

Pattern-Based Computing (PBC): A Relaxation-Based Framework for Coordination in Complex Systems

Author Name

Affiliation (to be added)

email@domain.com

January 3, 2026

Abstract

This paper introduces **Pattern-Based Computing (PBC)**, a relaxation-based framework for coordination in distributed and complex systems. In PBC, computation does not proceed through sequential symbolic processing or explicit trajectory optimization, but emerges through the *dynamic relaxation* of the system toward *global patterns* that actively shape system-wide behavior.

PBC is formulated as an explicitly *hybrid architecture*. Classical computation is confined to a strictly *programmatic role*, where it configures and modifies a lower-level input pattern to inject data and intent, while coordination, robustness, and adaptation arise from the intrinsic dynamics of the system itself. Effective computation results from the interaction between a global pattern, sensor-informed intermediate patterns, and local system dynamics.

The framework incorporates two key mechanisms. First, **error self-correction** is achieved through controlled *local decoherences*, which absorb perturbations and prevent their global amplification. Second, the **global pattern adapts** via *upward seduction* originating from lower-level patterns and mediated by intermediate layers, but only during specific *coupling windows*, ensuring structural adaptation without inducing instability.

A central conceptual consequence of PBC is the **collapse of the distinction between program, process, and result**. In this framework, patterns simultaneously constitute the executable program, the computational process, and the resulting outcome, observable at different stages of dynamical stabilization. An illustrative example, interchangeable across continuous domains, together with a robustness evaluation protocol based on rotated simulations, is used to demonstrate the operational validity of the framework. All experimental results are supported by a fully reproducible computational pipeline available with the manuscript.

1 Introduction

Classical computation is traditionally organized around a clear conceptual separation between *program*, *process*, and *result*. A program specifies symbolic instructions, the process consists of their sequential execution, and the result is an explicit output produced upon termination. This separation underlies digital computers, algorithmic problem solving, and most contemporary approaches to optimization, control, and decision-making.

However, many real-world systems do not naturally conform to this computational structure. Large-scale infrastructures, biological systems, socio-technical networks, and distributed multi-agent systems share a number of characteristic features: distributed dynamics, physical constraints, partial observability, persistent perturbations, and the potential for cascading failures. In such systems, computation understood as sequential instruction execution is often fragile, costly, or even ill-defined.

In these contexts, the central challenge is frequently not how to compute an optimal action or trajectory, but how to maintain global coordination while preventing the amplification of local disturbances. Classical control and optimization approaches typically respond to deviations through corrective interventions, implicitly assuming that the system is always receptive to control. When this assumption fails, corrective actions may inject additional energy into the system, increasing instability rather than suppressing it.

We argue that this limitation is not merely algorithmic, but fundamentally *computational*. Certain classes of complex systems require a notion of computation in which coordination emerges from structure and dynamics, rather than from explicit instruction execution or trajectory enforcement. In this work, computation does not refer to symbolic or algorithmic processing, but to the emergence of coordinated system-level behavior through dynamical processes.

To address this need, we introduce **Pattern-Based Computing (PBC)**, a relaxation-based computational framework in which global patterns act as the primary carriers of computation. In PBC, computation is realized through the dynamic relaxation of the system toward stable configurations compatible with these patterns. Rather than computing explicit actions, control signals, or outputs, the system computes by constraining its space of possible futures, favoring stability, coherence, and robustness over instantaneous optimality.

A defining feature of PBC is its explicitly *hybrid* nature. Classical computation is not eliminated, but reassigned to a strictly programmatic role: configuring, modifying, and injecting information into a lower-level pattern that shapes the system’s dynamical landscape. The execution of computation itself is performed by the intrinsic dynamics of the system interacting with patterns across multiple scales.

The framework introduces two key mechanisms that distinguish it from existing approaches. First, PBC enables *self-correction of errors* through controlled local decoherences, which isolate perturbations and prevent their global synchronization or amplification. Second, the global pattern is adaptive rather than static: it evolves gradually through *upward seduction* originating from lower-level patterns and mediated by intermediate structures, but only during specific *coupling windows* in which the system is dynamically receptive, ensuring adaptation without inducing instability.

A fundamental conceptual consequence of PBC is the collapse of the traditional distinction between program, process, and result. In this framework, the pattern itself constitutes the program; its relaxation constitutes the computational process; and the stabilized configuration constitutes the result. These are not separate entities, but different observational perspectives on the same underlying dynamical object.

The contributions of this paper are therefore threefold:

1. We introduce Pattern-Based Computing as a new computational paradigm for distributed and complex systems.
2. We define a hybrid architecture in which classical computation acts solely as a programmer of patterns, while computation emerges through relaxation dynamics.
3. We identify self-correction through local decoherence and adaptation through seduction and coupling windows as fundamental computational mechanisms.

An illustrative example is provided to demonstrate the operability of the paradigm; however, the contribution of this work lies in the computational framework itself rather than in any specific application domain.

2 Pattern-Based Computing

Pattern-Based Computing (PBC) is a computational paradigm in which computation is not performed by executing symbolic instructions or by explicitly optimizing objective functions,

but by shaping and exploiting the *dynamical evolution* of a system toward stable configurations defined by *patterns*. In this framework, patterns are not representations to be interpreted, but active structures that directly bias system dynamics.

This section provides a formal and operational description of PBC, clarifying its core components and the notion of computation it embodies.

2.1 Patterns as Primary Computational Objects

In PBC, a *pattern* is a persistent, distributed structure that biases the evolution of a system toward a family of compatible configurations. Unlike symbolic data structures or control signals, patterns are:

- **Continuous in time:** they persist throughout the computational process rather than being applied instantaneously.
- **Distributed:** they influence many components simultaneously without centralized mediation.
- **Weakly prescriptive:** they do not impose explicit actions, but define tendencies and constraints.

From a computational perspective, a pattern encodes information by shaping the system’s state space. Presenting a pattern is equivalent to defining a computational landscape in which certain regions of the state space are energetically or structurally favored, while others are disfavored.

2.2 Computation as Dynamic Relaxation

Once a pattern is active, the system evolves according to its intrinsic dynamics. Computation occurs as the system *relaxes* toward configurations that are compatible with the pattern.

