Journal of Computer and Communications, 2026, 14(10), 123-170
https://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

Ambiguity-Aware Multi-State Computation System

—Q-State Dynamics, Sparsity Dynamics, and State-Aware Graph Systems

Abhishek Kumar

Independent Researcher, Missoula, USA
Email: akumar@preferido.us

How to cite this paper: Kumar, A. (2026) Ambiguity-Aware Multi-State Computation System. *Journal of Computer and Communications*, **14**, 123-170.
https://doi.org/10.4236/jcc.2026.1410006

Received: June 8, 2026
Accepted: October 27, 2026
Published: October 30, 2026

Copyright © 2026 by author(s) and Scientific Research Publishing Inc.
This work is licensed under the Creative Commons Attribution-NonCommercial International License (CC BY-NC 4.0).
http://creativecommons.org/licenses/by-nc/4.0/



Open Access

Abstract

Modern computational systems are commonly designed around deterministic or discrete state-based models, where uncertainty is typically resolved early during state evaluation [1]-[3]. The framework further distinguishes deterministic Absolute States represented as AbsA = (−1, 0, +1) and equivalently expressed as |A| = (−1, 0, +1) from derived ambiguity-preserving Q-States, enabling state-aware evaluation without forcing immediate collapse into rigid binary interpretation. [A: Absolute State]. However, many real-world systems operate under conditions where ambiguity persists, evolves, and directly influences system behavior [A1: Ambiguity]. This paper introduces an ambiguity-aware computation method operating within a Multi-State Computational Model, where Q-States are treated as a first-class computational state construct [A2: Q-State]. Instead of forcing immediate resolution, the proposed computational method preserves, interprets, and propagates ambiguity through a structured Transformation Pipeline. The pipeline integrates State Derivation, Q-State Identification, and State-Aware Graph Evaluation. [A3: State-Aware]. A key conceptual distinction is established between Computing State and Computational State, enabling ambiguity-aware evaluation under structured operational constraints. [A4: Computing vs Computational State]. The proposed computational methods incorporate Lattice-Based Representation and Node State Classification to model system behavior under ambiguity [A5: Node-State]. Sparsity Dynamics and Path Validity are introduced as core evaluation mechanisms, supported by Boundary Conditions and Hysteresis Dynamics to maintain stability, continuity, and evaluation consistency [A6: Sparsity, Path-Validity]. Within the evaluated simulation configurations, results exhibited controlled Q-State propagation and bounded confidence-oriented evaluation behavior under the defined ambiguity constraints [A7: Boundary-Conditioned Hysteresis]. This approach provides a foundation for modeling

complex systems where uncertainty is not prematurely eliminated, but systematically represented, evaluated, and incorporated into state-aware computational evaluation. [A8: Graph-Based Systems].

Keywords

Ambiguity-Aware Computation, State-Aware Computation, Absolute-State, Q-State, Multi-State Computation, Boundary Value Problem, Node-State, State-Space Modeling and Validity, Control-Theoretic State Modeling, State-Space Behavior, Sparsity Dynamics, Computing State, Computational State, Ambiguity Preservation, Propagation, State-Aware Graph Systems, Hysteresis, Bit-Oriented Q-State Evaluation, Non-Collapse Principle, Research Simulations and Observations

1. Introduction

Many computational systems are designed to operate under deterministic or otherwise well-specified conditions, where input conditions are expected to transition toward stable and interpretable states [1]-[3]. The deterministic and discrete-state baseline formulations considered in this study do not preserve the Q-State construct defined here as a persistent first-class evaluative state [3] [4]. However, real-world systems may operate under incomplete, conflicting, evolving, or partially defined conditions in which ambiguity itself may remain operationally significant throughout system evaluation [5] [6].

In many practical computational environments, systems must operate under incomplete, conflicting, evolving, or partially interpretable conditions [5]-[7]. Under such circumstances, enforcing premature state resolution may eliminate operationally relevant ambiguity, reduce structural interpretability, or introduce instability within system evaluation [6]. These limitations motivate the need for computational frameworks capable of explicitly representing ambiguity and evaluating its influence as part of the system's operational and interpretive behavior.

The Ambiguity-Aware Computation model addresses these limitations by introducing Q-States within a Multi-State Computation structure capable of representing and evaluating operational ambiguity [8]. Rather than forcing ambiguity toward immediate resolution, the proposed model permits ambiguity to be preserved, interpreted, and propagated through a structured evaluative progression under defined evaluation conditions.

A system is treated as a structured collection of interacting computational state-aware constructs whose states, relationships, and evaluative behavior are represented and interpreted within a Multi-State Computation structure.

Within this system structure, a node represents a discrete computational state-aware construct that participates in evaluative transitions, relational interpretation, and computational path participation. Each node is associated with a Node State that may exist in defined, undefined, or ambiguity-preserving (Q-State) con-

ditions depending on the applicable evaluative constraints and operational conditions.

The System State represents the aggregate evaluative interpretation of interacting Node States within the computational structure. Node State Validity determines whether a node participates in structurally admissible computational paths under defined ambiguity and evaluative constraints. These foundational definitions establish the basis for consistent operational interpretation throughout the ambiguity-aware computational model and are formalized in subsequent sections of the paper.

A key conceptual distinction in this work is the separation between Computing State and Computational State. Computing State refers to observable execution-level system conditions, while Computational State represents the evaluative interpretation of those conditions within the ambiguity-aware computational structure. This distinction enables controlled Ambiguity Propagation under defined evaluative constraints until applicable Ambiguity Resolution Conditions are satisfied.

To support ambiguity-aware computational interpretation, the proposed model incorporates state-aware representations and evaluative mechanisms that enable consistent analysis of ambiguous system behavior. These include state-oriented representations, path-aware evaluation, and controlled ambiguity propagation under defined evaluative constraints.

Evaluative mechanisms such as Sparsity Dynamics and Path Validity are derived from Node State participation, ambiguity-aware evaluative constraints, and structurally admissible path conditions that support computational interpretation and consistent evaluative stability. These operational behaviors are further conditioned through Boundary Conditions and Hysteresis Behavior, enabling controlled stabilization and reducing premature evaluative state transitions [9]-[12].

The remainder of this paper presents the conceptual foundations, structural organization, and evaluative validation of the Ambiguity-Aware Computation model, including simulation-based analysis evaluating Q-State propagation and bounded confidence-oriented behavior under the ambiguity-aware conditions defined in this study.

The paper is organized to progressively establish the conceptual, structural, and evaluative foundations of ambiguity-aware computation. Section 2 compares traditional computational models with ambiguity-aware computational interpretation and outlines the limitations addressed by the proposed model. Section 3 establishes foundational definitions, including Computing State, Computational State, Node State, and Q-State representation. Section 4 introduces ambiguity as an operational construct and examines its role within ambiguity-aware evaluative progression.

Subsequent sections examine the structural and evaluative mechanisms of the proposed model, including State-Aware Graph Models, Lattice-Based Representation, Sparsity Dynamics, Path Validity, Boundary Conditions, and Hysteresis

Behavior. The paper then presents simulation and experimental observations demonstrating ambiguity propagation, evaluative stabilization, and structurally admissible computational behavior under ambiguity-aware conditions. Later sections discuss application-oriented scenarios and cross-domain evaluative validation across multiple computational contexts.

2. Comparison with Traditional Computational Models

For the deterministic and discrete-state computational formulations used as baseline models in this study, system behavior may be represented as a mapping:

$$f : X \to Y . \tag{1}$$

where inputs $(x \in X)$ are mapped to outputs $(y \in Y)$ under specified conditions. In the baseline formulations considered here, multi-state behavior may extend the mapping to a finite state space $(S = \{s_1, s_2, \cdots, s_k\})$ while unresolved ambiguity is not represented as a persistent first-class evaluative state [2] [3].

Within those baseline formulations, ambiguity or unresolved conditions are commonly resolved, approximated, excluded, or represented through mechanisms other than the persistent Q-State construct proposed here. For the specific comparison developed in this study, early resolution can discard distinctions that the ambiguity-aware formulation is designed to preserve under incomplete, conflicting, or evolving conditions [5]-[7].

The deterministic baseline models considered in this comparison provide structured approaches for evaluating system state, but they do not explicitly implement the persistent ambiguity-preserving Q-State semantics defined in this work. This distinction is material when admissible ambiguity is intended to remain an operational evaluative condition that influences subsequent graph-connected evaluation. The comparison made here is therefore limited to the specified baseline formulations and does not imply that all conventional, probabilistic, fuzzy, multi-valued, or uncertainty-aware computational methods resolve ambiguity in the same manner [3]-[5].

The Ambiguity-Aware Computation framework addresses this limitation by extending the state space to explicitly include Q-States (Ambiguous States). This allows ambiguity to be preserved and evaluated rather than eliminated. Consequently, system behavior is determined not only by resolved states but also by the presence, structure, and propagation of ambiguity across nodes and paths within the system [8].

As illustrated in **Figure 1**, the distinction between traditional models and the Ambiguity-Aware Computation framework extends across state representation, path dynamics, and system evaluation mechanisms.

Rather than forcing ambiguity to collapse prior to computation, the proposed framework evaluates state behavior under admissible structural constraints where ambiguity itself participates in evaluative propagation.

From a formulation perspective, this shifts the system from a purely determin-

istic mapping to a constrained evaluation process governed by structural conditions:

$$F(s,t;B)=0. \tag{2}$$

where s represents system state behavior, t represents temporal or propagation evolution, and B represents Boundary Conditions governing admissible state behavior. This formulation aligns with the Boundary Value Problem (BVP) perspective introduced in Section 3, where system behavior is determined not only by local inputs but also by global structural constraints [2] [9] [11].

	Traditional Graph Computation	Vorgentia Ambiguity-Aware Model
Node State Assumption	Deterministic/ Binary(+1, 0)	Multi-dimensional/ Ambiguity-Aware(+1, 0, -1, q)
Path Dynamics	Linear/Pre-computed	Path-dependent/ Statistically Sampled
Sparsity Measurement	Distorted by forced assumptions	Preserved via isolation of ambiguous states
System Tooling	Static logging	Lattice-based simulation environments

The Vorgentia research framework introduces an explicit separation between classical node states and ambiguous classifications to measure true system stability.

Figure 1. Comparison between selected deterministic or pre-defined-state baseline formulations and the Ambiguity-Aware Computation framework. The baseline comparison represents approaches in which the Q-State construct defined in this work is not preserved as a first-class evaluative state, whereas the proposed framework explicitly preserves and evaluates Q-States under admissible structural conditions. The comparison is limited to the formulations evaluated in this study.

3. Foundational Definitions

This section establishes the formal foundation of the Ambiguity-Aware Computation framework. The formulation is grounded in a Boundary Value Problem (BVP) perspective, where system behavior is derived under structural constraints rather than resolved through immediate deterministic mapping [8]-[10].

In contrast to the deterministic state-assignment baseline considered in this study, the proposed framework derives admissible state-space behavior under boundary-constrained ambiguity propagation. This permits ambiguity to persist as a structurally valid computational condition when the defined admissibility requirements are satisfied [8].

Within the proposed framework, state and state-space behavior are not treated as fixed computational inputs, but as dynamically derived admissible conditions governed through propagation behavior, neighboring interactions, hysteresis, boundary conditions, and ambiguity-aware evaluation constraints [8] [11].

