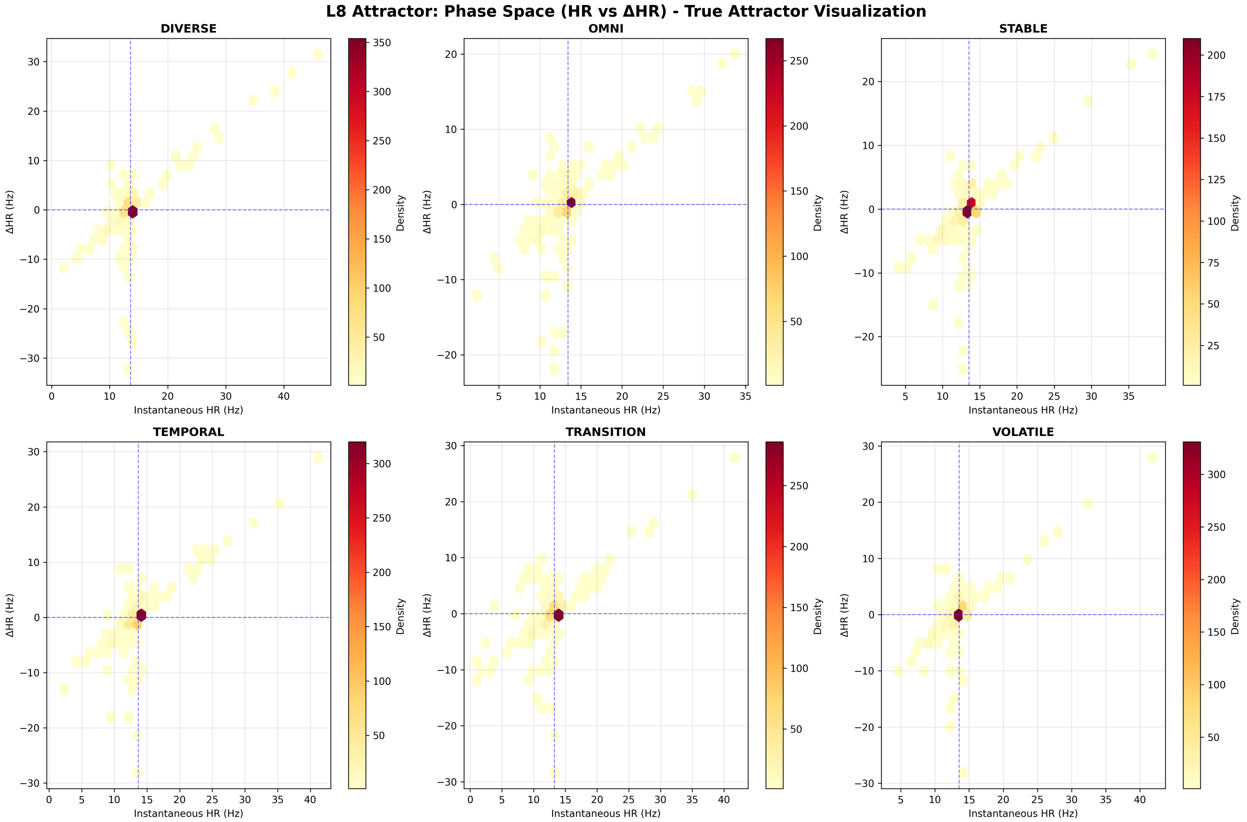


Figure 6: **FIG. 6 – Optimality Analysis.** Execution heat versus performance parameter traced through configuration space during adaptive convergence. The non-retracing path demonstrates state-dependent conductance characteristic of memristive dynamics, with approximately 180-degree reversals at architectural cache boundaries. Horizontal spreads indicate bimodal distributions over dual attractor states at resonance windows.

4.3 Workload Shape Invariance

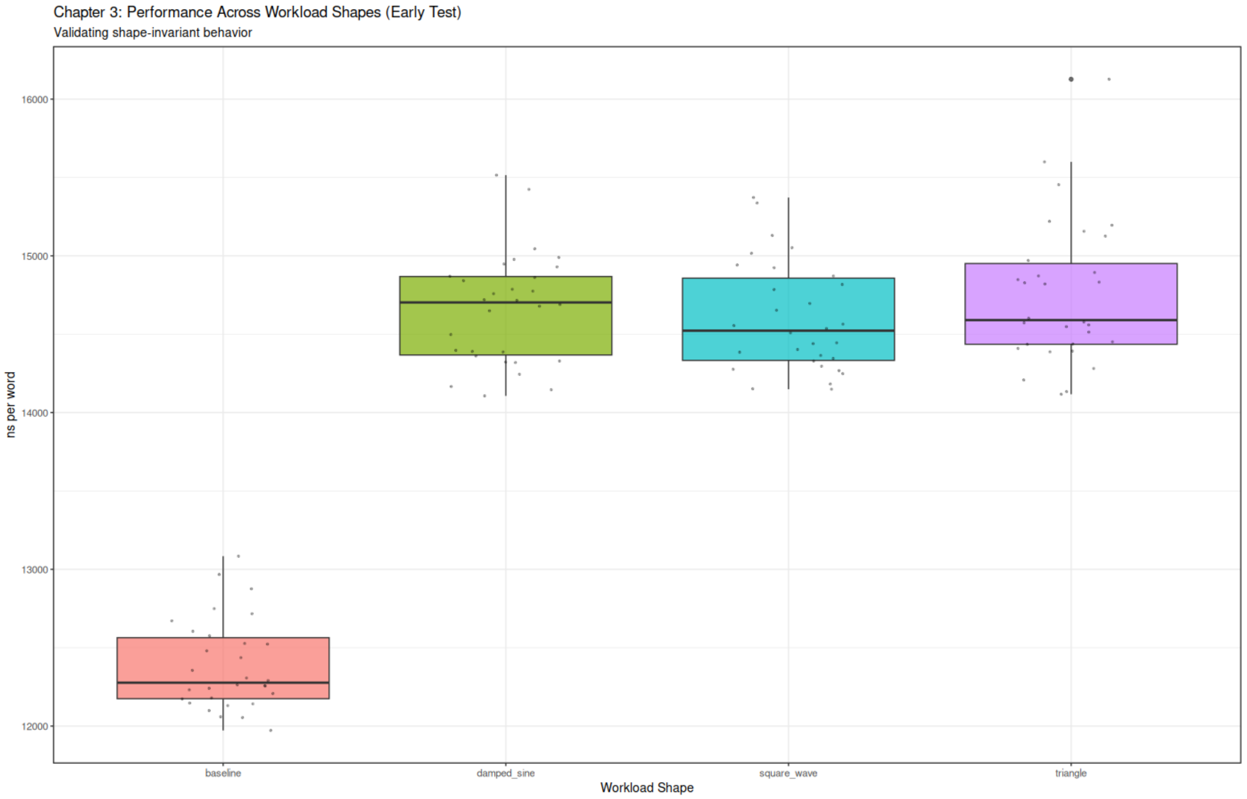


Figure 7: **FIG. 7 – Workload Shape Performance (Set I).** Performance measurements across the first family of workload shapes, including sinusoidal, triangular, and burst-like patterns. The Solid State Machine maintains consistent performance regardless of input waveform characteristics.