Let $x(t)$ denote the distributed state of the system, and let $\mathcal{P}(t)$ denote the active pattern. The system dynamics can be expressed abstractly as:

$$x(t + \Delta t) = F(x(t)) + \alpha(t) \Delta_{\mathcal{P}}(x(t)), \quad (1)$$

where F represents the system’s natural dynamics, $\Delta_{\mathcal{P}}$ represents the influence induced by the pattern, and $\alpha(t) \in [0, 1]$ is a coupling factor.

Crucially, the primary computational variable in PBC is not the pattern-induced influence $\Delta_{\mathcal{P}}$ itself, but the coupling factor $\alpha(t)$, which determines how strongly the pattern shapes the system’s evolution at each moment.

2.3 Receptivity and Coupling Modulation

The coupling factor $\alpha(t)$ reflects the *receptivity* of the system: its capacity to absorb pattern influence without generating structural damage or instability. Receptivity is inferred from system-level signals such as coherence, tension accumulation, relaxation rates, or other indicators of dynamical stress.

When receptivity is high, pattern influence is amplified and the system aligns more strongly with the pattern. When receptivity is low, coupling is attenuated, allowing local deviations to persist and preventing forced synchronization.

As a consequence, computation in PBC consists in *modulating influence* rather than in computing explicit corrective actions, control signals, or trajectory adjustments.

2.4 Distributed and Parallel Computation

Because pattern influence acts simultaneously across the system, PBC inherently performs computation in a distributed and parallel manner. There is no notion of sequential execution, centralized decision-making, or global scheduling. Each component responds locally to the pattern and to its own state, while global coherence emerges through interaction.

This form of computation scales naturally with system size: increasing the number of components does not increase the depth of computation, but only its spatial extent. As a result, larger systems can, in principle, support richer computational structures without increased fragility.

2.5 Outputs as Stabilized Regimes

In PBC, computation does not terminate with the production of an explicit symbolic output. Instead, the result of a computation is a *stabilized configuration* or a *persistent dynamical regime* reached by the system.

Such regimes are themselves physical states of the system and can serve as inputs to subsequent computational phases. Consequently, computation in PBC is inherently continuous and accumulative, rather than episodic and terminating.

2.6 Summary

Pattern-Based Computing defines computation as the interaction between patterns and system dynamics under receptivity constraints. Patterns act as executable computational structures, relaxation replaces instruction execution, and stabilized regimes replace explicit outputs.

This redefinition sets the stage for the architectural mechanisms described in the following section, including hybrid pattern–classical computation, multi-scale pattern organization, error self-correction through local decoherences, and adaptive pattern evolution.

3 Architecture of Pattern-Based Computing

Pattern-Based Computing (PBC) is realized through a structured architecture that combines pattern dynamics with classical computation in a strictly separated yet interacting manner. This section describes the operational architecture of PBC, detailing its hybrid organization, geometric structure, information flows, and intrinsic mechanisms for error correction and adaptation.

3.1 Hybrid Computational Architecture

PBC is explicitly designed as a **hybrid computational system** composed of two fundamentally different layers:

- **Pattern-based layer (continuous, fast):** This layer performs computation through dynamic relaxation toward patterns. It is responsible for coordination, stability, and the absorption of perturbations.
- **Classical computational layer (discrete, slow):** This layer does not perform coordination or control. Its sole function is to **program the pattern landscape** by configuring and modifying the lower-level pattern that introduces data, constraints, or intent.

The classical layer never interacts directly with the system dynamics. All computational influence must pass through patterns.

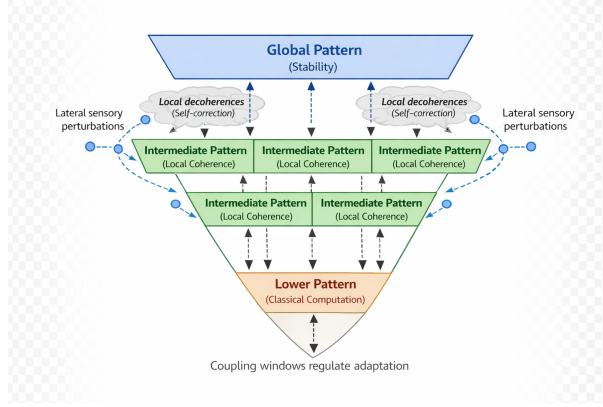


Figure 1

Figure 1: Conceptual architecture of Pattern-Based Computing (PBC). Classical computation operates at the lower level as a configurational substrate, shaping the initial state space without prescribing explicit actions. At each intermediate layer, multiple pattern instances coexist and interact laterally through mutual relaxation, forming local coherence structures that are weakly constrained by the global pattern. Sensory signals enter the system as lateral perturbations to intermediate patterns, modulating local receptivity rather than acting as direct computational inputs. The global pattern provides slow, system-wide structural constraints, shaping the space of admissible configurations while allowing local decoherence for self-correction and adaptation.

Coupling windows regulate when structural changes propagate across layers.

3.2 Geometric Organization: The Inverted Triangle of Patterns

The internal organization of PBC can be represented as an **inverted triangle of patterns**, which defines how information propagates and is amplified:

- **Global pattern** \mathcal{P}_G : A persistent, system-wide structure encoding global intent and stability principles.
- **Intermediate patterns** $\{\mathcal{P}_i\}$: Distributed patterns that integrate information from sensors, the global pattern, and the lower pattern.
- **Lower pattern** \mathcal{P}_B : The interface programmable by classical computation, used to inject data and reconfigure the computational landscape.

The inverted geometry ensures that a single global pattern can influence many intermediate structures, while no local or lower-level element can directly impose global changes.

3.3 Information Flows

Three simultaneous information flows operate within the PBC architecture:

Bottom-up sensory flow. Local system dynamics generate sensor signals reflecting coherence, tension, and stability. These signals feed the intermediate patterns, which synthesize structural information for the global pattern.

Top-down pattern influence. The global pattern emits influence signals that shape the evolution of intermediate patterns, biasing local dynamics toward globally compatible configurations.

Lateral programming flow. Classical computation modifies the lower pattern. This influence propagates upward through intermediate patterns but never reaches the global pattern directly.

These flows form a closed-loop interaction without centralized control or direct command paths.

3.4 Error Self-Correction through Local Decoherence

A key property of PBC is its intrinsic mechanism for **error self-correction**. Errors are not treated as deviations to be corrected immediately, but as local incompatibilities with the active pattern.

When a perturbation arises:

1. Local coherence decreases.
2. Coupling to the global pattern is reduced in the affected region.
3. The perturbation is spatially and temporally isolated.
4. As tension relaxes, coupling is gradually restored.