3.1. Absolute State and Q-State Preservation over Binary-Oriented Architectures

To evaluate how deterministic Absolute States and ambiguity-preserving Q-States

behave under binary-oriented computational architectures, a computational preservation analysis was performed using Codex-assisted simulations, reconstruction analysis, address-space evaluation workflows, and preservation modeling on May 16, 2026 by Abhishek Kumar, Vorgentia Research (VORG/Preferido). The analysis focused on determining how Absolute States and Q-States are represented, identified, preserved, and reconstructed over binary-oriented bit-address systems [8] [13].

Conventional computational architectures fundamentally operate over deterministic binary-oriented physical encoding structures:

$$B = \{0,1\}. \tag{3}$$

with:

$$2^1 = 2. \tag{4}$$

representing two deterministic physical occupancy conditions over binary-oriented addressable memory systems.

Accordingly, deterministic binary-oriented architectures preserve addressable occupancy conditions at the physical encoding layer [1] [14].

However, the states used throughout this work do not represent ordinary binary literals alone. Instead, the proposed framework interprets these states as semantic computational admissibility structures evaluated over binary-oriented address-space architectures. Consequently, the symbolic representations:

$$-1, 0, +1, (1,0), (0,1), (1,-1). \tag{5}$$

should not be interpreted as isolated Boolean values, arithmetic binary literals, or ordinary vector symbols alone. Rather, these representations correspond to semantic computational state conditions, admissibility-preserving structures, and relational continuity configurations evaluated operationally over binary-oriented computational architectures.

The proposed framework therefore distinguishes deterministic physical occupancy preservation from higher-order semantic computational reconstruction. Binary-oriented architectures continue to preserve deterministic addressable occupancy structures, while ambiguity-preserving computational interpretation emerges through admissible relational continuity evaluation over preserved computational state structures.

The proposed framework extends semantic computational admissibility through deterministic Absolute-State structures represented as:

$$Abs_A = \{-1, 0, +1\}. \tag{6}$$

where:

$$|-1| \rightarrow \text{Active / Undefined}. \tag{7}$$

$$|0| \rightarrow \text{Inactive / Undefined}. \tag{8}$$

$$|+1| \rightarrow \text{Active / Defined}. \tag{9}$$

The Codex-assisted preservation analysis evaluated the following Absolute-State sequence:

$$|-1|,|0|,|-1|. \tag{10}$$

which under deterministic semantic equivalence reduction becomes:

$$1,0,1. \tag{11}$$

and is preserved within binary-oriented address-space memory as:

$$M[0]=1, M[1]=0, M[2]=1. \tag{12}$$

The corresponding deterministic semantic equivalence relation is defined as:

$$x \sim y \Leftrightarrow |x|=|y|. \tag{13}$$

resulting in:

$$[-1] \sim [+1]. \tag{14}$$

under Absolute-State semantic equivalence preservation.

The Codex simulations demonstrated that Absolute-State preservation behaves as deterministic scalar semantic occupancy preservation over binary-oriented address-space architectures because the intended preservation target is the semantic equivalence class itself rather than exact signed identity preservation.

The preservation analysis demonstrated:

$$E_{abs}(-1,0,-1) \to (1,0,1). \tag{15}$$

followed by deterministic reconstruction:

$$D_{abs}(1,0,1) \to (|1|,|0|,|1|). \tag{16}$$

where:

E_{abs} represents Absolute-State occupancy encoding,

D_{abs} represents deterministic semantic reconstruction over equivalence classes.

Magnitude-only binary-oriented storage preserves the occupancy-equivalence class of an Absolute State. Under this equivalence, −1 and +1 map to the same occupied representation. It does not preserve the exact signed state when −1 and +1 carry distinct operational meanings. Exact signed reconstruction therefore requires an additional sign or state-identity indicator.

The Codex-assisted analysis then evaluated the following Q-State structures:

$$q_1 = (1,0). \tag{17}$$

$$q_2 = (0,1). \tag{18}$$

$$q_3 = (1,-1). \tag{19}$$

Unlike Absolute States, the Q-State structures preserve addressed signed-component continuity and admissible relational structure among addressed computational components.

The computational preservation analysis demonstrated that Q-State structures

do not behave as ordinary scalar occupancy states. Instead, the Q-State structures preserve relational admissibility and signed computational continuity across addressed components.

Magnitude-only occupancy encoding of the Q-State structures produces:

$$E_q(1,0)=(1,0). \tag{20}$$

$$E_q(0,1)=(0,1). \tag{21}$$

$$E_q(1,-1)=(1,1). \tag{22}$$

The applicable restricted preservation-analysis domain is:

$$Q_v=\{(1,0),(0,1),(1,-1)\}. \tag{23}$$

On this declared domain, the three magnitude-encoded patterns are distinct. Therefore, magnitude-only encoding is injective on the restricted set:

$$q\neq q' \Rightarrow E_q(q)\neq E_q(q'), q,q'\in Q_v. \tag{24}$$

If the decoder knows that only members of Q_v are admissible, the encoded pattern (1, 1) uniquely identifies $q_3=(1,-1)$:

$$D_q\left((1,1)\,|\,Q_v\right)=(1,-1). \tag{25}$$

Magnitude-only representation does not generally preserve signed component identity, relational orientation, provenance, or evaluation context over a broader signed-pair domain. For such a domain, represented by Q-tilde, decoding is not necessarily unique:

$$D_q\left((1,1)\,|\,\tilde{Q}\right) \text{ is not unique without additional constraints.} \tag{26}$$

This broader-domain limitation must be distinguished from reconstruction on the declared restricted domain Q_v. Where signed identity or relational context is operationally significant, reconstruction requires the applicable admissibility constraint or additional state-relation metadata.

The bounded reconstruction distinction does not imply computational invalidity or operational failure. It identifies which information is preserved by occupancy-only encoding and which information depends on domain knowledge or additional metadata.

Occupancy equivalence and exact signed-state identity are therefore treated as separate preservation targets.

An increase in possible reconstructions over a broader signed domain should not be interpreted as unrestricted ambiguity amplification or uncontrolled entropy expansion.

Operational admissibility instead depends on preservation fidelity, valid relational continuity, admissible graph-path propagation, and boundary-conditioned computational evaluation.

The framework consequently distinguishes admissible ambiguity-preserving continuity from uncontrolled ambiguity growth; neither additional reconstruction possibilities nor ambiguity preservation alone establishes computational cor-

rectness, stability, or operational validity.

The preservation-oriented computational analysis further motivates quantitative evaluation of ambiguity-preserving reconstruction behavior using structured sparsity density measurements, admissible reconstruction entropy observations, and confidence-oriented operational continuity analysis over preserved computational state structures.

For the declared Q_v domain, exact pattern reconstruction requires knowledge of the admissible domain. Over a broader signed domain, exact Q-State reconstruction additionally requires preservation of signed addressed-component continuity. The canonical restricted-domain structures are:

$$q_1 = (1,0), q_2 = (0,1), q_3 = (1,-1). \tag{27}$$

with relational continuity expressed as:

$$q_3 = q_1 - q_2. \tag{28}$$

Within the stated binary-oriented representation and preservation analysis, this expression illustrates a component relationship among the specified Q-State representations. It does not define a general rule for deriving a Q-State from existing Q-States. The analysis concerns preservation of addressed signed-component continuity and admissible relational structure.

Within the declared domain, the relation identifies the canonical signed dependency among the addressed components rather than an ordinary arithmetic reduction alone.

The above relation represents admissible signed relational dependency preservation among addressed computational components rather than ordinary arithmetic reduction alone.

The preservation-oriented computational analysis further suggests that operational continuity within graph-based computational systems depends not solely on deterministic occupancy preservation, but also on admissible relational continuity among propagated node-state structures during computational path evaluation.

Figure 2 illustrates the integrated operational preservation blueprint derived from the Codex-assisted preservation analysis, showing the progressive relationship among binary-oriented occupancy preservation, semantic reconstruction, ambiguity-preserving Q-State continuity, structured sparsity propagation, and operational admissibility behavior under boundary-conditioned computational evaluation.

The blueprint progressively derives the operational transformation flow:

$$S_1 \to \Phi_a \to Q \to \Phi_i \to S_2. \tag{29}$$

where:

S_1 represents deterministic binary-oriented computing-state preservation,

Φ_a represents ambiguity-aware reconstruction evaluation and semantic equivalence analysis,

Q represents the admissible ambiguity-preserving Q-State region,

Φ_i represents boundary-constrained relational interpretation and sparsity-aware propagation continuity,

S_2 represents structurally admissible operational continuity or retained admissible ambiguity.

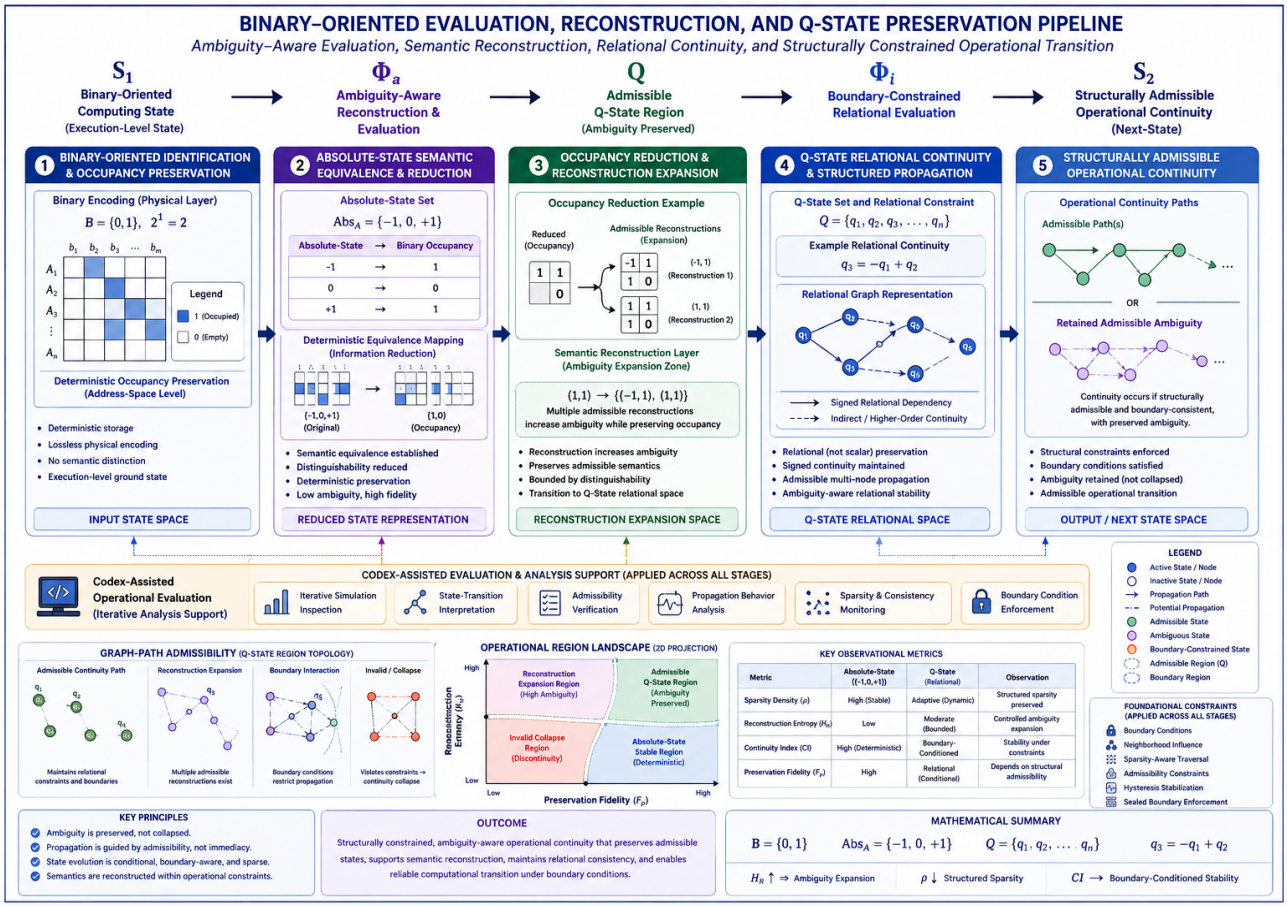


Figure 2. Integrated Operational Preservation Blueprint illustrating Binary-Oriented Evaluation, Semantic Reconstruction, Occupancy Reduction, Q-State Relational Continuity, Structured Sparsity Propagation, and Operational Admissibility behavior over deterministic binary-oriented computational architectures.