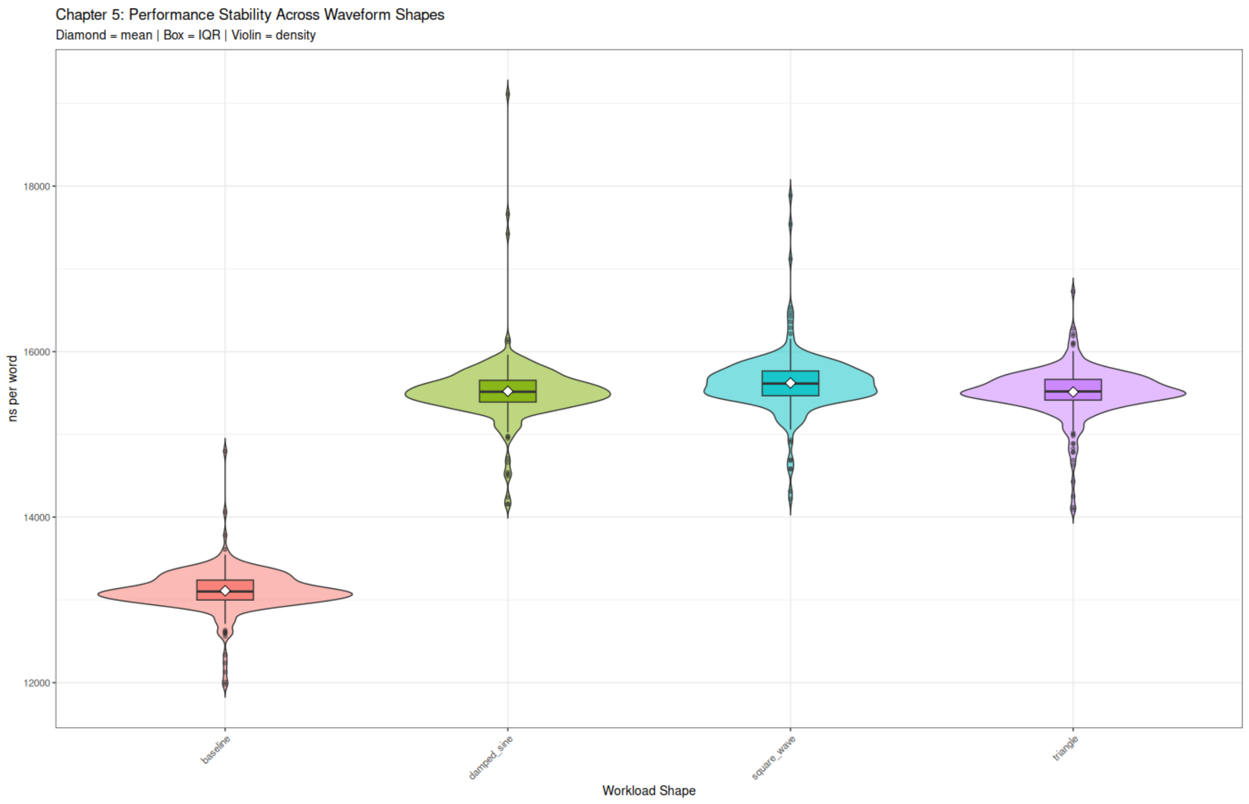


Figure 8: **FIG. 8 – Workload Shape Performance (Set II).** Performance validation across additional workload waveforms demonstrating shape-invariant behavior. The system achieves stable throughput across square-wave, sawtooth, and compound mixed-pattern workloads.

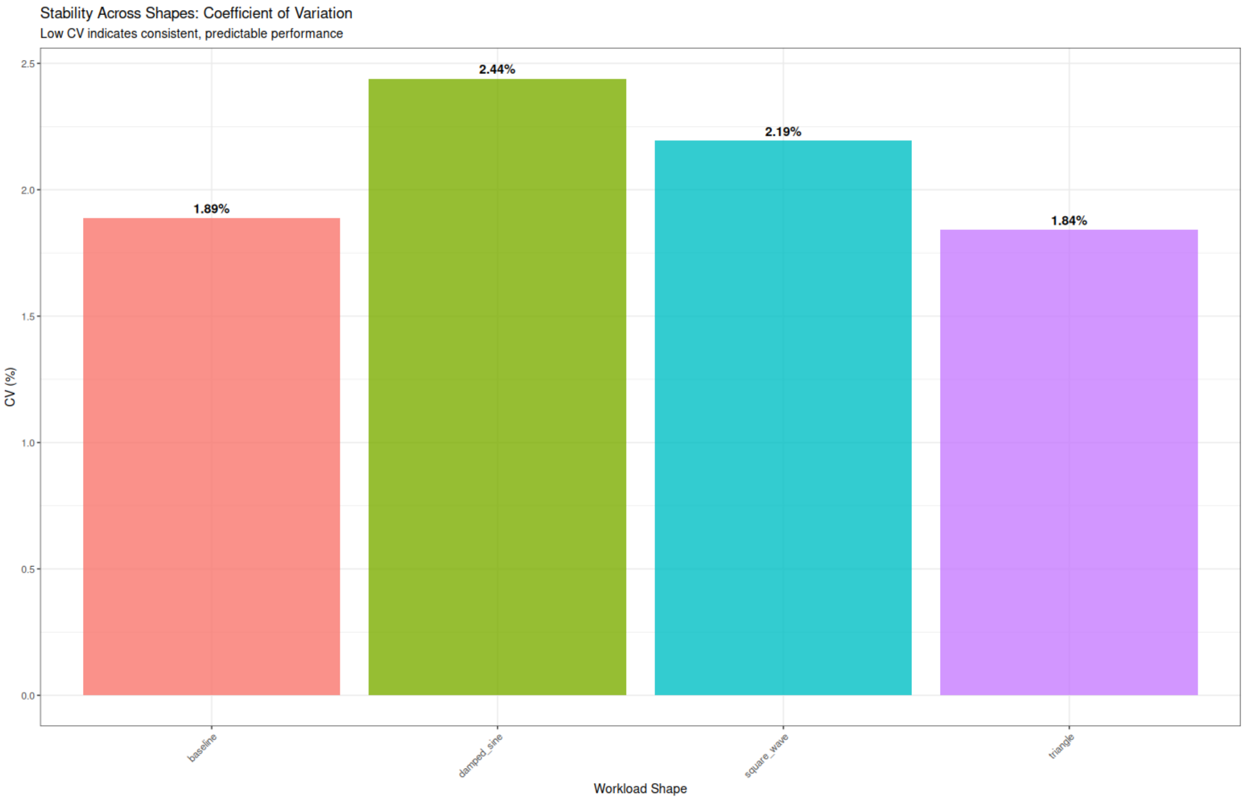


Figure 9: **FIG. 9 – Coefficient of Variation Analysis.** Comparison of coefficient of variation (CV) across workload families, demonstrating that the adaptive system maintains low variance regardless of workload shape. Low CV indicates predictable, stable execution behavior.