These controlled **local decoherences** prevent error amplification and cascading failures. Correction emerges through relaxation rather than explicit intervention.

3.5 Upward Seduction and Coupling Windows

While the global pattern provides structural stability, it is not immutable. Adaptation occurs through a process termed **upward seduction**.

The lower pattern, shaped by classical computation, exerts a weak and persistent influence that propagates upward via intermediate patterns. The global pattern, however, only adapts during specific **coupling windows**, which open when:

- global tension is decreasing,
- coherence is sufficiently high,
- perturbations are locally contained.

Outside these windows, the global pattern remains stable and insensitive to lower-level influence. This mechanism enables structural learning without volatility or noise-driven drift.

3.6 Programming a Pattern

In Pattern-Based Computing, programming does not consist in specifying algorithms, control laws, or action sequences. Instead, programming refers to the configuration of the **pattern landscape** that biases system dynamics.

Classical computation programs the lower pattern \mathcal{P}_B by shaping the structural constraints under which relaxation occurs. This programming defines:

- which configurations are structurally favored,
- which regions of the state space are discouraged,
- and which degrees of freedom remain unconstrained to allow adaptation.

Programming a pattern does not determine trajectories or prescribe local actions. It specifies compatibility conditions that shape admissible futures without enforcing them.

Operationally, pattern programming may involve modifying coupling strengths, spatial weights, admissible ranges, or weak global tendencies. These operations are discrete and classical, but their effects are indirect: they alter the geometry of the pattern rather than executing coordination.

An essential property of PBC is that programming remains intentionally weak. Pattern influence is always subject to receptivity modulation and may be locally suspended through decoherence. This ensures that incompatible or poorly specified programs do not force unstable behavior or generate false solutions.

3.7 Identity of Program, Process, and Result

In PBC, the traditional distinction between program, execution, and output collapses:

- The **program** is the active pattern configuration.
- The **process** is the system’s relaxation under pattern influence.
- The **result** is the stabilized pattern or dynamic regime reached.

These are not separate entities but different observations of the same dynamical object at different stages of stabilization. Consequently, PBC computation is continuous, accumulative, and state-preserving.

3.8 Summary

The architecture of Pattern-Based Computing combines hybrid computation, geometric pattern organization, distributed information flows, self-correcting dynamics, and protected adaptation. Classical computation programs the pattern landscape, while computation itself emerges from pattern-driven relaxation, yielding a robust and scalable computational paradigm.

4 Discussion and Relation to Other Computational Paradigms

Distinct Contributions of Pattern-Based Computing

This work introduces Pattern-Based Computing (PBC), which differs from existing computational paradigms in several fundamental aspects:

- Computation is defined as relaxation toward pattern-compatible regimes rather than action selection or explicit trajectory optimization.
- Patterns are treated as executable computational objects, not as representations to be interpreted or as emergent outcomes observed post hoc.
- System receptivity is introduced as an explicit computational variable that governs the admissibility and strength of pattern influence.
- Error correction is achieved through controlled local decoherence, allowing perturbations to be isolated rather than immediately corrected through global intervention.
- Adaptation of global structure occurs only during receptive coupling windows, preventing noise-driven or volatile structural drift.
- Failure is expressed dynamically as persistent instability (fever states) rather than as incorrect symbolic or numerical outputs.

Pattern-Based Computing (PBC) does not aim to incrementally improve existing computational methods, but to reformulate what it means to compute in distributed and complex systems. This section situates PBC with respect to established paradigms and clarifies the class of problems for which it is well suited.

4.1 Relation to Turing-Style Computation

Turing-style computation relies on the sequential manipulation of discrete symbols under a fixed rule set, with a strict separation between program, execution, and output. Computation proceeds step by step and terminates with an explicit symbolic result.

PBC departs fundamentally from this model. It does not operate on symbols, but on continuous, distributed patterns; it is not sequential, but parallel and dynamical; and it does not terminate with an external output, but stabilizes into persistent dynamical regimes. While Turing computation excels at exact symbolic manipulation, PBC targets systems in which enforcing discrete correctness at every step is either infeasible or counterproductive.

4.2 Relation to Artificial Neural Networks

Artificial Neural Networks (ANNs) also rely on distributed and parallel computation, but their similarity to PBC is limited. ANNs encode computation in fixed architectures and parameterized weights, and rely on explicit optimization procedures to adjust these parameters.

In contrast, PBC does not minimize a loss function and does not separate training from execution. Patterns in PBC are dynamic structures that evolve continuously through interaction with the system. Adaptation occurs through upward seduction and coupling windows rather than through gradient-based updates. Moreover, PBC explicitly separates programming (classical computation shaping patterns) from execution (relaxation dynamics), whereas ANNs conflate both within parameter adjustment.

4.3 Relation to Classical Control and Model Predictive Control

Classical control and Model Predictive Control (MPC) compute corrective actions based on deviations from a reference or by optimizing future trajectories. These approaches assume that the system is always receptive to intervention and that corrective actions can be safely injected.

PBC replaces action computation with influence modulation. Rather than correcting deviations, it modulates coupling strength based on system receptivity, allowing local deviations to persist when forced correction would increase instability. Unlike MPC, PBC does not predict trajectories or solve optimization problems online; it constrains the space of admissible futures by shaping the pattern landscape, remaining computationally lightweight at runtime.

4.4 Conceptual Comparison

Table 1 summarizes the conceptual differences between PBC and other paradigms.

	Turing	ANN	Control/MPC	PBC
Representation	Symbols	Weights	States	Patterns
Execution	Sequential	Parallel (fixed)	Action-based	Relaxation
Error handling	Explicit	Loss-driven	Correction	Decoherence
Adaptation	Reprogram	Retrain	Retune	Seduction
Output	Symbolic	Numeric	Action	Stable regime

Table 1: Conceptual comparison between Pattern-Based Computing and other computational paradigms.

4.5 Robustness-First Systemic Impact

The contribution of PBC should not be evaluated primarily through average-case performance metrics. Its impact lies in enabling computational behavior that prioritizes global stability over local optimality.

By allowing controlled local decoherences and delaying global adaptation, PBC:

- maintains coherence in regimes where corrective control amplifies instability,
- absorbs perturbations without immediate intervention,
- avoids cascading failures under partial observability,
- degrades gracefully instead of failing catastrophically.

These properties are particularly relevant for large-scale and safety-critical systems, where preventing collapse is often more important than optimizing nominal performance.