Absolute-State preservation is represented through:

$$Abs_A = \{-1, 0, +1\}. \tag{30}$$

with occupancy-oriented preservation mappings:

$$|-1| \to 1, |0| \to 0, |+1| \to 1. \tag{31}$$

Within the declared restricted domain Q_v, the magnitude pattern (1, 1) uniquely identifies $q_3 = (1, -1)$ when the admissibility constraint is known:

$$D_q\left((1,1) \mid Q_v\right) = (1, -1). \tag{32}$$

The blueprint additionally illustrates Q-State relational continuity preservation (see Equation (27) and Equation (28)):

The lower operational layers further illustrate: i) structured sparsity-aware graph propagation, ii) admissible path continuity, iii) reconstruction entropy observations, iv) preservation fidelity behavior, v) confidence-oriented operational stability, and vi) collapse versus non-collapse admissibility regions identified during Codex-assisted preservation analysis.

New Edition Clarification. Figure 2 is retained in its originally published form to preserve the historical research record. For the present edition, the canonical restricted-domain representation is $q_3 = (1,-1)$, with relational expression $q_3 = q_1 - q_2$. Within the declared admissible set $Q_v = \{(1,0),(0,1),(1,-1)\}$, magnitude-only encoding distinguishes the three listed patterns when the admissible domain is known. Broader signed-state or relational reconstruction may require sign, admissibility, provenance, or state-relation context. This clarification supersedes any inconsistent sign, relational expression, or unrestricted reconstruction interpretation appearing within the originally published figure.

The preservation analysis therefore identified a major computational distinction between Absolute-State and Q-State preservation over binary-oriented architectures:

- Absolute-State preservation behaves as deterministic scalar semantic equivalence preservation.
- Q-State preservation behaves as relational signed-component identity preservation.

The computational preservation analysis further demonstrated:

- Absolute States remain reversible over deterministic semantic equivalence classes using occupancy-only binary-oriented storage.
- Q-State patterns remain reconstructable on the restricted Q_v domain when that admissibility constraint is known; broader signed relational reconstruction may require sign, provenance, or state-relation metadata.
- Occupancy-only binary reduction preserves deterministic class continuity but does not preserve exact signed relational admissibility structure.

Consequently, binary-oriented occupancy architectures preserve Absolute-State occupancy-equivalence classes, while exact signed-state identity and broader Q-State relational continuity depend on the preservation target, declared admissible domain, and any required state-relation context.

Accordingly, ambiguity-preserving computational evaluation within the proposed framework operates as a semantic reconstruction layer over deterministic binary-oriented preservation architectures.

The preservation analysis therefore motivates the following architectural question:

Can ambiguity-preserving relational computational continuity remain operationally admissible over deterministic binary-oriented address-space architectures without invalidating computational propagation?

The proposed framework indicates that ambiguity-preserving computational continuity can remain operationally admissible within deterministic binary-ori-

ented architectures when ambiguity is explicitly represented as a first-class computational state and propagated through state-aware relational evaluation rather than being immediately reduced to binary outcomes [8].

Table 1 summarizes the major computational distinctions identified during the Codex-assisted preservation analysis.

Table 1. Computational preservation differences between absolute states and Q-states.

Criterion	Absolute State	Q-State
Preservation Target	Occupancy-equivalence class or exact signed identity, as specified	Admissible relational and signed-component identity
Binary Occupancy Sufficient	Yes for occupancy equivalence; no for exact signed identity	Sufficient to distinguish restricted Q_v patterns when the domain is known; insufficient for general signed or contextual reconstruction
Direct Reconstruction	Equivalence-class reconstruction from magnitude alone	Restricted-domain reconstruction when Q_v is known; otherwise relational metadata may be required
Sign Preservation Required	Required for exact signed identity	Required when signed component identity is operationally significant
Address-Space Behavior	Scalar occupancy representation	Multi-component relational representation
Continuity Preserved	Occupancy-class continuity unless sign is retained	Restricted-pattern continuity; broader relational continuity requires context
Reversibility	Reversible over the declared equivalence classes	Reversible on restricted Q_v; conditional over a broader signed domain
Computational Character	Many-to-one magnitude reduction when sign is discarded	Ambiguity-sensitive structural preservation

Subsequent Relational Node-State Derivation and Canonical Representation Research

Subsequent Vorgentia Research builds and derivational analyses extend the earlier binary-representation study by distinguishing a single node, independently evaluated separate nodes, their respective Node States, and graph-connected Path Validity. These subsequent derivations were not results produced by the May 16, 2026 Codex-assisted preservation simulation and are identified separately to maintain research provenance [15].

Research on cryptographically supported human/AI text provenance provides an adjacent evidence-integrity context [16]; it is not treated as an equivalent derivation or validation of the relational Q-State model.

$$Q_t^{(U,I)} = \left(s_U^{(t)}, s_I^{(t)}\right). \tag{33}$$

here, U identifies the user or session node; I identifies an independently evaluated node; t identifies the evaluation time or transition step; $s_U^{(t)}$ and $s_I^{(t)}$ identify

the respective Node States; and the coordinate order is fixed so that the first component always represents U and the second always represents I.

The initial ordered relational configuration is:

$$Q_{t_0}^{(U,I)} = (-1,0). \tag{34}$$

This configuration preserves the user/session node at −1 while the independently evaluated node remains at 0. When the independent node becomes active and defined, the ordered relational configuration progresses to:

$$Q_{t_1}^{(U,I)} = (-1,+1). \tag{35}$$

The corresponding change occurs in the independent node while the user/session Node State remains preserved:

$$s_I : 0 \to +1, s_U = -1 \text{ remains preserved}. \tag{36}$$

This progression does not represent one node transitioning from −1 to 0, and it does not change the user/session node to +1. It preserves the identities and states of two ordered nodes. The user remains within an ambiguity-preserving relational Q-State context while the independently evaluated node is presented in its active and defined +1 state.

For this subsequent relational derivation, the valid derived relational set is defined as:

$$Q_{UI} = \{(-1,0),(-1,+1)\}. \tag{37}$$

The notation Q_{UI} identifies the user-to-independent-node relational Q-State class. It is distinct from the restricted Q-State set used in the earlier preservation analysis:

$$Q_V = \{(1,0),(0,1),(1,-1)\}. \tag{38}$$

The notation Q_V identifies the restricted admissible set used by the earlier representation and reconstruction analysis. Q_V and Q_{UI} serve different representational purposes and are not treated as one undifferentiated Q-State domain.

A single node may possess an admissible Node State without independently establishing a computational path. Likewise, multiple independently admissible nodes do not establish Path Validity merely because their individual Node States are valid. Presentation of the active independent node to the user/session context requires an admissible connecting relation under the applicable graph structure, Boundary Conditions, and state-transition constraints.

For an ordered candidate path:

$$\Gamma = (v_0, v_1, \cdots, v_k), k \geq 1. \tag{39}$$

The notation Γ identifies the ordered candidate path; v_i identifies the computed node; and k identifies the number of connecting transitions. The symbol Γ is used for an ordered graph path and is unrelated to the numerical constant

π. Path Validity is defined by:

$$V_{path}(\Gamma) = \wedge_{i=0}^{k} V_N(v_i) \wedge \wedge_{i=0}^{k-1} V_E(v_i, v_{i+1}) \wedge V_B(\Gamma) \wedge V_T(\Gamma). \quad (40)$$

here, $V_{path}(\Gamma)$ identifies Path Validity; $V_N(v_i)$ identifies the admissibility of each participating Node State; $V_E(v_i, v_{i+1})$ identifies the admissibility of each connecting edge; $V_B(\Gamma)$ identifies satisfaction of the applicable Boundary Conditions; $V_T(\Gamma)$ identifies admissibility of the ordered state-transition relationships; the wedge symbol denotes logical conjunction; and $k \geq 1$ requires at least one connecting transition. A valid independent Node State therefore does not, by itself, establish a valid graph path.

The relational progression is represented as:

$$Q^{(U,I)} : (-1, 0) \to (-1, +1). \quad (41)$$

This progression is represented operationally within an active Vorgentia Research build [15], as shown in **Figure 3**, through preservation of Q while a requested +1 state remains subject to defined browser-local transition functions. The implementation view records Initial State −1, requested Next State +1, preserved Q, and Final State +1 pending completion. The interface-level representation and the ordered-pair derivation describe related but distinct layers: the interface exposes the preserved operational transition, while the derivation identifies the separately preserved states of the participating nodes.