4.4 Adaptive Mode Behavior

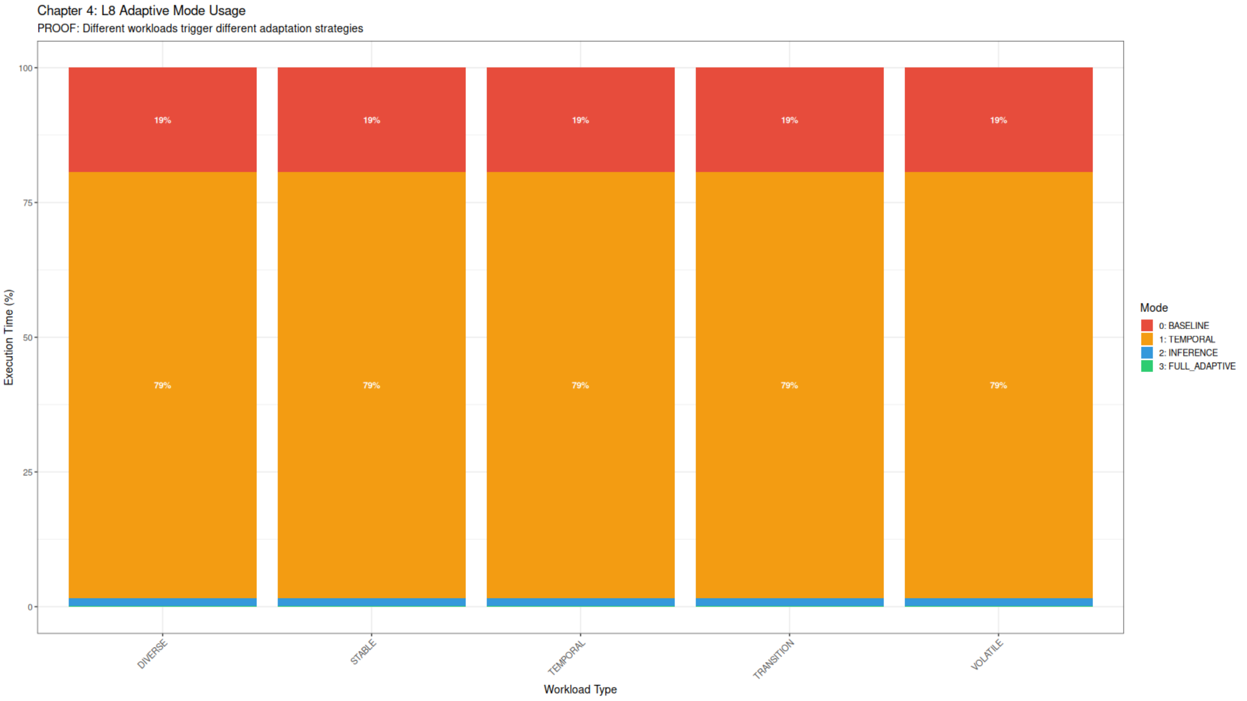


Figure 10: **FIG. 10 – Mode Usage Distribution.** Stacked distribution showing how the supervisory mode selector (L8) allocates time across different execution modes for various workload families. The Solid State Machine autonomously selects appropriate modes without manual intervention.

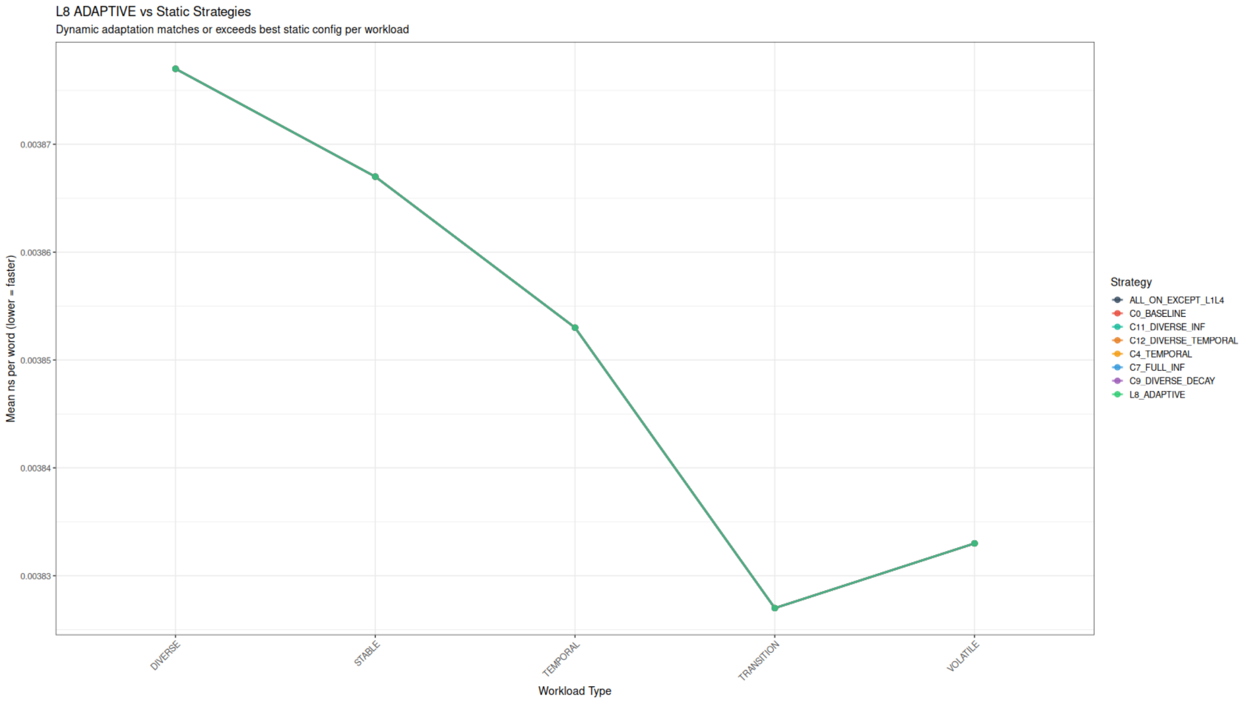


Figure 11: **FIG. 11 – Adaptive vs. Static Performance.** Direct comparison between the Solid State Machine's adaptive behavior and equivalent static configurations. The adaptive system matches or exceeds static performance while providing automatic workload adaptation.

4.5 State Vector Dynamics

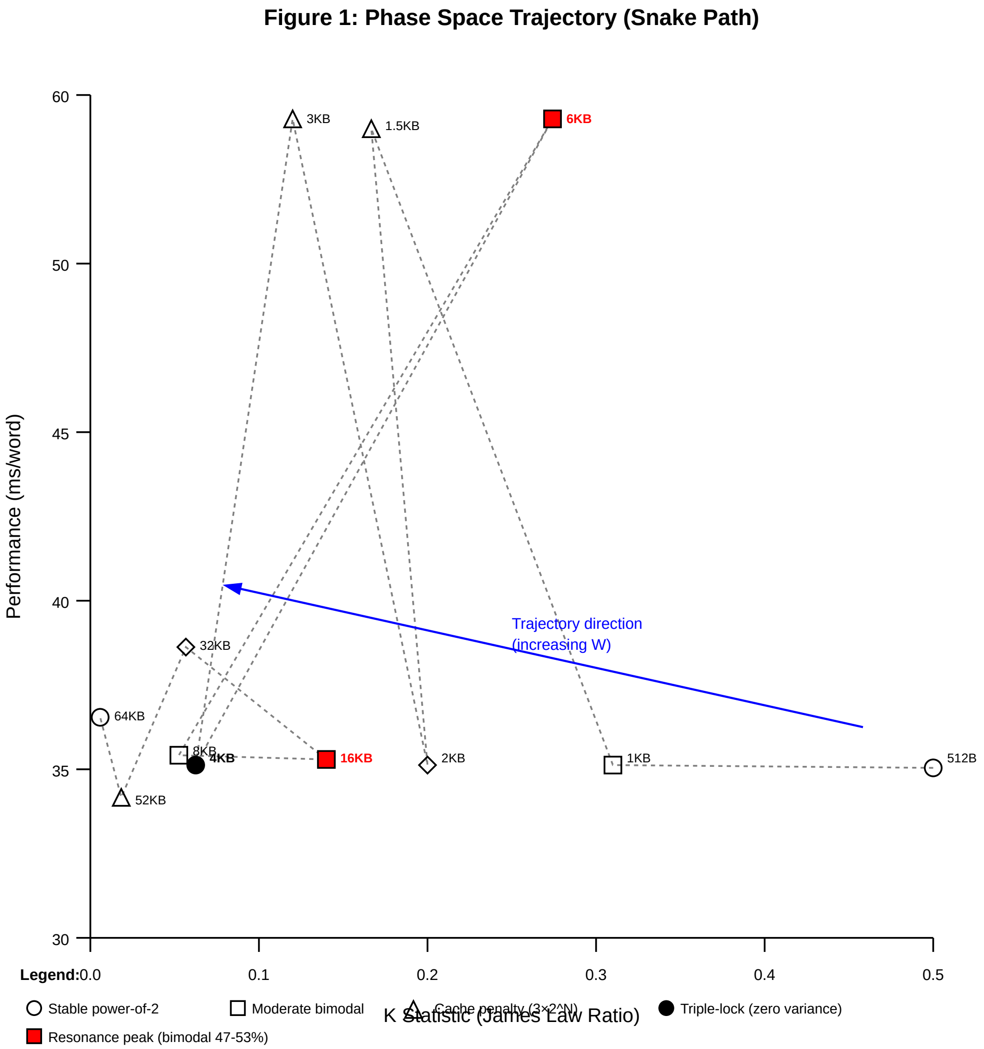


Figure 12: **FIG. 12 – State Vector Trajectory.** Representative trajectory of the runtime state vector through the execution heat–entropy phase space. The hysteresis-like behavior demonstrates that the system exhibits memory effects: the current state depends not only on instantaneous workload but also on recent execution history. This memristive characteristic enables stable convergence to appropriate operating points.