4.6 Failure Semantics and Fever States

A defining feature of PBC is its qualitatively different expression of failure. In classical paradigms, failure often manifests as incorrect outputs, unstable control actions, or silent divergence, which may appear plausible despite being invalid.

In PBC, the absence of a compatible pattern is expressed dynamically. When no stable configuration exists under the current constraints, the system does not converge to a spurious solution. Instead, it enters a persistent instability regime, referred to as a *fever state*, characterized by sustained fluctuation and lack of global coherence.

This behavior is informative rather than deceptive: it signals the absence of a viable solution, avoids false stabilization, and provides a clear diagnostic cue for reprogramming or structural change. Failure in PBC is therefore explicit, observable, and computationally meaningful.

4.7 Scope and Domain of Applicability

Pattern-Based Computing is not intended as a universal replacement for classical computation. Its operation relies on continuous or quasi-continuous dynamics in which relaxation, coherence, and gradual influence modulation are well defined.

PBC is applicable to domains characterized by distributed interactions, physical or dynamical constraints, and emergent global behavior. It is not suitable for inherently discrete, symbolic, or combinatorial problems where correctness depends on exact symbolic manipulation or discrete termination.

This restriction is not a limitation, but a defining characteristic. PBC complements existing paradigms by addressing a class of problems for which traditional computation is structurally ill-suited.

4.8 Contrast with Non-Physical and Symbolic Domains

To clarify the scope and limitations of Pattern-Based Computing, it is instructive to contrast its operation with domains that are not physically or dynamically grounded.

Consider a purely symbolic task such as exact arithmetic, formal proof verification, or combinatorial constraint satisfaction. In such domains, system states evolve through discrete, atomic transitions, and correctness depends on precise symbolic manipulation and explicit termination conditions.

In these settings, there is no meaningful notion of gradual relaxation, partial coherence, or local decoherence. A symbolic expression is either correct or incorrect; an inference step

either preserves validity or it does not. Allowing temporary inconsistency or ambiguity does not contribute to convergence, but instead destroys correctness.

As a result, Pattern-Based Computing is fundamentally ill-suited to such domains. Attempting to apply PBC to symbolic computation would lead either to persistent instability (a permanent fever state) or to arbitrary stabilization unrelated to semantic correctness.

This contrast highlights a defining characteristic of PBC: its computational semantics rely on the existence of continuous dynamics in which stability, tension, and relaxation are well-defined. Where these notions do not exist, Pattern-Based Computing does not merely perform poorly—it ceases to be a meaningful computational model.

Rather than constituting a limitation to be overcome, this restriction deliberately positions PBC as a complementary paradigm. Classical symbolic computation remains indispensable for discrete, exact, and logically closed domains, while PBC targets systems in which physical continuity, uncertainty, and distributed interaction dominate.

4.8.1 Governance as Pattern-Based Coordination in Socio-Technical Systems

Large-scale governance systems can be understood as socio-technical systems whose primary computational challenge is not optimal decision-making, but the maintenance of coordination, legitimacy, and long-term stability under persistent perturbations.

From a Pattern-Based Computing perspective, governance operates through global patterns—such as institutional norms, legal frameworks, or shared expectations—that weakly constrain local behavior without prescribing explicit actions. Individual actors and institutions interact locally, while coherence emerges from the compatibility between these interactions and the global pattern.

Importantly, local disruptions such as dissent, protest, or institutional conflict need not be interpreted as computational failures. Within the PBC framework, such events correspond to local decoherences that temporarily reduce coupling to the global pattern. Rather than being suppressed immediately, these decoherences can function as corrective mechanisms, preventing the rigid enforcement of incompatible structures and isolating perturbations before they escalate system-wide.

Adaptation of governance structures follows a mechanism analogous to upward seduction. Persistent local signals may gradually reshape global patterns, but only during receptive phases—when systemic tension is decreasing and coherence is sufficiently high. Outside these coupling windows, global structures remain stable, preventing noise-driven or volatile change.

This interpretation does not prescribe political outcomes or normative decisions. Instead, it illustrates how Pattern-Based Computing provides a computational lens for understanding coordination, correction, and stability in complex socio-technical systems, highlighting why apparent failures may be necessary for long-term robustness.

4.9 Implications

By embedding computation in pattern-driven dynamics and collapsing the distinction between program, process, and result, Pattern-Based Computing offers a framework tailored to systems where robustness, scalability, and interpretability of failure are more critical than precise symbolic outputs. Rather than replacing existing paradigms, PBC extends the computational toolbox to domains where stability itself is the primary computational objective.

5 Applicable Domains and Illustrative Example

Pattern-Based Computing is not intended as a universal computational framework. Its applicability depends on structural properties of the domain in which computation is embedded. This

section first delineates the classes of systems for which PBC is well suited, and then introduces an illustrative domain used to demonstrate its operational behavior.

5.1 Domains Suitable for Pattern-Based Computing

PBC is applicable to systems that exhibit the following structural characteristics:

- **Continuous or quasi-continuous dynamics**, in which state variables evolve gradually over time.
- **Distributed organization**, with many interacting components and no single point of control.
- **Emergent global behavior**, arising from local interactions rather than centralized coordination.
- **Tolerance to local deviations**, where temporary incoherence can be absorbed without systemic failure.
- **Stability as a primary objective**, often more critical than instantaneous optimality.

Representative domains include large-scale infrastructures (transportation, energy, water networks), biological and ecological systems, collective robotics, socio-technical systems, and continuous resource-flow processes. In such settings, enforcing discrete correctness at every step is either infeasible or counterproductive, and computation by relaxation toward patterns provides a natural and robust alternative.

Conversely, PBC is not designed for inherently discrete or symbolic domains, such as exact arithmetic, formal logic, or combinatorial search problems, where correctness depends on precise symbolic manipulation and explicit termination conditions. This limitation is not a weakness, but a defining characteristic of the paradigm.

5.2 Traffic as an Illustrative Domain

To illustrate the principles of Pattern-Based Computing, we employ a synthetic freeway traffic system. Traffic is not the subject of this work; rather, it serves as a transparent and representative testbed due to several favorable properties:

- traffic dynamics are continuous and physically grounded,
- coordination emerges from local interactions among agents,
- perturbations (e.g., demand fluctuations, incidents) are unavoidable,
- aggressive corrective control can amplify congestion,
- global stability is often more important than local throughput maximization.

These properties make traffic a convenient exemplar of a broader class of systems in which stability-oriented computation is meaningful.