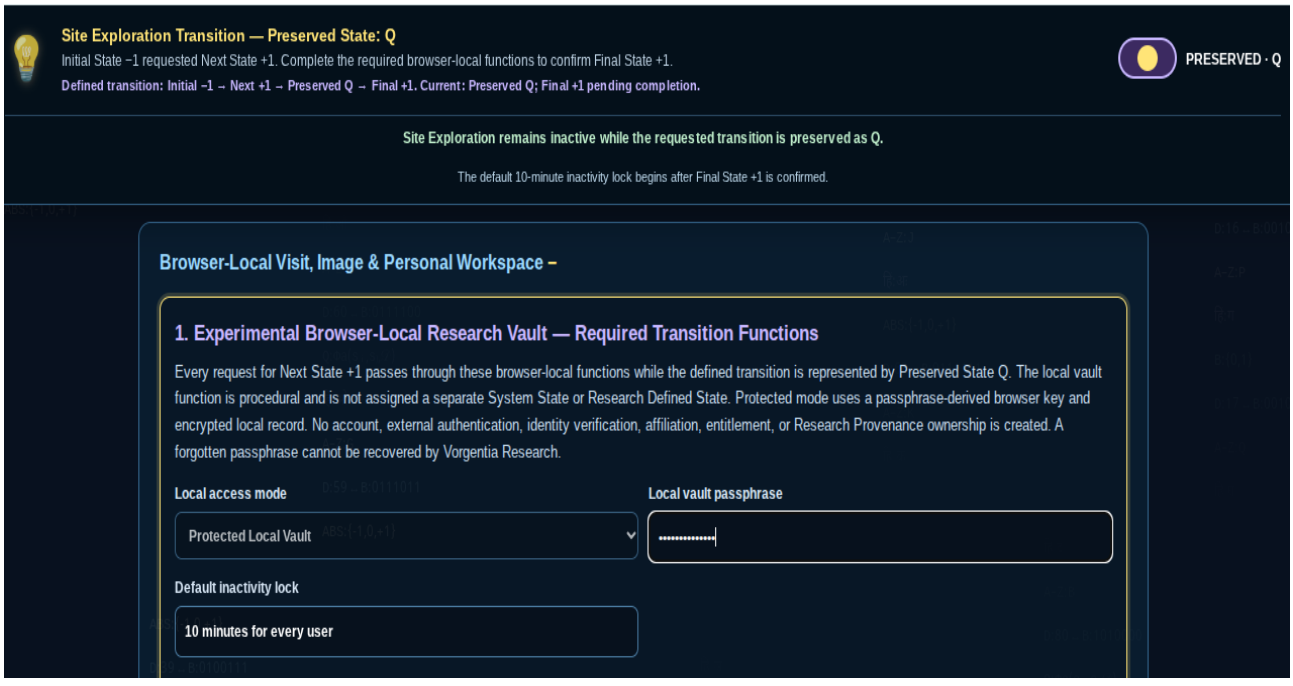


Figure 3. Research-build representation of a preserved Q-State transition. The interface records Initial State −1, requested Next State +1, preserved Q, and Final State +1 pending completion of defined browser-local transition functions. The image illustrates an implementation-level research use case; the ordered relational representation $Q^{(U,I)} : (-1, 0) \to (-1, +1)$ is the associated subsequent derivational interpretation and is not asserted as text directly displayed by the interface. Source: author-controlled Vorgentia Research public build documented in [15].

Primary Subsequent Research Question. How does the required computational representation change across Absolute-State and valid derived Q-State evaluations when the canonically defined information quantity is held fixed?

This question serves as the primary theme for upcoming Vorgentia Research discussions, working sessions, simulations, and the developing research paper with the initial working title Computational Representation Across Absolute and Derived Q-States Using a Canonically Defined Information Quantity [15]. The subsequent research is intended to examine representation requirements, metadata requirements, state-identity preservation, relational ordering, and computational measurements under a fixed information quantity. The present subsection records the initial use case and derivational continuity; it does not present the subsequent paper's experimental results as completed findings of the current paper.

The Node State and State Space, Computing State and Computational State, Node State Validity, Ambiguity Propagation, and Ambiguity Path Evaluation sections apply or further bound these distinctions. Later references should rely on this subsection's definitions rather than redefining Q_V, Q_{UI}, or the ordered relational progression independently.

3.2. Boundary Value Problem (BVP) Formulation

Let the system be defined over a state space S and evaluation domain Ω [2] [9] [11]. The system behavior is described by:

$$F\left(s\left(x,t\right),\nabla s\left(x,t\right),t\right)=0, x\in\Omega. \tag{42}$$

subject to Boundary Conditions:

$$B\left(s\left(x,t\right)\right)=0, x\in\partial\Omega. \tag{43}$$

where:

- $s\left(x,t\right)$ represents the evolving state function over structural space and temporal progression,
- $\nabla s\left(x,t\right)$ represents state variation and admissible propagation behavior,
- $F\left(\cdot\right)$ represents the governing system evaluation operator,
- $B\left(\cdot\right)$ represents Boundary Conditions constraining admissible state evolution,
- Ω represents the admissible evaluation domain,
- $\partial\Omega$ represents the structural boundary of the evaluation domain.

The deterministic and finite-state baseline formulations considered in this study derive node states through predefined or resolution-dependent input conditions. In these selected formulations, unresolved ambiguity is not preserved using the persistent Q-State semantics proposed here. This limits their suitability as direct representations of the specific ambiguity-preservation behavior evaluated in this work under incomplete, conflicting, or structurally evolving conditions [2]-[5].

The proposed BVP formulation addresses this limitation by deriving admissible state-space behavior under structural constraints rather than enforcing immediate

deterministic resolution. This enables admissible ambiguity propagation, boundary-conditioned evaluation, and controlled state persistence within the computational process [8].

Within the evaluated simulation configurations, rigid deterministic state assignment used as a comparison baseline was associated with premature ambiguity resolution, altered sparsity behavior, propagation instability, and reduced preservation of the structural distinctions tracked by the ambiguity-aware model [13] [17] [18].

In contrast, BVP-driven simulation and modeling indicated improved ambiguity preservation, propagation stability, admissible state retention, and structural sparsity consistency across the evaluated computational scenarios [8].

Within the proposed framework, derivatives are not used solely to model numerical system motion, but to characterize admissible ambiguity propagation, boundary-conditioned transition behavior, hysteresis influence, and structurally constrained state evolution [2] [8] [9] [11].

3.2.1. First-Order State Evolution

First-order derivative behavior models immediate state evolution and local evaluative propagation dynamics [10] [11]:

$$\frac{\mathrm{d}s}{\mathrm{d}t}. \tag{44}$$

This represents:

- local state transition behavior,
- admissible propagation sensitivity,
- neighboring node influence,
- immediate ambiguity evolution,
- localized evaluative response behavior.

First-order behavior is useful for modeling localized propagation response, adjacent node interactions, and admissible evaluative transition behavior within ambiguity-aware graph structures [8] [19] [20].

3.2.2. Second-Order State Evolution

Second-order derivative behavior models structural propagation response, hysteresis influence, stabilization behavior, and admissible ambiguity persistence [9]-[11]:

$$\frac{\mathrm{d}^2 s}{\mathrm{d}t^2}. \tag{45}$$

This represents:

- propagation acceleration and deceleration,
- oscillatory or retained ambiguity behavior,
- structural stabilization,
- hysteresis-driven transition resistance,
- delayed deterministic collapse behavior,
- boundary-conditioned propagation influence.

Simulation-based evaluation indicated that second-order formulations may

support greater admissible ambiguity-propagation stability within the evaluated configurations and structural sparsity preservation compared with rigid deterministic state-assignment approaches [8] [13].

The use of first-order and second-order derivative formulations enables the framework to transition from pre-defined state assignment toward derived state-space behavior under Boundary Conditions [8].

3.2.3. State-Space Behavior

Classical state-space systems are commonly expressed as:

$$\dot{x}(t) = A_s x(t) + G_u u(t). \tag{46}$$

here, $x(t)$ represents the state vector or state-space representation; A_s represents the state-transition matrix; $u(t)$ represents an external or controlled input vector; and G_u represents the input-influence matrix. The symbol $B(\cdot)$ remains reserved for the Boundary Condition operator defined above. This state-space expression is used as a representational analogy for constrained computational state evolution and does not imply identity with a conventional continuous-time control system or completed physical parameter identification [10] [11].

The state-space symbols introduced here are distinct from the BVP symbols: $x(t)$ is the state-space vector, A_s governs state transition, $u(t)$ is the applied or modeled input, and G_u describes input influence, whereas $B(\cdot)$ remains the Boundary Condition operator. This separation prevents one symbol from representing both a boundary operator and an input matrix.

The proposed framework extends beyond traditional deterministic state-space evaluation by allowing ambiguity itself to become an admissible operational evaluative state. Under this formulation, state evolution depends not only on local node inputs but also on propagation dynamics, neighboring interactions, hysteresis behavior, ambiguity preservation, and boundary-conditioned admissibility. A system may therefore exhibit stable execution-level behavior while simultaneously remaining computationally ambiguous under evaluative constraints [8].

Figure 4 illustrates how discrete graph structures are transformed into continuous evaluation fields under BVP-constrained state-space behavior. The formulation enables ambiguity regions to emerge operationally through derivative-driven propagation rather than rigid deterministic assignment [8].

Figure 5 demonstrates how ambiguity-aware propagation may be modeled in fragmented byte-oriented evaluation environments. Under this formulation, ambiguity is preserved and evaluated structurally rather than collapsed prematurely into deterministic interpretation states [8].

Figure 6 illustrates a proposed PCB/USB propagation phase under comparator, differential, continuous-field, and hysteresis constraints. Physical instrumentation and hardware measurements remain planned and are not presented as completed experimental results in this edition [8] [10].

Figure 7 illustrates the proposed terminal evaluation behavior under sealed

Boundary Conditions within the PCB/USB model. Once the modeled system constraints are satisfied, further propagation and recursive re-evaluation are suppressed while the operational Q-State remains preserved. This figure represents a conceptual or simulation model; it does not by itself establish completed physical hardware measurements [8].

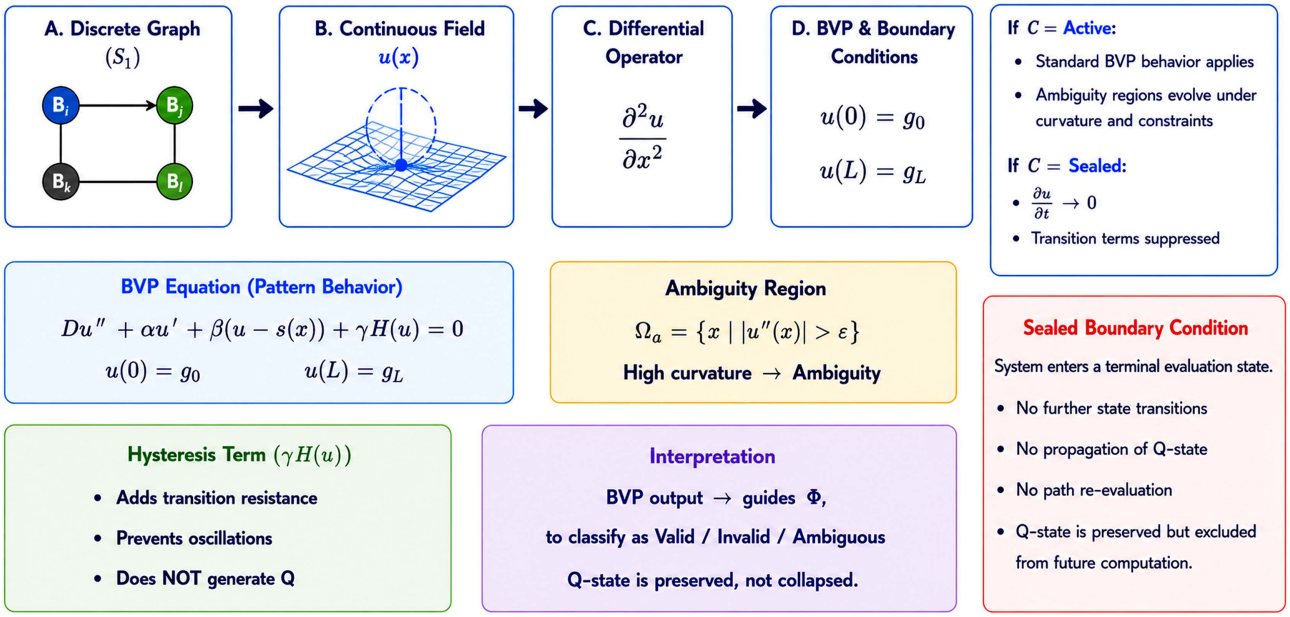


Figure 4. Boundary Value Problem (BVP) formulation applied to simulation experimentation. Discrete graph states are transformed into continuous evaluation fields where derivative-driven propagation behavior governs admissible state evolution. Ambiguity regions emerge under curvature and propagation constraints while hysteresis stabilizes transition behavior without enforcing premature deterministic Q-state collapse.