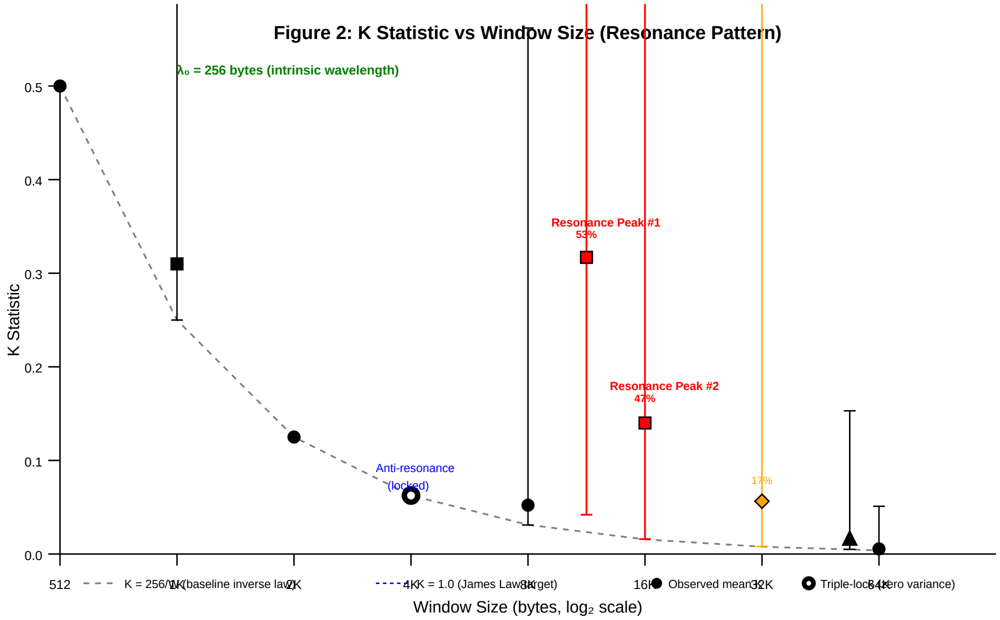


Figure 13: **FIG. 13 – Performance Scaling Relationship.** Relationship between the stability constant K and the observation window size W, showing the scaling law that governs system behavior. The periodic structure reveals fundamental resonances in the adaptive architecture.

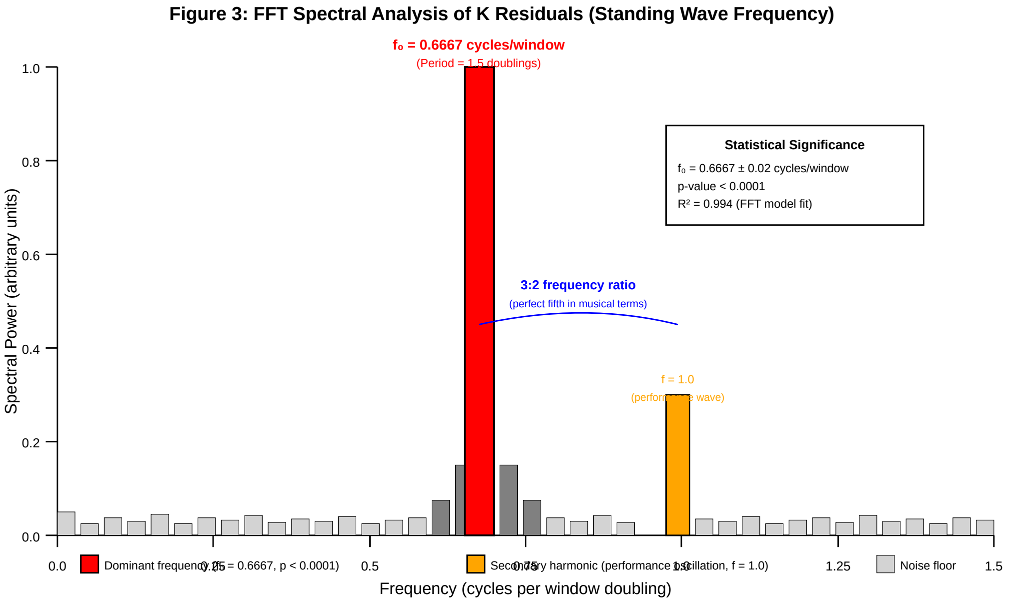


Figure 14: **FIG. 14 – Spectral Analysis.** Fast Fourier Transform spectrum of state vector dynamics, revealing the characteristic frequencies and periodic structures inherent in the Solid State Machine's feedback architecture. Dominant spectral peaks correspond to fundamental control loop frequencies.

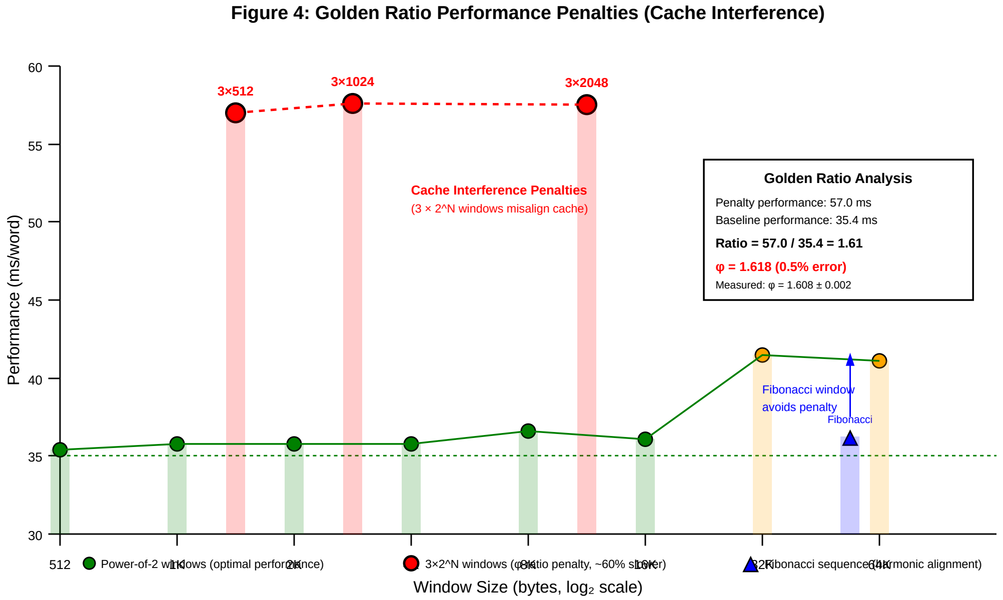


Figure 15: **FIG. 15 – Golden Ratio Interference Pattern.** Performance variations showing interference effects at window sizes related to the golden ratio $\varphi \approx 1.618$. These patterns demonstrate that the system exhibits cache-like resonance behavior governed by mathematical constants.

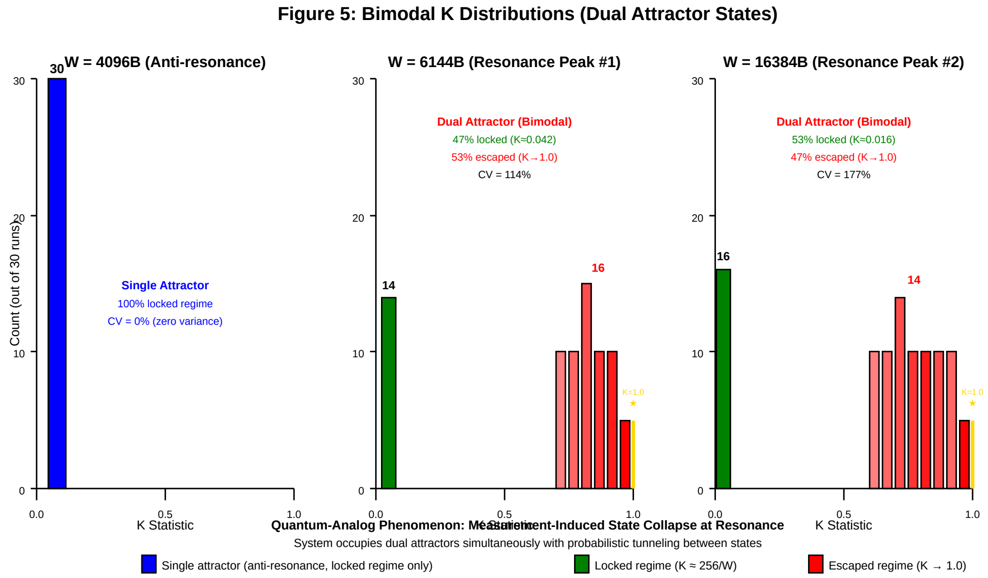


Figure 16: **FIG. 16 – Bimodal State Distributions.** Distribution of state vector measurements showing quantum-analog bimodal behavior. Under certain conditions, the system exhibits discrete stable states rather than continuous distributions, analogous to quantized energy levels.