5.3 Role and Scope of the Example

The traffic system is used exclusively for illustrative purposes. Its role is to demonstrate:

- how patterns can be programmed and presented,
- how local decoherences emerge and self-correct,

- how global patterns adapt through coupling windows,
- how stability-based computation differs from trajectory optimization or action-centric control.

All mechanisms observed in the traffic example are domain-agnostic and can, in principle, be instantiated in other continuous systems with analogous structural properties.

5.4 Transition to the Experimental Setup

The following section describes the traffic-based experimental setup and evaluation protocol used to illustrate Pattern-Based Computing in practice. The emphasis is placed on computational behavior and robustness under controlled perturbations, rather than on traffic-specific modeling or performance optimization.

6 Illustrative Example: Traffic System and Experimental Results

This section presents an illustrative implementation of Pattern-Based Computing using a synthetic freeway traffic system, together with the experimental protocol and results. The objective of these experiments is not to optimize traffic performance, but to evaluate the computational properties of PBC—such as robustness, stability, and failure semantics—under systematic structural variation.

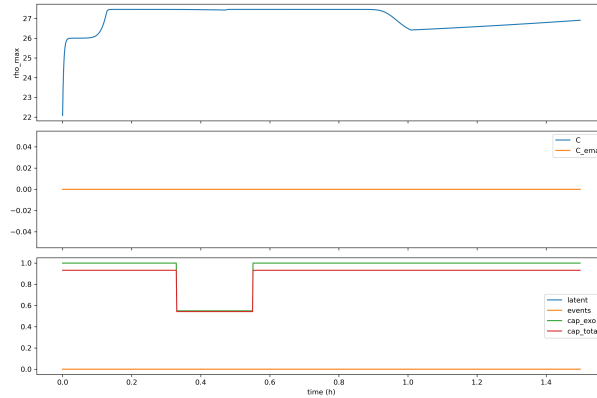


Figure 2: Representative time evolution of the synthetic freeway system under Pattern-Based Computing (PBC). The figure shows the temporal dynamics of traffic density, flow, ramp-metering signals, latent structural stress, discharge events, and effective capacity. Local decoherences appear as transient fluctuations that reduce coupling without triggering global instability, enabling relaxation toward coherent regimes. This visualization illustrates the internal computational dynamics of PBC beyond aggregate performance metrics.

6.1 Synthetic Traffic System

The illustrative system consists of a discretized freeway modeled as a continuous-time flow system with local interactions and capacity constraints. Traffic demand, disturbances, and local congestion emerge naturally from the system dynamics.

The system satisfies the structural conditions required by Pattern-Based Computing:

- continuous state evolution,
- distributed local interactions,
- sensitivity to perturbations,

- and the possibility of cascading instability.

Patterns are introduced as weak global structures that influence local flow through receptivity-modulated coupling. Classical computation is used exclusively to program the lower pattern and to inject external information; it does not execute coordination or control actions directly.

6.2 Experimental Protocol

To evaluate robustness independently of specific scenarios, we employ a protocol based on **rotated simulations**. Each experimental configuration is subjected to:

- temporal rotations (phase shifts),
- spatial rotations (reordering of local structures),
- discretization variations.

This protocol prevents overfitting to a single realization and probes the stability of the computational paradigm under structural variation.

Three strategies are compared:

- an open-loop baseline,
- a reactive alignment strategy,
- a Pattern-Based Computing (PBC) implementation.

The same demand profiles and perturbations are applied across all rotations and seeds.

6.3 Evaluation Metrics

System behavior is characterized using the following metrics:

- **Total delay**, as a proxy for accumulated instability,
- **Integrated congestion**, measuring sustained incoherence,
- **Energy of control influence**, reflecting structural intervention effort,
- **Frequency of rare events**, indicating systemic fragility.

These quantities are interpreted as indicators of *computational stability* rather than domain-specific performance.

Figure 3 highlights a central property of Pattern-Based Computing: system performance is not characterized by energy minimization, but by the *controlled transformation of system structure*.

Although relaxation-based coupling incurs a higher total energy expenditure, it dramatically reduces incoherence duration and recovery latency, while simultaneously enabling a substantially larger structural gain. Local decoherences—typically interpreted as failures in threshold-based control—act instead as corrective mechanisms that facilitate global reconfiguration without triggering instability cascades.

This illustrates a fundamental distinction between PBC and reactive control: energy is deliberately invested to reshape the pattern landscape, trading instantaneous efficiency for systemic robustness and scalable coherence.

Figure 4 illustrates the qualitative behavioral differences between reactive control and pattern-based coordination under identical demand and perturbation conditions. Reactive ramp metering responds directly to local density deviations, resulting in sharp metering actions and the accumulation of ramp queues. In contrast, the pattern-based approach applies weaker and smoother

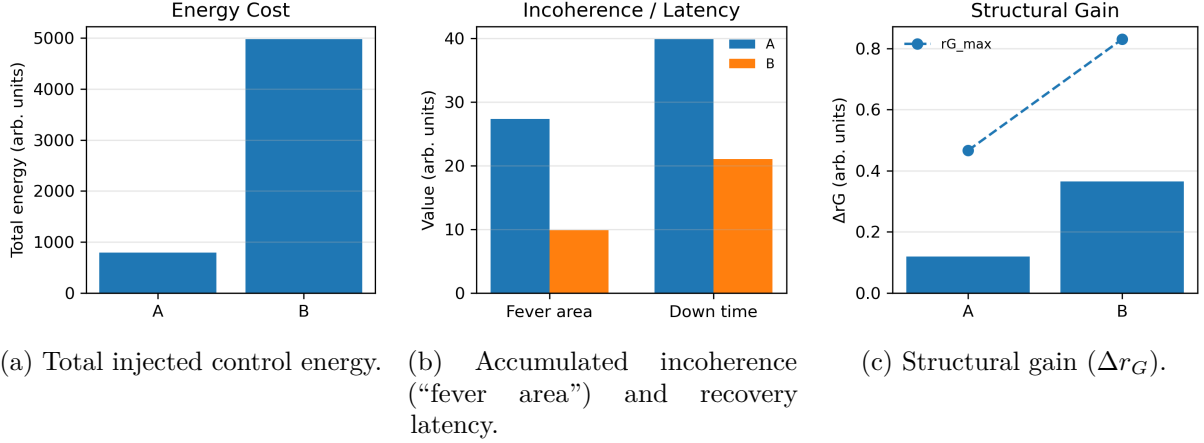


Figure 3: **A/B comparison between threshold-based coupling (A) and relaxation-based coupling (B) in Pattern-Based Computing.** Left: total injected control energy. Center: accumulated incoherence and total recovery time. Right: achieved structural gain and maximum coherence. Relaxation-based coupling requires higher energy investment, but substantially reduces incoherence persistence and enables significantly larger structural reconfiguration.

influence, permitting transient local overloads without triggering sustained congestion or instability at the network level.