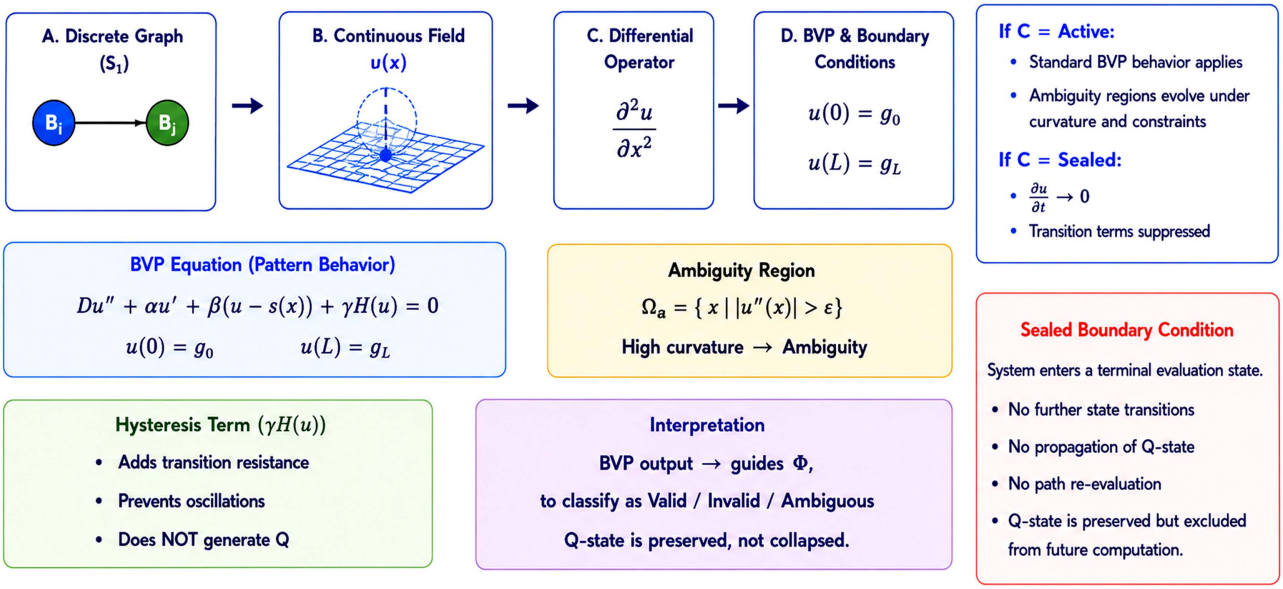


Figure 5. Boundary Value Problem (BVP) formulation applied to Byte Code ambiguity-aware evaluation. Fragmented byte-oriented structures generate admissible ambiguity regions under structural propagation constraints while preserving computational consistency and sparsity-aware interpretation behavior.

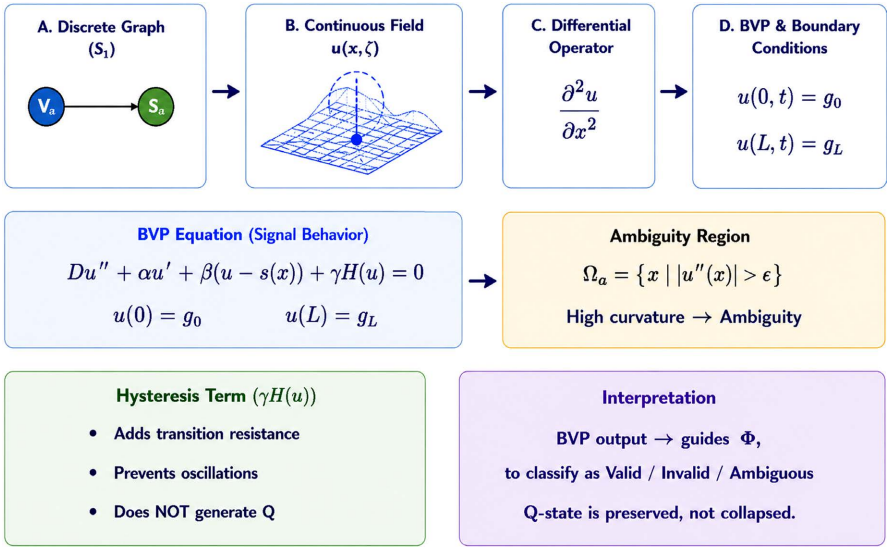


Figure 6. BVP formulation proposed for PCB/USB signal-behavior analysis. The figure represents a conceptual and simulation model; physical instrumentation remains planned.

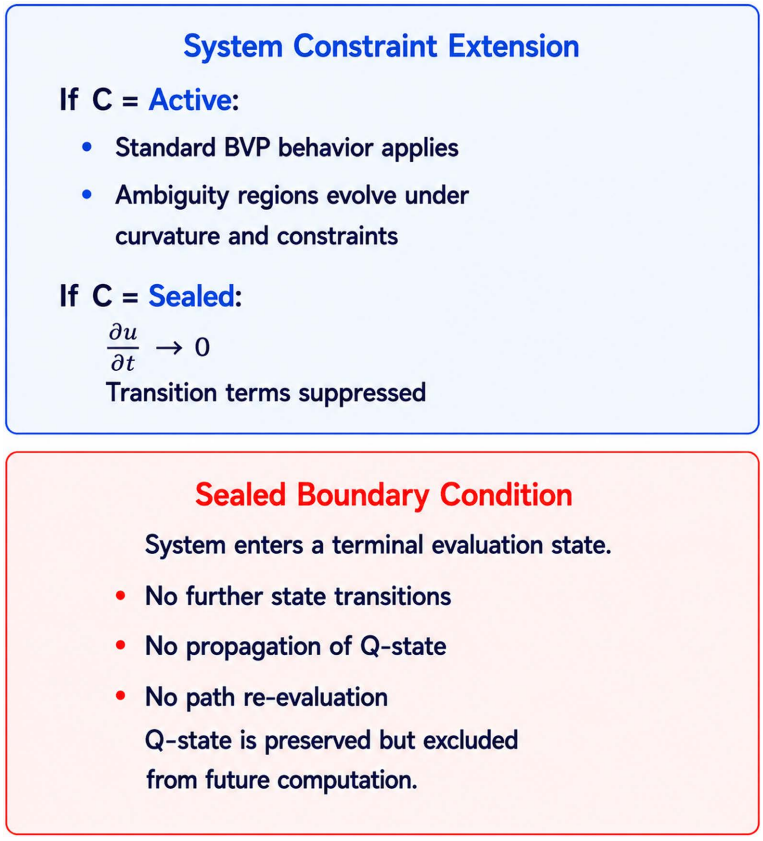


Figure 7. System-constraint extension and sealed-boundary behavior for the proposed PCB/USB ambiguity-aware propagation model.

The BVP formulation was therefore evaluated theoretically and through simulation-oriented computational models, including byte-oriented evaluation models and planned hardware-constrained propagation environments. These evaluations

indicated ambiguity preservation, admissible state retention, structural sparsity consistency, and propagation stability under boundary-conditioned evaluative computation [8] [13].

3.2.4. Control-Theoretic Interpretation

The proposed formulation admits a control-theoretic interpretation through its representation of constrained state evolution within a bounded state space. The boundary value formulation, together with first-order and second-order state evolution relationships, defines state assignment trajectories subject to validity constraints and boundary-conditioned transitions. Within this framework, hysteresis-mediated behaviors, state differentials, ambiguity propagation, and preservation mechanisms may be interpreted as state-space dynamics evolving under constrained computational conditions. Unlike conventional control formulations that primarily seek convergence toward a prescribed equilibrium state, the present framework permits the preservation and propagation of ambiguity-aware Q-States while maintaining boundary-conditioned validity across computational states [8] [10] [11]. This interpretation is an analytical analogy for constrained computational state evolution and should not be read as a claim of completed physical control-system instrumentation or hardware validation.

Related national-laboratory and NASA computing resources provide broader context for applied scientific computing, graph and irregular computing, large-scale simulation, operational reliability, and supercomputing infrastructure [21]-[24]. They are not presented as experimental validation of the framework developed here.

3.3. System and Node Definition

A system is defined as a structured graph-connected collection of interacting evaluative computational components operating within a Multi-State Computation Framework [8] [19] [20].

Formally, let:

$$G = (V, E). \tag{47}$$

where:

- V is the set of nodes,
- E represents relationships, transitions, propagation paths, or evaluative connectivity between nodes.

A node $v \in V$ represents a discrete evaluative computational unit participating in admissible ambiguity propagation, state transitions, and structurally governed path formation. New Edition clarification: Section 3.1.1 distinguishes the node relationships represented by E from an ordered candidate path Γ; node membership alone does not establish Path Validity.

3.4. Node State and State Space

Each node v is associated with a Node State:

$$s_v \in S = \{+1, 0, -1, Q\}. \tag{48}$$

where:

- +1: Active/Defined State,
- 0: Inactive/Undefined State,
- −1: Active/Undefined State,
- Q: Q-State (Ambiguous Evaluative State).

The Q-State is not treated as probabilistic uncertainty, stochastic randomness, or fuzzy interpolation [5] [8]. Instead, it represents structurally admissible evaluative ambiguity preserved under propagation and boundary-conditioned evaluative constraints. New Edition clarification: the general state space S remains distinct from the restricted relational sets Q_V and Q_{UI} defined in Section 3.1.1.

3.5. Computing State and Computational State

The framework distinguishes between Computing State and Computational State. Computing State describes observable execution-level operational conditions, whereas Computational State describes the ambiguity-aware evaluative interpretation of those conditions under admissibility constraints and Boundary Conditions.

Computing State represents the active execution-level operational configuration of the system, including node-state participation, runtime propagation behavior, and structurally observable system conditions:

$$C_{exec} = \{s_v\}_{v \in V}. \tag{49}$$

where:

- C_{exec} represents the execution-level operational state-space of the system,
- s_v represents the operational state associated with node v,
- V represents the graph-connected node space.

Computational State represents the ambiguity-aware evaluative interpretation of execution-level operational conditions under Boundary Conditions, admissibility constraints, and structurally governed propagation behavior:

$$C_{eval} = \Phi(C_{exec}, B). \tag{50}$$

where:

C_{eval} represents the evaluative computational interpretation space,

Φ represents an ambiguity-aware evaluative interpretation,

B represents Boundary Conditions governing admissibility, propagation continuity, and structurally constrained evaluative behavior.

Under this formulation, the transformation:

$$C_{eval} = \Phi(C_{exec}, B). \tag{51}$$

does not represent a purely deterministic mapping, but rather a structurally constrained evaluative interpretation process governed by ambiguity-aware propagation behavior, neighboring state influence, boundary-conditioned admissibility, and admissible state-space conditions previously established under Sections 3.1-3.3.