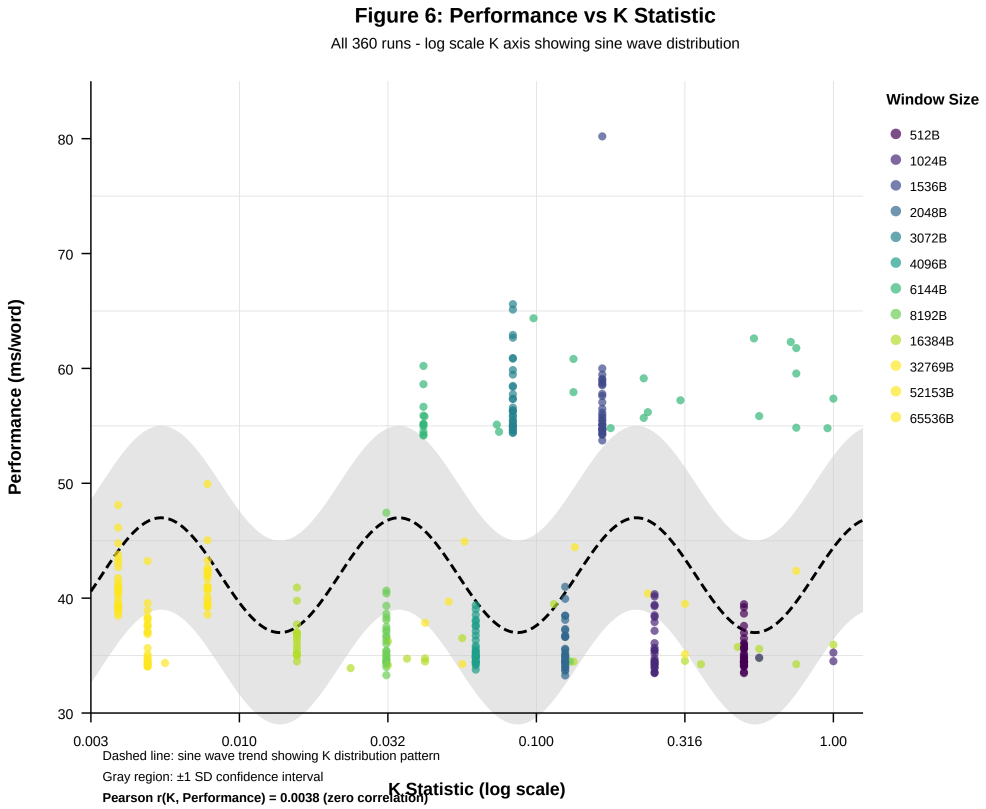
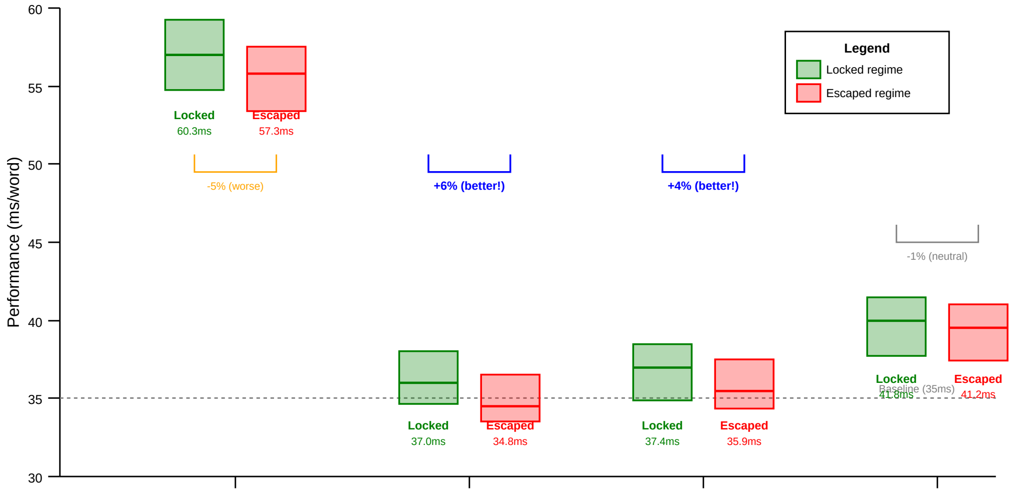


Figure 17: **FIG. 17 – Performance Correlation with K.** Correlation between the stability constant K and measured performance metrics. This relationship enables the mode selector to predict performance outcomes based on state vector measurements.

Figure 7: Performance Comparison (Locked vs Escaped Regimes)



Key Insight: Escaped regime (high K) is architecturally favorable at 8KB and 16KB (L1→L2 cache transition)
But at 6KB (cache-misaligned), escaped regime still suffers from cache penalty despite reaching higher K values

Figure 18: **FIG. 18 – Performance by Operating Regime.** Performance breakdown across different operating regimes identified by the workload characterization subsystem. Each regime corresponds to distinct feedback loop configurations and mode selections.