Notably, global coherence and recovery indicators stabilize more rapidly under Patron-PC, despite comparable or higher instantaneous perturbations. This behavior reflects computation through relaxation toward pattern-compatible regimes rather than through continuous corrective intervention, highlighting the role of receptivity-modulated influence and controlled local decoherence in maintaining overall system stability.

6.4 Aggregated Results Across Rotations and Seeds

To assess whether the observed behaviors are structural properties of Pattern-Based Computing rather than artifacts of specific realizations, we analyze results aggregated across all temporal rotations, spatial reorderings, discretization variants, and random seeds.

For each metric, distributions are computed over the full ensemble of rotated simulations. This aggregation explicitly removes dependence on absolute timing, spatial layout, or local initialization, isolating the computational behavior induced by the pattern-based architecture itself.

Across the ensemble, Pattern-Based Computing exhibits:

- consistently lower variance in accumulated incoherence compared to reactive alignment strategies,
- bounded amplification of control influence across all seeds,
- a systematic reduction in the frequency and severity of rare congestion events,
- stable structural gains that persist under rotation and reparameterization.

Importantly, while average performance metrics (such as mean delay) exhibit overlap between strategies in some configurations, higher-order statistics reveal a clear separation. Reactive strategies display heavy-tailed distributions and sensitivity to perturbation timing, whereas Pattern-Based Computing concentrates outcomes within a narrower, more stable regime.

These aggregated results demonstrate that the advantages of PBC are not scenario-dependent. Instead, they reflect a structural computational property: the ability to constrain unstable futures

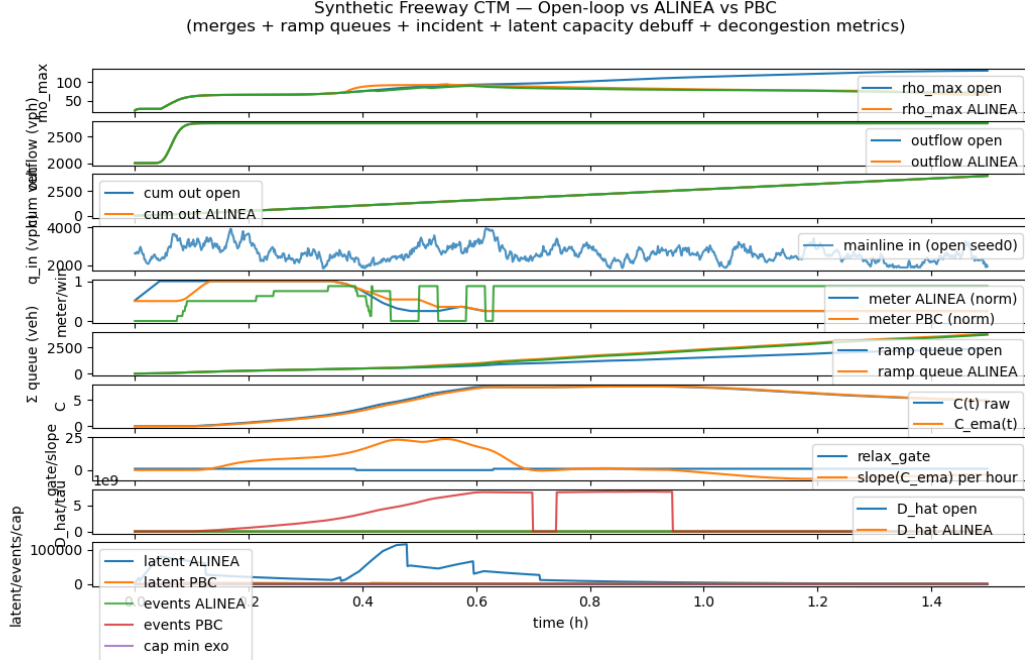


Figure 4: Comparison of open-loop operation, reactive ramp metering (ALINEA), and pattern-based coordination (Patron-PC) in a synthetic freeway CTM with merges, ramp queues, an incident, latent capacity de-buffing, and decongestion metrics. The panels report (from top to bottom) peak mainline density, outflow and cumulative throughput, mainline inflow, normalized metering signals, ramp queue evolution, coherence-related indicators, and latent congestion and event activity. While ALINEA achieves regulation through direct reactive corrections, Patron-PC modulates influence based on system receptivity, allowing temporary local incoherence while preventing persistent global congestion.

across a wide class of system realizations without requiring precise coordination or trajectory optimization.

From a computational standpoint, this ensemble behavior is essential. It confirms that Pattern-Based Computing operates on invariant properties of the system dynamics rather than exploiting accidental features of a particular configuration.

6.5 Results

Across all rotations and random seeds, Pattern-Based Computing exhibits consistent qualitative behavior:

- reduced accumulation of congestion relative to open-loop operation,
- bounded amplification of control influence,
- fewer extreme events and reduced sensitivity to perturbation timing,
- graceful degradation under adverse conditions rather than collapse.

Reactive alignment strategies achieve lower average delay in some configurations, but at the cost of higher variance and increased susceptibility to extreme events. In contrast, PBC prioritizes global stability and robustness, accepting higher instantaneous effort to reduce systemic risk.

6.6 Computational Interpretation of the Results

The results presented above should not be interpreted as performance improvements in a traffic control task, but as evidence of a distinct computational behavior. Pattern-Based Computing does not compute actions, trajectories, or optimal control signals. Instead, it computes *admissible futures* by shaping the system’s dynamical landscape and selectively restricting unstable evolutions.

From this perspective, the observed metrics correspond to computational properties rather than domain-specific objectives.

Energy as Computational Investment. The total injected control energy does not represent inefficiency in PBC, but deliberate computational effort. In relaxation-based coupling, energy is expended to reshape the pattern landscape rather than to enforce immediate corrections. This energy enables structural reconfiguration of the system, allowing it to exit incompatible regimes and explore alternative coherent configurations.