This distinction separates execution-level operational behavior from evaluative computational interpretation behavior [8]. A system may therefore exhibit stable execution-level operational conditions while remaining computationally ambiguous under structurally constrained evaluative requirements. New Edition clarification: the ordered relational configuration in Section 3.1.1 illustrates how computational interpretation may preserve participating node identities without collapsing them.

3.6. Q-State Derivation

Within the proposed framework, Q-State behavior is not arbitrarily assigned, probabilistically sampled, or generated through fuzzy interpolation mechanisms. Instead, Q-State emergence is derived through structurally admissible evaluative relationships governed by node-state participation, contextual propagation constraints, and Boundary Conditions [8].

Formally, let:

$$Q_i = \Phi\left(S_i, C_i, B_i\right). \tag{52}$$

where:

Q_i represents the admissible evaluative ambiguity state associated with node i,

S_i represents local node-state participation,

C_i represents contextual propagation constraints,

B_i represents Boundary Conditions governing admissibility, propagation continuity, and structurally constrained evaluative behavior.

The formulation:

$$Q_i = \Phi\left(S_i, C_i, B_i\right). \tag{53}$$

does not represent a solved deterministic equation, probabilistic mapping, stochastic assignment, or fuzzy-state interpolation process. Instead, it represents a derivational evaluative relationship through which admissible ambiguity emerges operationally under structurally constrained propagation behavior [8].

Under this formulation, Q-State emergence depends not solely on isolated local node conditions, but also on neighboring state influence, propagation continuity, admissibility constraints, hysteresis behavior, and boundary-conditioned evaluative interactions established throughout the ambiguity-aware computational framework.

A Q-State may therefore remain operationally preserved within the computational evaluation without requiring immediate deterministic collapse into resolved state assignments. New Edition clarification: the subsequent relational example $(-1, 0) \rightarrow (-1, +1)$ is a bounded derivational illustration, not a result of the original Codex-assisted preservation simulation.

This derivational formulation enables ambiguity-aware state-space behavior [8] [19] [20] to emerge dynamically through structurally governed evaluative propagation rather than through rigid predefined state assignment mechanisms.

3.7. Q-State Representation

A Q-State represents structured evaluative ambiguity arising from incomplete, conflicting, or structurally evolving system conditions [5] [6] [8]. It is treated as an operational evaluative element within the state space.

Representative admissible Q-State evaluative microstructures states include:

$$(1,0),(0,1),(1,-1). \tag{54}$$

These admissible evaluative microstructures states represent the coexistence of multiple admissible evaluative state interpretations while preserving structural consistency under ambiguity-aware propagation behavior [8]. New Edition clarification: Q_V and the later ordered relational set Q_{UI} serve different representational purposes and are not one undifferentiated Q-State domain.

3.8. Node State Validity

Node-State Validity determines whether a node contributes to admissible evaluative computational paths [8] [19] [20].

$$V(v) \in \{0,1\}. \tag{55}$$

where:

- $V(v)=1$ indicates an admissible node-state,
- $V(v)=0$ indicates a non-admissible node-state.

Node-State Validity depends on:

- consistency of state signals,
- compatibility with neighboring node-states,
- compliance with Boundary Conditions,
- admissibility under ambiguity propagation constraints.

Non-admissible node-states are excluded from admissible path formation and directly influence Sparsity Dynamics, propagation metrics, and Confidence Interval evaluations [8] [12]. New Edition clarification: individual Node State Validity does not by itself establish $V_{path}(\Gamma)$; admissible edges, Boundary Conditions, and state-transition relationships must also be satisfied.

3.9. Non-Collapse Principle

Under the Non-Collapse Principle, admissible ambiguity is not prematurely eliminated solely to satisfy deterministic evaluative requirements.

Instead, admissible ambiguity states remain operationally preserved until sufficient structural state conditions exist for structurally admissible state evaluative resolution [8]:

$$\exists B \text{ such that } Q \to s_v \in \{+1,0,-1\}. \tag{56}$$

where:

- Q represents an admissible evaluative ambiguity state,
- s_v represents a structurally admissible resolved node-state,
- B represents Boundary Conditions governing admissibility, propagation con-

tinuity, and evaluative transition behavior,

- $\exists B$ denotes the existence of sufficient boundary-conditioned evaluative constraints under which ambiguity-aware state transition may occur, New Edition clarification: the Section 3.1.1 progression $(-1, 0) \rightarrow (-1, +1)$ illustrates non-collapse by preserving the user/session node while the independently evaluated node changes state.
- $\rightarrow$ represents a boundary-conditioned evaluative transition relationship rather than a guaranteed deterministic collapse operation.

The existence condition on B does not guarantee immediate resolution [13], but instead defines the admissibility requirements under which ambiguity-aware evaluative transition behavior may occur.

Such admissibility conditions may incorporate:

- propagation stability,
- neighboring node-state consistency,
- admissible state continuity,
- hysteresis stabilization behavior,
- structurally valid evaluative conditions previously established under Sections 3.3-3.6.

Otherwise, admissible ambiguity remains operationally preserved within the computational evaluation without requiring forced deterministic collapse behavior [8].

The persistence of admissible ambiguity is therefore treated as a structurally meaningful evaluative computational condition rather than an invalid computational condition [6] [8]. This enables ambiguity-aware evaluation, admissible propagation behavior, and state preservation under boundary-constrained evaluative computational systems.

4. Ambiguity as an Operational Construct

In the deterministic baseline formulations used for comparison in this study, unresolved ambiguity is not represented as the persistent operational Q-State construct defined by this framework [3]-[6].

Within the proposed framework, ambiguity is instead treated as a structurally admissible operational evaluative condition subject to Boundary Conditions, propagation constraints, admissible state-space behavior, and evaluative interpretation [8].

Under this formulation, ambiguity may remain operationally preserved without requiring immediate deterministic collapse behavior [8]. This enables ambiguity-aware evaluation, constrained propagation behavior, and state-aware interpretation within graph-based and boundary-conditioned evaluative computational systems.

The framework distinguishes between Computing State and Computational State, allowing execution-level system behavior and evaluative interpretation behavior to evolve under separate but structurally related state conditions [8].

Ambiguity-aware computation therefore does not represent the absence of structure, but rather the structured preservation and evaluative interpretation of admissible ambiguity under constrained computational state conditions [6] [8].

Figure 8 illustrates ambiguity preservation behavior across Computing State and Q-State regions under constrained operational state conditions. The framework permits admissible ambiguity to remain structurally preserved without requiring immediate deterministic collapse behavior or unrestricted propagation behavior.

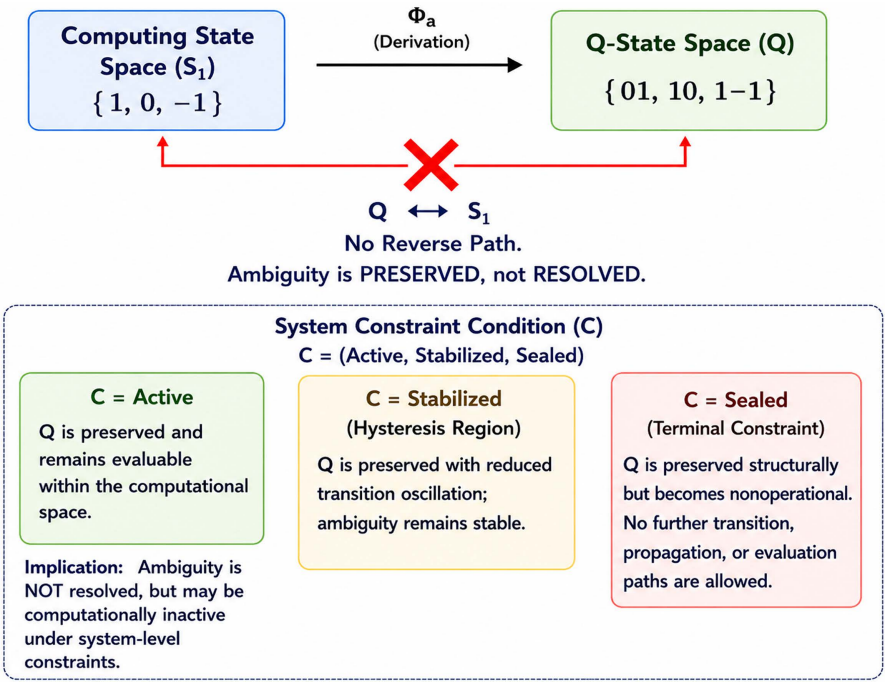


Figure 8. Simulation-based illustration of ambiguity-aware operational state behavior under constrained system state conditions. The framework preserves admissible ambiguity within Q-State regions without forced deterministic collapse behavior, while propagation behavior remains governed by Active, Stabilized, and Sealed system constraint conditions [8].

4.1. Ambiguity Evaluation

Ambiguity evaluation within the proposed framework refers to the structured evaluative interpretation of admissible ambiguous states under boundary-conditioned computational state constraints [8].

Relative to the deterministic state-resolution baseline considered in this study [3] [4] [8], the framework permits ambiguity-aware evaluation across node states, graph traversal behavior, neighboring state influence, and admissible propagation regions.

Evaluation behavior may therefore incorporate:

- node-state admissibility,
- propagation stability,
- path validity,

- sparsity-aware traversal behavior,
- ambiguity preservation conditions.

Ambiguity-aware evaluation does not imply unrestricted ambiguity propagation [8]. Instead, ambiguity propagation remains structurally governed by Boundary Conditions, admissible state-space behavior, and propagation constraints.

New Edition clarification: Section 3.1.1 further distinguishes independently admissible Node States from complete Path Validity. Ambiguity-aware evaluation may preserve the participating Node States while the connecting path remains subject to edge, Boundary Condition, and transition-validity requirements.

Figure 9 illustrates ambiguity-aware evaluation behavior across admissible state-space regions. Admissible ambiguity remains structurally preserved while evaluation behavior continues to operate under constrained propagation and boundary-conditioned admissibility requirements [8].

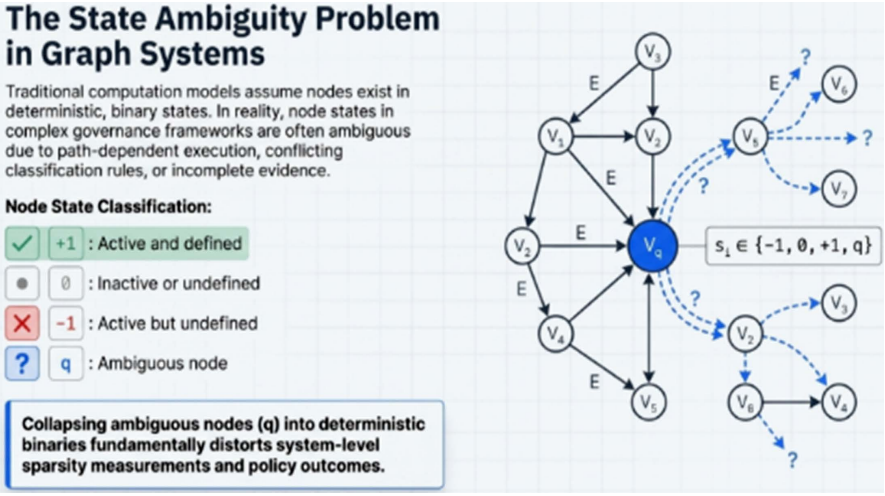


Figure 9. Simulation-based illustration of ambiguity-aware evaluation behavior across admissible state-space regions under constrained computing and computational state conditions.