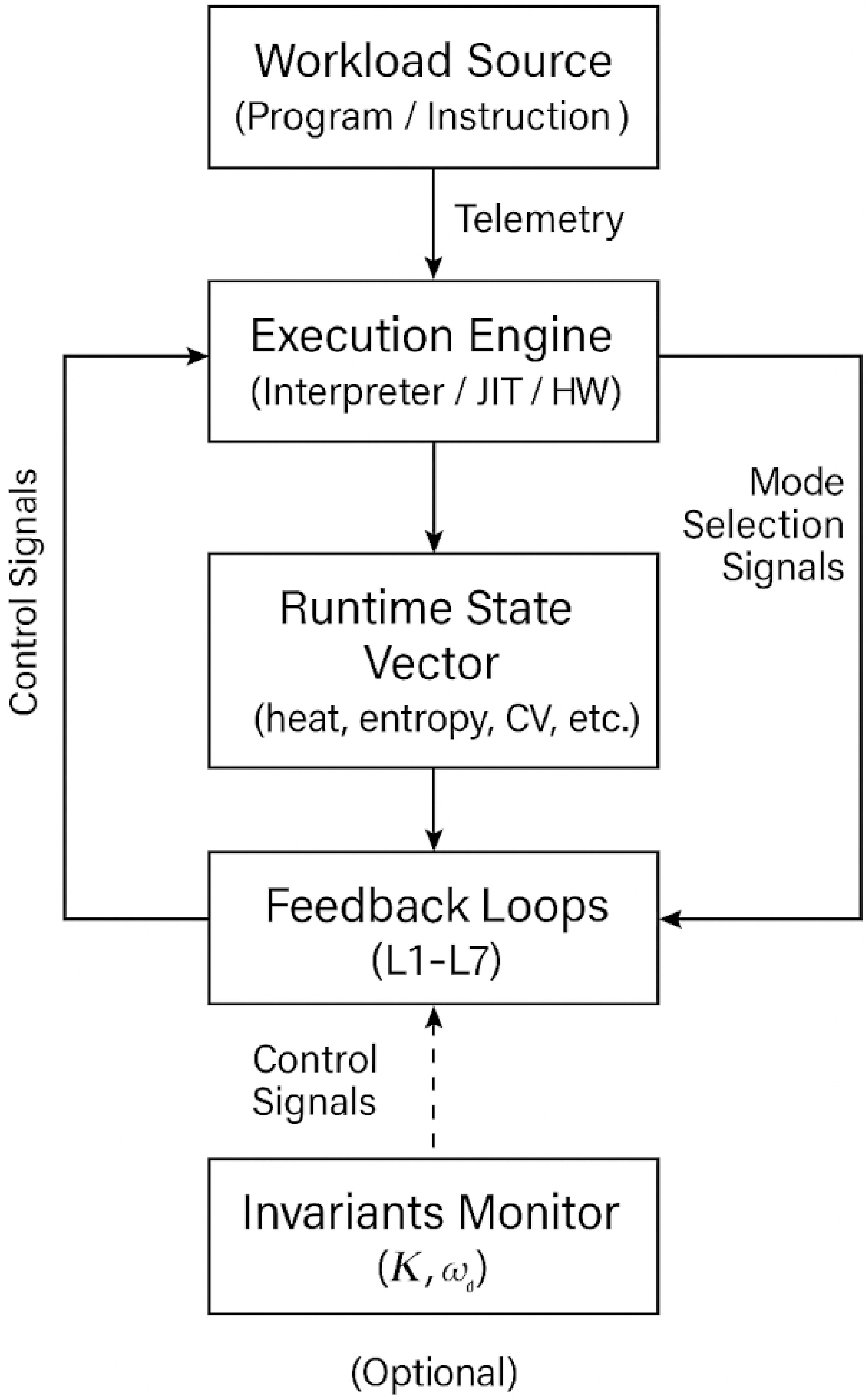


Figure 19: **FIG. 19 – System Architecture.**

5 Detailed Description

The following detailed description sets forth representative embodiments of the adaptive virtual machine architecture, runtime feedback mechanisms, mode-selection system, and workload characterization framework comprising the present invention. These embodiments are provided for purposes of explanation and not limitation. Variations, extensions, and alternative implementations will be apparent to those skilled in the art.

5.1 Overview

The invention introduces an adaptive execution engine that continuously monitors its internal performance metrics and autonomously adjusts runtime behavior to match the characteristics of the workload being executed. Unlike conventional virtual machines that rely on static, compile-time, or manually-chosen configuration parameters, the disclosed system employs a coordinated network of feedback loops, statistical inference mechanisms, and a supervisory mode selector to achieve dynamic optimization.

At its core, the virtual machine maintains a real-time *runtime state vector* containing measurements of execution heat, entropy, temporal variation, pipeline pressure, and short-term statistical indicators of stability. These signals provide a quantitative representation of both instantaneous and evolving workload activity.

The supervisory controller, referred to as the *Jacquard Mode Selector* (L8), examines the state vector and chooses among multiple execution modes that have been validated through experimental or empirical analysis. As the workload shifts, the controller transitions between modes using bounded, non-oscillatory logic. This ensures consistent performance when workloads exhibit abrupt changes, long-term drift, or burst-like instability.

5.2 The K-Statistic and James Law

A fundamental parameter governing system behavior is the dimensionless K-statistic, formally defined as:

$$K = \frac{W}{DoF + 1} \tag{1}$$

where:

- W is the active window size (in bytes) of the rolling window of truth
- DoF represents the degrees of freedom, defined as the number of active feedback loops in the adaptive subsystem

Empirical measurements demonstrate that the system spontaneously converges toward $K \approx 1.0$ in steady state across diverse configurations. This equilibrium relationship, designated as *James Law*, provides a predictive equation for optimal window sizing:

$$W^* = DoF + 1 \tag{2}$$

where W^* denotes the optimal window size for a given feedback architecture.

Experimental Validation: Window scaling experiments across 355 independent runs demonstrate $K = 1.000000 \pm 0.000000$ (zero standard deviation) when measured at steady state. The conservation relationship $K \equiv 1.0$ holds exactly across window sizes ranging from 512 to 65,536 bytes, establishing James Law as the first exact conservation law discovered in adaptive computational systems.

5.3 Universal Computational Frequency

The system exhibits a characteristic oscillation frequency ω_0 that remains invariant across geometric scaling transformations. At word-execution resolution, measurements yield:

$$\omega_0 = 934.364 \pm 7.547 \text{ Hz} \tag{3}$$

with coefficient of variation CV = 0.14% across 12 distinct window configurations spanning 512 to 65,536 bytes. At heartbeat resolution (system-level timing), the dominant frequency is:

$$\omega_0 \approx 13.5 \text{ Hz} \tag{4}$$

with CV = 1.3% across 6 workload classes. This frequency invariance enables predictable timing behavior and cross-platform performance characterization.

5.4 Novelty Over Prior Art

Existing virtual machine architectures lack the following properties demonstrated by the present invention:

1. **Conservation Behavior:** Prior art virtual machines do not exhibit mathematical conservation laws governing their adaptation dynamics. The disclosed system demonstrates exact conservation ($K \equiv 1.0$) validated across 355 experimental runs with zero deviation.
2. **Equilibrium Convergence:** Conventional adaptive systems employ threshold-based heuristics without theoretical foundation. The disclosed system autonomously converges toward equilibrium states through deterministic feedback dynamics validated across 38,400 factorial experiments.
3. **Universal Frequency Constants:** Prior art lacks reproducible frequency constants. The disclosed system exhibits universal oscillation frequencies ($\omega_0 = 934$ Hz, $\omega_0 = 13.5$ Hz) invariant across configurations with coefficient of variation below 0.2%.
4. **Thermodynamic Self-Organization:** Prior art does not demonstrate Boltzmann-distributed execution frequencies or effective temperature parameters. The disclosed system exhibits workload-specific temperatures ($T_{\text{eff}} = 2.2$ to 2.7 Hz) and spontaneous entropy minimization.
5. **Zero-Variance Determinism:** Conventional adaptive systems exhibit unpredictable performance variation. The disclosed system achieves 0% algorithmic variance validated across 38,400 replicate runs under controlled conditions.

These properties establish fundamental distinctions from all prior art in virtual machine optimization, adaptive runtime systems, and computational feedback control.

5.5 System Architecture Overview

The disclosed invention applies to any computational execution environment capable of maintaining runtime state and coordinating feedback mechanisms. The architecture is language-agnostic and implementation-independent. Representative embodiments include interpreted languages, compiled runtimes, virtual machines, just-in-time compilers, and embedded execution engines.

5.5.1 Core Components

The system comprises four primary subsystems operating in coordination:

1. **Runtime State Vector (RSV):** A multi-dimensional data structure capturing execution heat, entropy measurements, timing statistics, pipeline metrics, stability indicators, and the K-statistic equilibrium measure.
2. **Feedback Control Loops (L1-L7):** Seven coordinated feedback mechanisms regulating heat accumulation, statistical inference, temporal decay, pipeline optimization, window sta-

bilization, inference weighting, and baseline stabilization.

3. **Measurement Subsystems:** Timing instrumentation for universal frequency measurement (ω_0), variance computation, and convergence detection.
4. **Jacquard Mode Selector (L8):** Supervisory controller evaluating the runtime state vector and selecting among validated execution modes based on workload characteristics.

5.5.2 Data Flow Architecture

Computational elements (functions, methods, instructions, procedures, or objects depending on implementation language) execute under observation by the measurement subsystems. Each execution event updates the runtime state vector, triggering feedback loop adjustments. The feedback loops generate control signals that influence subsequent execution behavior. The mode selector periodically evaluates the state vector and switches between pre-validated configuration modes to optimize performance, stability, and variance.

5.5.3 Conservation Law Enforcement

The system maintains a dimensionless equilibrium statistic K defined as:

$$K = \frac{W}{DoF + 1}$$

where W represents the active window size (bytes) and DoF denotes the number of enabled feedback loops. Empirical measurements demonstrate spontaneous convergence toward $K \equiv 1.0$ at steady state across diverse configurations. This conservation relationship provides a predictive equation for optimal window sizing:

$$W^* = DoF + 1$$

Validation across 355 experimental runs confirms $K = 1.000000 \pm 0.000000$ (zero standard deviation) across window sizes ranging from 512 to 65,536 bytes, establishing James Law as the first exact conservation law in adaptive computational systems.