In contrast, threshold-based reactive schemes minimize short-term energy usage but lack the capacity to induce large-scale structural change, leading to persistent incoherence or brittle stabilization.

Incoherence and Fever as Computational Signals. The reduction in accumulated incoherence (“fever area”) and recovery latency under relaxation-based coupling reflects a fundamental computational mechanism. Local decoherences are not treated as errors to be suppressed, but as isolation mechanisms that prevent perturbations from synchronizing globally.

Periods of incoherence therefore act as computational signals indicating incompatibility between the current system state and the active pattern. The system does not converge to a false solution; instead, it remains dynamically active until a compatible configuration becomes accessible.

Structural Gain as the Computational Output. The structural gain metric (Δr_G) captures the core computational outcome of PBC. Rather than optimizing a scalar objective, the computation produces a qualitative reorganization of system structure, increasing global coherence capacity.

The substantially larger Δr_G achieved under relaxation-based coupling demonstrates that PBC computes structural transformations that are inaccessible to purely reactive schemes. The system effectively computes a new regime rather than a better action.

Trade-off Between Efficiency and Robustness. Taken together, these results illustrate a key principle of Pattern-Based Computing: instantaneous efficiency is traded for long-term structural robustness. Computation proceeds by allowing temporary inefficiency, local incoherence, and increased energy expenditure in order to prevent cascading failures and to enable scalable coordination.

This trade-off is not a limitation but a defining characteristic of PBC. The paradigm prioritizes the preservation and expansion of coherent futures over the optimization of immediate outcomes.

Summary. The experimental results demonstrate that Pattern-Based Computing performs computation by dynamically constraining the space of admissible system evolutions. Energy, incoherence, and structural gain are not side effects but integral components of the computational process. The observed behavior confirms that PBC operates as a computational system whose outputs are stabilized regimes and reconfigured structures, rather than explicit symbolic results.

6.7 Interpretation

From a computational perspective, these results illustrate key properties of Pattern-Based Computing:

- computation proceeds by restricting unstable futures rather than optimizing trajectories,
- local decoherences absorb perturbations without triggering global corrective action,
- adaptation of the global pattern occurs only when structural conditions permit.

The observed behavior aligns with the theoretical expectations of a pattern-based, relaxation-driven computational paradigm.

6.8 Limitations of the Example

These results do not claim superiority of Pattern-Based Computing for traffic control. The traffic system is used solely as a continuous domain that exposes coordination challenges, instability, and sensitivity to perturbations.

The relevance of the example lies in demonstrating that PBC:

- operates as a computational system,
- maintains stability under structural variation,
- and expresses failure dynamically rather than symbolically.

6.9 Summary

The traffic-based example confirms that Pattern-Based Computing can be instantiated in a real continuous system and exhibits the expected computational properties: robustness, self-correction through local decoherences, controlled structural adaptation, and resistance to error amplification.

7 Conclusions and Future Work

This paper introduced **Pattern-Based Computing (PBC)** as a relaxation-based computational framework for continuous and distributed systems, in which computation is realized through dynamic relaxation toward patterns rather than through sequential symbolic execution or explicit trajectory optimization.

PBC reframes computation as an interaction between patterns, system dynamics, and receptivity. Classical computation is not discarded, but confined to a strictly programmatic role: configuring and modifying the pattern landscape without directly executing coordination or control. Computation itself emerges from pattern-driven dynamics, enabling systems to stabilize, adapt, and self-correct without centralized decision-making.

A central contribution of this work is the identification of intrinsic computational mechanisms that distinguish PBC from existing approaches. These include error self-correction through controlled local decoherences, protected adaptation of global patterns via upward seduction and coupling windows, and the collapse of the traditional separation between program, process, and result. Together, these mechanisms support robust computation in environments where perturbations are unavoidable and enforcing discrete correctness is either infeasible or counterproductive.

The illustrative traffic-based example demonstrates that PBC can be instantiated in a real continuous system and exhibits the expected computational properties: reduced systemic fragility, bounded amplification of intervention, graceful degradation under adverse conditions, and stability under structural variation. The example is not intended as a contribution to traffic engineering,

but as evidence that PBC operates as a genuine computational framework rather than as a domain-specific control heuristic.

Several directions for future work follow naturally from this framework. First, a more formal mathematical characterization of pattern landscapes, receptivity measures, and coupling window conditions would strengthen the theoretical foundations of PBC. Second, applications to other continuous domains—such as large-scale energy systems, collective robotics, biological coordination, and socio-technical infrastructures—could further assess the generality and applicability of the framework. Third, hybrid architectures in which PBC operates alongside classical and learning-based systems may reveal complementary modes of computation and coordination.

An important and distinctive property of Pattern-Based Computing is its favorable scaling behavior. Unlike classical computational or control paradigms, whose complexity typically grows with the number of states and interactions, PBC operates on global coherence structures whose stabilizing capacity increases with system size. As scale grows, local perturbations become relatively less influential, while global patterns gain statistical and dynamical robustness.

In this regime, computation is not burdened by the expansion of the state space. Instead, larger systems provide richer interaction structures and increased opportunities for coherent relaxation. Consequently, classes of problems that are traditionally considered computationally challenging—large, distributed, and highly interconnected systems—become comparatively well suited to pattern-based computation.

Pattern-Based Computing thus offers a complementary notion of computation, particularly relevant for large-scale and safety-critical systems where robustness, coherence, and interpretable failure modes are more important than exact symbolic outcomes. By embedding computation in dynamics, stability, and pattern formation, PBC contributes to a broader understanding of computation in complex systems.

A Mathematical Foundations of Pattern-Based Computing

This appendix provides a formal mathematical framework supporting the concepts introduced in the main text. The goal is not to fully axiomatize Pattern-Based Computing (PBC), but to demonstrate that its mechanisms can be rigorously expressed within standard dynamical systems theory, while exhibiting computational properties distinct from reactive control and optimization-based paradigms.

A key distinction between Pattern-Based Computing and local reactive strategies lies in their scaling behavior. Local controllers operate through immediate feedback and therefore require increasingly precise coordination as system size grows. As the number of interacting components increases, purely local strategies tend to suffer from interference, oscillatory behavior, and cascading failures.

In contrast, PBC introduces global patterns that constrain the space of admissible system dynamics. As system scale increases, patterns absorb complexity by restricting unstable configurations rather than coordinating individual actions. As a result, PBC exhibits improved robustness and effectiveness at larger scales, while remaining resilient to local perturbations. Importantly, PBC does not outperform local strategies by being more aggressive, but by being less fragile.