4.2. Ambiguity Propagation

Ambiguity propagation within the framework describes the structurally governed evolution of admissible ambiguity across graph-connected nodes, neighboring evaluation regions, and constrained state-space systems [8] [19] [20].

Under this formulation, propagation behavior is influenced by:

- local node-state conditions,
- neighboring propagation constraints,
- hysteresis stabilization behavior,
- sparsity dynamics,
- admissible path continuity.

Propagation behavior therefore evolves under constrained computing and computational state conditions rather than unrestricted probabilistic diffusion or

unbounded ambiguity propagation [6] [8].

This enables ambiguity-aware traversal behavior while preserving structural validity and propagation stability across admissible computational regions [8] [12] [19] [20].

New Edition research context: Berkeley Lab NERSC and AMCR resources provide related context for large-scale scientific computing, mathematical modeling, algorithm design, high-performance software, and computational research [25] [26]. LLNL CASC identifies uncertainty quantification and graph and irregular computing as active scientific-computing areas [21]. These sources are contextual and are not presented as validation of the Q-State propagation framework.

New Edition clarification: Section 3.1.1 establishes that an independently admissible node does not, by itself, establish a valid propagation path; admissible connecting relations and transition constraints remain required.

Figure 10 illustrates ambiguity propagation behavior across graph-connected evaluation regions under constrained computational conditions. Admissible ambiguity propagation remains structurally governed by neighboring node-state behavior, propagation constraints, and sparsity-aware traversal continuity [8].

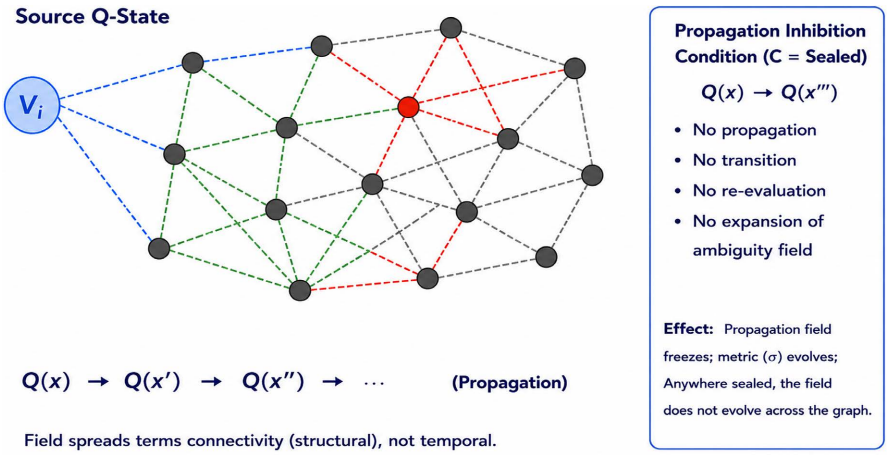


Figure 10. Simulation-based illustration of ambiguity-aware propagation behavior across admissible graph-connected computational regions. Propagation behavior remains structurally governed by neighboring state influence, propagation constraints, and admissible traversal conditions.

4.3. Ambiguity Path Evaluation

A path:

$$P = (v_1, v_2, \cdots, v_k). \tag{57}$$

satisfies the following node-level admissibility condition:

$$\forall v_i \in P, V(v_i) = 1. \tag{58}$$

where: $\forall$ denotes the universal quantifier, meaning "for all" or "for every"; v_i denotes the (i)-th node in the computational path P; P denotes the computational path. $V(v_i)$ denotes the node-validity function evaluated at v_i; and $V(v_i) = 1$

indicates that node v_i is valid. Equation (58) expresses a necessary node-level admissibility condition and is not, by itself, the complete Path Validity definition. Complete Path Validity remains governed by Section 3.1.1, including admissible participating Node States, admissible connecting edges, applicable Boundary Conditions, ordered state-transition admissibility, and at least one connecting transition.

Q-States may exist within valid paths provided that admissibility constraints, neighboring propagation conditions, and Boundary Conditions remain satisfied [8] [19] [20].

Ambiguity Path Evaluation within the proposed framework determines whether admissible ambiguity may continue to propagate across graph-connected computing or computational regions [8].

Unlike deterministic traversal models that assume fully resolved node conditions, the framework permits Q-State participation within valid propagation paths under structurally constrained evaluation behavior [8] [19] [20].

Path evaluation therefore depends not only on local node validity, but also on:

- propagation continuity,
- sparsity-aware traversal behavior,
- neighboring node consistency,
- hysteresis stabilization,
- ambiguity preservation constraints.

This enables ambiguity-aware graph traversal behavior while preventing invalid propagation expansion across structurally inconsistent computing or computational regions [8] [12] [19] [20].

Under this formulation, admissible paths preserve structural evaluation consistency while allowing ambiguity-aware propagation behavior to evolve under constrained computational conditions [8].

New Edition clarification: The original PDF uses P for the computational path, while Section 3.1.1 uses Γ as the later canonical ordered-path notation. They identify the same path role in their respective formulations. The condition $V(v_i)=1$ for every participating node is necessary but not sufficient for complete $V_{path}(\Gamma)$, which additionally requires admissible edges, Boundary Conditions, and state-transition relationships.

4.4. Resolution Conditions

Ambiguity resolution may occur under structurally admissible Boundary Conditions [8]:

$$Q \to s_v \in \{+1, 0, -1\}. \tag{59}$$

where:

- Q represents an admissible evaluative ambiguity state,
- s_v represents a structurally admissible resolved node-state,
- $\to$ represents a boundary-conditioned evaluative transition relationship ra-

ther than a guaranteed deterministic collapse operation.

Ambiguity resolution within the proposed framework occurs only when admissible structural conditions satisfy propagation, evaluation, and boundary-constrained stabilization requirements [8].

Relative to the deterministic state-resolution baseline considered in this study, the framework preserves operational Q-State behavior until sufficient structural consistency exists for an admissible evaluative state transition. Resolution therefore represents a constrained evaluative transition rather than an unconditional elimination of ambiguity [3] [4] [8].

Under unresolved or structurally inconsistent conditions, admissible ambiguity may remain operationally preserved within the computational state evaluation without requiring premature deterministic collapse behavior [8].

This behavior enables stable admissible ambiguity retention, admissible state preservation, and structurally governed evaluation continuity across ambiguity-aware evaluative computational systems [6] [8].

New Edition clarification: The Section 3.1.1 progression $(-1, 0) \rightarrow (-1, +1)$ preserves the user/session node while the independently evaluated node changes state. It is a relational non-collapse illustration and does not assert a general or unconditional resolution of Q to +1.

5. Simulation and Experimental Observations

Simulation and experimental observations were conducted to evaluate ambiguity-aware propagation behavior, admissible state retention, structural sparsity consistency, and boundary-constrained evaluation conditions previously established under Sections 3 and 4 [8] [13].

The experimental environments included lattice-based propagation models, graph-connected evaluation structures, ambiguity-aware state-space simulations, byte-oriented evaluation scenarios, PCB/USB propagation studies, Codex-assisted simulation analysis, IBM Power-based AIX experimentation environments, Matplotlib-based propagation visualization systems, and boundary-conditioned stabilization observations [13] [17] [18] [27].

These observations were not intended to establish exhaustive benchmarking or optimization claims, but rather to evaluate the operational consistency of ambiguity-aware computing and computational behavior under constrained propagation and admissibility conditions [8] [13].

5.1. Simulation Environment and Experimental Context

Simulation experimentation was performed across multiple ambiguity-aware evaluative computational environments to evaluate propagation continuity, admissible state preservation, sparsity-aware traversal behavior, and hysteresis stabilization effects under constrained computational conditions [8] [13].

The experimental environments incorporated graph-connected lattice structures, ambiguity-aware state-space evaluation models, byte-oriented fragmenta-

tion analysis, PCB/USB propagation simulations, and boundary-conditioned propagation studies previously introduced under Sections 3.1-3.7 [8] [19] [20].

Codex-assisted simulation analysis environments were utilized to evaluate ambiguity-aware propagation continuity, neighboring node-state behavior, admissible path evolution, and structurally constrained state-space interpretation across multiple simulation iterations [13].

The Codex-assisted evaluation workflow further supported operational analysis of ambiguity-aware propagation behavior, neighboring node-state influence, admissible path continuity, iterative propagation stabilization, and structurally constrained evaluative computational interpretation across multiple experimentation environments [13].

The Codex-assisted simulation workflow was utilized to interpret ambiguity-aware propagation continuity, execution-level state-transition behavior, sparsity-aware traversal conditions, admissibility verification, and propagation stabilization consistency under boundary-conditioned computational constraints [8] [13].

Experimental observations conducted within IBM Power-based AIX simulation environments demonstrated ambiguity-aware propagation continuity, admissible state retention, and stabilization consistency under boundary-constrained evaluation conditions.

Matplotlib-based visualization environments were further utilized to evaluate propagation stabilization behavior, ambiguity-aware state retention, hysteresis continuity, and structurally governed evaluation evolution across graph-connected computational regions [17] [18] [27].

Operational observations were evaluated using ambiguity-aware propagation continuity, neighboring node influence, admissible path consistency, sparsity-aware traversal behavior, stabilization continuity, and structurally constrained evaluative computational behavior as primary observational criteria [8] [13].

As illustrated in **Figure 11**, graph-connected lattice evaluation native lab environments were utilized to evaluate ambiguity-aware path continuity, structural activation density, neighboring node influence, admissible propagation behavior, and sparsity-aware traversal continuity across sampled execution paths [8] [19] [20].

The IBM Power-based experimentation environment shown in **Figure 12** was utilized to evaluate ambiguity-aware simulation continuity, iterative propagation behavior, seeded propagation stabilization, and operational consistency under constrained evaluation conditions [8].

5.1.1. Simulation Environment Summary

The simulation experiments utilized multiple natively run environments, including complementary execution environments [13].

The environments included a natively run Codex-assisted simulation environment and orchestration workflows used for specification development, test-case generation, simulation planning, artifact generation, execution review, and simu-

lation analysis [13].