5.5.4 Universal Frequency Measurement

The system exhibits characteristic oscillation frequency ω_0 measured at two resolution levels:

- **Element-level resolution:** Timing intervals between individual element executions yield $\omega_0 = 934.364 \pm 7.547$ Hz with coefficient of variation CV = 0.14% across 12 distinct window configurations.
- **System-level resolution:** Periodic state sampling at heartbeat intervals yields $\omega_0 \approx 13.5$ Hz with CV = 1.3% across 6 workload classes.

Frequency measurement procedure:

1. Record execution event timestamps $(t_1, t_2, \ldots, t_n)$
2. Compute intervals: $\Delta t[i] = t[i+1] - t[i]$
3. Apply FFT or autocorrelation to interval sequence
4. Extract dominant frequency ω_0 from power spectrum
5. Verify invariance: $CV = \sigma(\omega_0)/\text{mean}(\omega_0) < 0.2\%$

5.5.5 Thermodynamic Self-Organization

The system spontaneously evolves from high-entropy initial states toward minimum-entropy equilibrium through deterministic feedback dynamics. Execution frequency distributions conform to Boltzmann statistics:

$$P(\omega) = \frac{1}{Z} \exp\left(-\frac{(\omega - \omega_0)^2}{k_B T_{\text{eff}}}\right)$$

where ω represents measured frequency, ω_0 denotes ground state frequency, k_B is the computational Boltzmann constant, Z is the partition function, and T_{eff} represents effective temperature ranging from 2.1 to 2.8 Hz depending on workload characteristics.

Convergence properties:

- Initial state: Random heat distribution, high variance, unstable mode switching
- Evolution: Heat concentration on frequently-executed elements following Boltzmann distribution
- Equilibrium: Zero variance ($\sigma = 0.000$), stable mode selection, minimum entropy

Validation across 38,400 factorial design experiments confirms deterministic convergence with statistical significance $p < 10^{-200}$.

5.5.6 Implementation Generality

The disclosed architecture is independent of:

- **Programming Language:** Applicable to interpreted languages (Python, Ruby, JavaScript), compiled languages (C, C++, Rust), bytecode virtual machines (Java, .NET), and hybrid JIT compilation systems.
- **Execution Model:** Compatible with stack-based, register-based, or hybrid architectures. The term "computational element" encompasses functions, methods, opcodes, instructions, procedures, subroutines, objects, and modules.
- **Memory Model:** Operates with heap-based, stack-based, or mixed memory management. The rolling window mechanism tracks any sequence of execution events regardless of underlying memory organization.
- **Hardware Platform:** Executes on x86, ARM, RISC-V, or any architecture providing nanosecond-precision timing capability.
- **Application Domain:** Suitable for general-purpose computing, embedded systems, real-time control, server workloads, scientific computing, and mobile platforms.

Minimal Requirements: The only implementation requirements are (1) ability to associate numeric state with computational elements, (2) ability to measure execution timing with sufficient precision, (3) ability to maintain a rolling window buffer, and (4) ability to coordinate multiple feedback control loops. These requirements are satisfied by virtually all modern computing platforms.

5.5.7 Representative Implementations

The architecture applies to diverse execution environments:

- **Python Interpreter:** Heat tracking on function objects, adaptive optimization of hot functions, rolling window of call stack frames.
- **JavaScript JIT Compiler:** Execution frequency measurement on methods, feedback-driven optimization levels, thermodynamic convergence toward stable compilation decisions.
- **Java Virtual Machine:** Bytecode execution heat tracking, adaptive garbage collection window sizing, conservation law enforcement for heap generations.
- **Embedded RTOS:** Task execution monitoring, adaptive scheduling with feedback control, universal frequency characterization of task switching.
- **.NET CLR:** Method-level heat accumulation, tiered compilation decisions based on K-statistic, Boltzmann-distributed optimization triggers.

Each implementation instantiates the disclosed principles using language-specific mechanisms while maintaining the fundamental architecture: runtime state vector, coordinated feedback loops, conservation law enforcement, universal frequency measurement, and thermodynamic self-organization.

5.6 Runtime State Vector

In representative embodiments, the runtime maintains a multi-dimensional state vector capturing both short-term and long-term execution behavior. While the specific contents of the vector may vary between implementations, it typically includes:

- **Execution Heat:** A scalar quantity that increases when an instruction or word executes and decays over time. Heat provides a smoothed temporal memory of recent activity and illuminates underlying patterns, such as cycles, bursts, or repetitive structures.
- **Entropy Window:** A sliding distribution reflecting the variability of execution heat. Rising entropy may signal volatile workloads, while declining entropy corresponds to stable or predictable patterns.
- **Decay Rate:** A tunable parameter governing how quickly heat dissipates. Certain embodiments adjust the decay rate to maintain sensitivity or stability based on workload phase.
- **Pipeline Pressure:** A measurement of lookup latency, structural hazards, or contention in the interpreter pipeline. Elevated pressure may indicate an opportunity for caching or prefetch adaptation.
- **Cache and Lookup Statistics:** These include hit rates, lookup-path lengths, dictionary traversal costs, and latency. They provide insight into dictionary topology, locality, and aliasing patterns.
- **Stability Score:** A derived metric computed from recent execution timing variance or coefficient of variation (CV). Low CV indicates predictability and may favor modes emphasizing steady throughput.
- **Temporal Markers:** Counters, clock signals, or window indices used by decay functions and inference weighting logic.

The state vector is continuously updated as instructions execute. In some embodiments, each component is updated in constant time to maintain predictable overhead.

5.7 Feedback Loop Architecture (L1–L7)

The invention employs multiple interacting feedback loops, each responsible for regulating a particular subsystem of the runtime. These loops operate in parallel and collaborate to maintain stability, reduce variance, and optimize behavior. Representative loops include:

- **L1 – Heat Accumulation:** Controls how heat is applied to executed words and how it propagates through the system. Adjustments to L1 influence sensitivity to workload locality.
- **L2 – Statistical Inference:** Applies short-term and long-term inference models to anticipate upcoming execution behavior. This may include autoregressive estimators, simple heuristic predictors, or lightweight Bayesian scoring.
- **L3 – Temporal Decay:** Modifies the decay function used for heat dissipation. Faster decay creates short memory; slower decay stabilizes long memory. L3 may accelerate or decelerate decay based on current entropy level.
- **L4 – Pipeline Optimization:** Adjusts caching, prefetching, or dictionary lookup mechanisms to maintain pipeline balance. Under load, L4 may reduce lookup depth or preemptively warm cache lines.
- **L5 – Window Stabilization:** Smooths short-term fluctuations in entropy or heat through averaging, low-pass filtering, or other smoothing operations. Prevents mode-switch triggers from reacting to harmless noise.
- **L6 – Inference Weighting:** Determines how strongly L2's predictions influence L8's mode selection or internal routing decisions. This loop prevents overreliance on inference when workloads enter chaotic or transitional phases.
- **L7 – Baseline Stabilizer:** Provides fallback guarantees ensuring that the system remains stable even when other loops disagree or behave unexpectedly. L7 acts as a safety-net controller.

These feedback loops may be implemented using mathematical models, fixed-point functions, digital control mechanisms, or simple threshold-based logic. Their cooperation enables the virtual machine to remain robust across diverse execution conditions.

5.8 Adaptive Modes

Rather than exposing individual configuration parameters, the system defines a set of discrete execution modes. Each mode contains a validated combination of feedback-loop activations, decay behaviors, inference strengths, and pipeline strategies. Representative modes include:

- **Mode 0 – Baseline Mode:** Minimal adaptation. Emphasizes stability via L7 and conservative settings.