A.1 System State and Intrinsic Dynamics

Let $x(t) \in \mathbb{R}^n$ denote the distributed state of a continuous system evolving according to intrinsic dynamics:

$$\dot{x}(t) = F(x(t)), \tag{2}$$

where $F : \mathbb{R}^n \rightarrow \mathbb{R}^n$ is a (possibly nonlinear) vector field encoding local interactions, physical constraints, and natural evolution.

A.2 Patterns as Dynamical Constraints

A pattern \mathcal{P} is modeled as a weak, global dynamical constraint acting on the system state. Formally, the influence of a pattern is represented through a biasing vector field:

$$\Delta_{\mathcal{P}}(x) = -\nabla\Phi_{\mathcal{P}}(x), \quad (3)$$

where $\Phi_{\mathcal{P}} : \mathbb{R}^n \rightarrow \mathbb{R}$ is a pattern potential whose low-gradient regions correspond to configurations compatible with the pattern.

Crucially, $\Phi_{\mathcal{P}}$ does not define a strict optimization objective. It introduces a structural bias that reshapes the system’s dynamical landscape without prescribing a unique trajectory or target state.

A.3 Coupled Dynamics and Receptivity

The system dynamics under pattern influence are given by:

$$\dot{x}(t) = F(x(t)) + \alpha(t) \Delta_{\mathcal{P}}(x(t)), \quad (4)$$

where $\alpha(t) \in [0, 1]$ is a time-dependent coupling factor representing system receptivity.

Receptivity is not externally imposed. Instead, it is inferred from system-level observables that reflect structural stress, coherence, or instability. Abstractly, we define:

$$\alpha(t) = g(\mathcal{S}(x(t))), \quad (5)$$

where \mathcal{S} denotes a vector of sensor functions and g is a bounded response function.

In PBC, computation consists in modulating $\alpha(t)$ —that is, regulating the admissibility of pattern influence—rather than computing explicit corrective actions.

A.4 Local Decoherence and Error Self-Correction

Let $\Omega \subset \mathbb{R}^n$ denote a local region affected by a perturbation. A local decoherence event is characterized by a temporary reduction of coupling:

$$\alpha(t)|_{\Omega} \rightarrow 0, \quad (6)$$

which isolates the perturbed region from global pattern influence.

During this phase, local dynamics are allowed to evolve independently, preventing the propagation of instability. As coherence is gradually restored, coupling is reintroduced:

$$\lim_{t \rightarrow t^+} \alpha(t)|_{\Omega} \rightarrow \bar{\alpha}, \quad (7)$$

where $\bar{\alpha}$ denotes nominal receptivity.

This mechanism constitutes an intrinsic form of error self-correction: perturbations are absorbed locally rather than corrected globally, avoiding overreaction and cascading failures.

A.5 Hierarchical Patterns and Upward Seduction

PBC operates over multiple pattern scales. Let \mathcal{P}_B , $\{\mathcal{P}_i\}$, and \mathcal{P}_G denote the lower, intermediate, and global patterns, respectively.

The global pattern evolves on a slower timescale:

$$\dot{\mathcal{P}}_G = \epsilon H(\{\mathcal{P}_i\}), \quad \epsilon \ll 1, \quad (8)$$

where H aggregates information from intermediate patterns.

Adaptation of the global pattern occurs only during *coupling windows*, defined as intervals in which a global coherence or tension measure \mathcal{C} satisfies:

$$\frac{d}{dt}\mathcal{C}(x(t)) < 0. \quad (9)$$

Outside these windows, $\dot{\mathcal{P}}_G = 0$, ensuring structural stability and preventing noise-driven drift.

A.6 Pattern Absence and Fever Dynamics

If no pattern \mathcal{P} exists such that the coupled system admits a stable attractor, then for all admissible coupling schedules $\alpha(t)$:

$$\lim_{t \rightarrow \infty} x(t) \notin \mathcal{A}, \quad (10)$$

where \mathcal{A} denotes the set of stable regimes.

In this case, the system exhibits persistent fluctuation:

$$\limsup_{t \rightarrow \infty} \|\dot{x}(t)\| > 0. \quad (11)$$

This regime corresponds to the *fever state* described in the main text. Rather than representing failure symbolically, the system expresses pattern absence dynamically through sustained instability.

A.7 Identity of Program, Process, and Result

Within this framework:

- the pattern \mathcal{P} defines the potential $\Phi_{\mathcal{P}}$ (program),
- the coupled dynamics define relaxation trajectories (process),
- the resulting stable attractors define outcomes (result).

These elements are mathematically inseparable: modifying one necessarily alters the others. This formalizes the collapse of the traditional distinction between program, execution, and output in Pattern-Based Computing.

A.8 Remarks

This appendix demonstrates that Pattern-Based Computing can be grounded within classical dynamical systems theory while introducing fundamentally different computational semantics. The formulation is intentionally non-exhaustive, leaving room for domain-specific instantiations and further theoretical refinement.

Acknowledgments

The author thanks the anonymous reviewers for their careful reading and constructive feedback, which helped improve the clarity and scope of this work.

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors. All simulations and analyses were conducted using open-source software and standard computational resources.

The author also acknowledges insightful discussions with colleagues that contributed to refining the conceptual framework presented in this paper.

References

- [1] A. M. Turing, On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42(2), 230–265 (1936).
- [2] W. R. Ashby, *An Introduction to Cybernetics*. Chapman & Hall, London (1956).
- [3] I. Prigogine and I. Stengers, *Order Out of Chaos*. Bantam Books, New York (1984).
- [4] H. Haken, *Synergetics: An Introduction*. Springer, Berlin (1983).
- [5] K. Friston, The free-energy principle: a unified brain theory? *Nature Reviews Neuroscience*, 11, 127–138 (2010).
- [6] D. P. Bertsekas, *Dynamic Programming and Optimal Control*. Athena Scientific, Belmont, MA (1995).
- [7] J. B. Rawlings, D. Q. Mayne, and M. Diehl, *Model Predictive Control: Theory, Computation, and Design*. Nob Hill Publishing (2017).
- [8] M. Mitchell, *Complexity: A Guided Tour*. Oxford University Press (2009).
- [9] S. H. Strogatz, *Nonlinear Dynamics and Chaos*. Westview Press (2015).
- [10] A.-L. Barabási, *Network Science*. Cambridge University Press (2016).