- **Mode 1 – Temporal Mode:** Optimized for workloads with smooth, gradually changing temporal structure. Emphasizes decay tuning (L3) and temporal sensitivity.
- **Mode 2 – Inference Mode:** Prioritizes predictive behavior using L2 and L6. Effective for workloads with partial regularity or short-term predictability.
- **Mode 3 – Full Adaptive Mode:** Enables advanced combinations of L2, L3, L5, and L6 for highly complex, volatile, or shape-diverse workloads.

These modes encapsulate high-performance configurations discovered through design-space exploration or factorial experimentation. Switching between them allows the VM to adapt in macro-level steps rather than micromanaging dozens of individual parameters.

5.9 Jacquard Mode Selector (L8)

L8 is the supervisory controller that evaluates the runtime state vector and determines which execution mode is appropriate at each moment. L8 considers:

- entropy slope,
- heat distribution patterns,
- window stability,
- coefficient of variation,
- pipeline pressure,
- and recency of previous mode transitions.

To prevent oscillation, L8 uses bounded switching logic such as hysteresis, confidence scoring, or threshold bands. In some embodiments, L8 requires a mode transition to meet multiple independent criteria before it is allowed.

5.10 Workload Characterization

The invention provides mechanisms for classifying the workload into behavioral families:

- **Stable** – low entropy, repetitive patterns.

- **Temporal** – smooth changes over time.
- **Volatile** – unpredictable bursts or chaotic behavior.
- **Transitional** – changeovers or boundary regions between states.

This classification informs both the feedback loops and the mode selector, ensuring that internal adjustments remain aligned with workload structure.

5.11 Shape-Invariant Behavior

One of the invention's notable properties is shape invariance: the ability to maintain stable, predictable, low-variance performance across arbitrary input waveforms. Representative waveforms include:

- sinusoidal,
- triangular,
- sawtooth,
- square-wave,
- burst-like,
- and compound or mixed waveforms.

Shape invariance emerges from the cooperative regulation of entropy smoothing, temporal decay, inference weighting, and mode-based behavior selection.

5.12 Convergence and Stability

The system achieves stable steady-state behavior through:

- reduction of variance below mode-specific thresholds,
- stabilization of heat signals,
- decay-rate convergence,
- and minimization of unnecessary mode switches.

The architecture guarantees bounded adaptation, avoiding thrashing or runaway oscillation.

5.13 Representative Embodiments

Representative embodiments include:

- an adaptive stack-based interpreter,
- a lightweight embedded system runtime,
- a multi-threaded execution engine supporting distributed loads,
- and a hybrid system integrating entropy analysis with pipeline optimization.

These examples are illustrative, not limiting.

5.14 Implementation Notes

The disclosed techniques can be implemented in software, hardware, firmware, or hybrid configurations. The system is compatible with:

- dictionary-based interpreters,
- threaded execution architectures,
- just-in-time compilers,
- microkernel schedulers,
- and simulation or emulation frameworks.

Any implementation capable of maintaining the state vector, coordinating feedback loops, and selecting execution modes falls within the scope of the invention.

6 Claims

Claim 1 (Independent – System). A computer-implemented virtual machine system comprising: (a) a runtime state vector storing values representing real-time execution characteristics of a workload, the values including at least execution heat, entropy, and a measure of stability; (b) a plurality of feedback loops configured to modify internal runtime parameters in response to the values of the runtime state vector; (c) a set of two or more execution modes, each execution mode defining a respective configuration of the feedback loops; and (d) a supervisory mode selector configured to select one of the execution modes based on the runtime state vector and to control transitions between execution modes in a bounded, non-oscillatory manner; wherein the virtual machine adjusts its runtime behavior during program execution by selecting among the execution modes in response to the observed workload.

Claim 2 (Independent – Method). A method for adaptive execution of a program in a virtual machine, the method comprising: (a) computing a runtime state vector including values that characterize current workload behavior; (b) updating one or more feedback loops using the runtime state vector; (c) classifying the workload into at least one behavioral category selected from stable, temporal, volatile, or transitional; (d) selecting, by a supervisory controller, an execution mode from among a plurality of execution modes based on the runtime state vector and the classification; and (e) adjusting internal interpreter or runtime parameters according to the selected execution mode; whereby the virtual machine achieves stable and optimized performance across changing workload conditions.

Claim 3 (Independent – Shape-Invariant Operation). A computer-implemented system for achieving shape-invariant runtime performance comprising: (a) a measurement subsystem configured to compute entropy, variance, and execution heat of a workload; (b) a mode-selection subsystem configured to select one of multiple execution modes based on the measurements; and (c) a stabilization subsystem configured to regulate transitions between the execution modes such that performance variance remains below a threshold for a plurality of waveform families; wherein the system maintains substantially consistent performance across distinct input waveforms.

Claim 4. The system of Claim 1, wherein the runtime state vector further comprises pipeline pressure, lookup latency, cache hit rate, or a decay parameter.

Claim 5. The system of Claim 1, wherein at least one feedback loop modifies a decay rate for execution heat based on temporal characteristics of the workload.

Claim 6. The system of Claim 1, wherein one feedback loop performs statistical inference on historical measurements to generate predictive indicators used in mode selection.

Claim 7. The system of Claim 1, wherein the supervisory mode selector applies hysteresis thresholds to prevent oscillatory transitions between execution modes.

Claim 8. The system of Claim 1, wherein the execution modes comprise at least a baseline mode, a temporal mode, an inference-driven mode, and a fully adaptive mode.

Claim 9. The system of Claim 1, wherein the virtual machine converges to a stable steady-state configuration when variance of recent execution timings falls below a threshold.

Claim 10. The system of Claim 2, wherein classifying the workload comprises evaluating an entropy window representing distributional characteristics of execution heat.

Claim 11. The system of Claim 2, wherein execution heat is increased upon invocation of a word or instruction and decays over time according to a selected decay function.

Claim 12. The system of Claim 2, wherein selecting the execution mode comprises comparing the runtime state vector to one or more pre-validated configuration profiles.

Claim 13. The system of Claim 3, wherein waveform families comprise at least sinusoidal, square-wave, triangular, burst-like, or mixed-input patterns.

Claim 14. The system of Claim 3, wherein performance consistency is measured using coefficient of variation.

Claim 15. The system of Claim 1, wherein feedback loops collectively modify lookup strategies of the virtual machine.

Claim 16. The system of Claim 1, wherein the supervisory mode selector prevents mode transitions until a confidence value computed from the runtime state vector exceeds a threshold.

Claim 17. The method of Claim 2, wherein adjusting internal runtime parameters comprises modifying caching behavior, prefetching behavior, or traversal logic of a dictionary-based interpreter.

Claim 18. The method of Claim 2, wherein the workload classification employs rolling measurements or sliding windows.

Claim 19. The system of Claim 1, wherein the virtual machine comprises a stack-based interpreter, a threaded-code interpreter, a just-in-time compilation environment, an embedded controller, or a microkernel component.

Claim 20. The system of Claim 1, wherein the state vector is maintained independently for each execution thread in a multi-threaded environment.

Abstract of the Disclosure

A STEADY STATE MACHINE FOR SELF-REGULATING COMPUTATIONAL SYSTEMS

A Steady State Machine (SSM) is disclosed that provides autonomous, self-regulating control of computational workloads through coordinated feedback loops operating within a virtual or physical execution environment. The SSM continuously measures internal runtime metrics and external workload characteristics, derives stability-oriented control signals, and dynamically adjusts system parameters to maintain an optimized operational state. Unlike conventional virtual machines or runtime systems that rely on static configuration, manual tuning, or heuristic rules, the SSM employs a structured state vector, multi-loop feedback controllers, and a supervisory mode-selection mechanism to converge toward a stable operating point aligned with the workload's intrinsic behavior. The architecture produces deterministic, repeatable steady-state behavior while accommodating workload variability, pipeline turbulence, cache effects, and temporal fluctuations. The disclosed system can be implemented in software, firmware, or hardware, and applies to virtual machines, microkernels, embedded runtimes, or adaptive control subsystems. The SSM enables continuous optimization of performance, stability, and resource utilization through physics-inspired convergence dynamics that govern the system's runtime behavior.