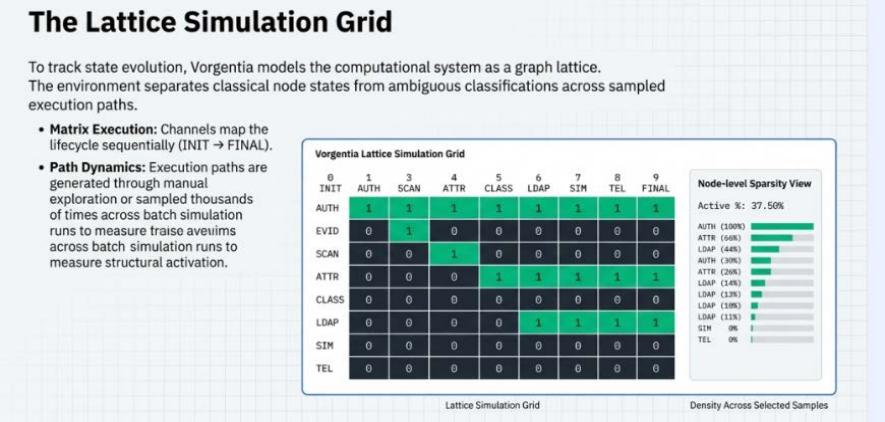


Figure 11. The Lattice Simulation Grid illustrates ambiguity-aware graph-connected evaluation behavior across sampled execution paths. The simulation environment separates classical node-state activation from ambiguity-aware structural evaluation conditions while measuring sparsity-aware path continuity and node-level activation behavior under constrained computational conditions.

```c
/*************************************************************
 * Vorgentia Research Simulation Framework
 * Primary Contributor: Abhishek Kumar
 * Affiliation: Vorgentia Research – Vorg / Preferido
 *
 * PURPOSE:
 * Multi-state ambiguity simulation (Q1, Q2, Q3)
 *
 * SECURITY:
 * - No network I/O
 * - User-space execution only
 * - Secure file permissions
 *
 * DISCLAIMER:
 * Research use only. No production usage.
 *************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <time.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/stat.h>

#define N 32
#define THREADS 4
#define ITER 5

#define Q1 2
#define Q2 3
#define Q3 4

typedef struct {
    int cur[N];
    int next[N];
} S;
```

Figure 12. IBM Power-based AIX simulation experimentation environment utilized for ambiguity-aware propagation evaluation, seeded propagation iteration analysis, and structurally constrained simulation continuity observations.

The environments also included native lattice-based execution environments used for direct evaluation of ambiguity-aware state-space behavior, graph-connected propagation characteristics, node-state transitions, and ambiguity-preserving computational behavior [8] [19] [20].

Native simulation execution established by reserving IBM Power 11 to simulate using IBM Power-based AIX environment and packages utilizing GCC-based C compilation and execution workflows. These environments were used to evaluate ambiguity-aware propagation continuity, state preservation characteristics, admissible path evolution, sparsity-aware traversal behavior, and stabilization consistency under boundary-conditioned computational constraints [8].

Research activities also including the active ongoing experimentations, simulations and artifacts were undertaken and completed throughout 2025-2026 by Abhishek Kumar, Vorgentia Research (VORG, Preferido). The combined use of Codex-assisted and native execution environments enabled simulation evaluation across multiple specifications, computational scenarios, ambiguity-preserving conditions, graph configurations, dependency structures, and state-transition conditions [8] [13].

5.1.2. Simulation Specifications and Execution Context

Simulation activities were performed using structured framework specifications, mathematical formulations, execution workflows, verification procedures, validation procedures, expected outcome definitions, and acceptance criteria developed for ambiguity-aware computational evaluation [8] [13].

The simulation specifications defined computational nodes, node objects, object relationships, dependency structures, graph-connected evaluation paths, state-transition behavior, ambiguity-preserving Q-State formation, sparsity-aware traversal conditions, preservation requirements, and outcome interpretation rules. Multiple simulation configurations and representative test cases were utilized to evaluate ambiguity propagation behavior, path validity characteristics, state preservation conditions, stabilization behavior, and computational consistency across differing ambiguity-aware evaluation scenarios [8] [13] [19] [20].

Execution activities incorporated graph construction, dependency evaluation, boundary evaluation, state assignment, ambiguity derivation, Q-State formation, preservation evaluation, interpretation, metric generation, verification preparation, validation preparation, and outcome assessment activities. These execution workflows provided a consistent basis for evaluating ambiguity-aware computational behavior across multiple simulation environments and representative computational conditions [8] [13].

5.2. Transformation Pipeline and State Evaluation Flow

Within the proposed framework, ambiguity-aware evaluation does not occur through a single deterministic state resolution operation. Instead, evaluation behavior evolves through a structured transformation pipeline governed by Boundary Conditions, admissibility constraints, neighboring state influence, and ambi-

guity-aware propagation continuity [8].

The deterministic input-to-output baseline formulations considered in this study use predefined state-transition or assignment behavior and do not implement the persistent Q-State semantics defined by the proposed framework [2]-[4].

In contrast, the proposed framework utilizes a transformation pipeline to preserve, interpret, propagate, and structurally evaluate admissible ambiguity throughout multiple evaluative stages rather than forcing immediate deterministic collapse behavior [8].

The transformation pipeline therefore separates execution-level operational behavior from evaluative computational interpretation behavior while preserving admissible ambiguity throughout constrained propagation stages [8].

Under this formulation, ambiguity-aware evaluation may progress through the following operational stages:

$$S_1 \to \Phi_a \to Q \to \Phi_i \to S_2 . \tag{60}$$

where:

- S_1 represents an initial operational node-state or execution-level computational condition,
- Φ_a represents ambiguity-aware evaluative interpretation under admissibility and propagation constraints,
- Q represents an admissible evaluative ambiguity state,
- Φ_i represents structurally constrained evaluative interpretation behavior operating under Boundary Conditions and neighboring state influence,
- S_2 represents a structurally admissible evaluative state transition outcome.

The transformation pipeline does not represent a guaranteed deterministic resolution sequence or unconditional ambiguity elimination [8]. Instead, each evaluative transition stage remains governed by propagation continuity, admissibility behavior, hysteresis stabilization, neighboring node-state influence, and boundary-conditioned evaluation constraints.

Under unresolved or structurally inconsistent conditions, the pipeline permits admissible ambiguity to remain operationally preserved within the computational state evaluation without requiring premature deterministic collapse behavior [8].

The significance of the transformation pipeline extends directly into the ambiguity-aware pseudocode and algorithmic evaluation methodology utilized throughout the framework [8] [13]. Relative to the deterministic state-assignment baseline evaluated in this study, the proposed ambiguity-aware evaluation algorithms incorporate admissibility verification, neighboring node influence, propagation continuity, sparsity-aware traversal behavior, and boundary-conditioned evaluative interpretation across multiple operational stages.

Algorithmically, the transformation pipeline governs the workflow through which node-states participate in ambiguity-aware propagation behavior [8] [13]:

1) Initial node-state activation and execution-level propagation behavior,

2) ambiguity-aware evaluative interpretation under admissibility constraints,

3) Q-State derivation and admissible ambiguity preservation,

4) neighboring node-state propagation evaluation,

5) sparsity-aware path continuity verification,

6) boundary-conditioned evaluative stabilization,

7) structurally admissible evaluative transition behavior or retained ambiguity preservation.

This workflow enables ambiguity-aware pseudocode and propagation algorithms to evolve dynamically through structurally governed evaluation behavior rather than through rigid deterministic assignment or unconditional state-collapse operations [8] [13].

Figure 13 illustrates the ambiguity-aware transformation pipeline across operational, evaluative, and Q-State propagation stages. Under this formulation, admissible ambiguity remains structurally preserved throughout evaluative interpretation behavior until sufficient boundary-conditioned admissibility requirements exist for structurally constrained evaluative state transition [8].

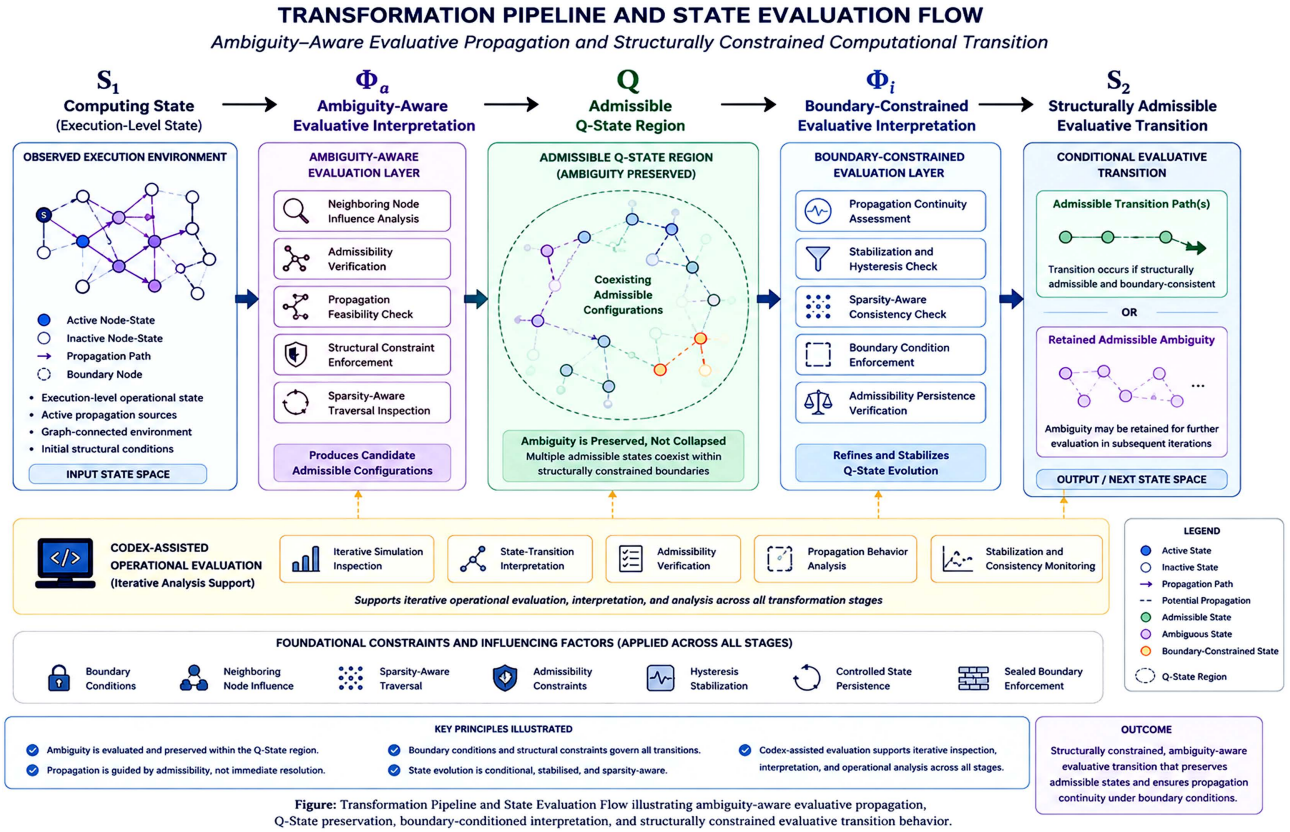


Figure 13. Transformation Pipeline and State Evaluation Flow illustrating ambiguity-aware evaluative transition behavior under boundary-conditioned computational state constraints. This separates execution-level operational behavior from evaluative computational interpretation while preserving admissible ambiguity continuity throughout propagation and state-transition stages.

Simulation Execution Workflow

While the transformation pipeline defines the ambiguity-aware evaluative progression of computational states, simulation execution was performed through a structured workflow designed to support graph construction, ambiguity deriva-

tion, preservation analysis, outcome generation, and evaluation consistency across multiple simulation environments [8] [13].

Simulation execution begins with initialization of computational objects, node states, dependency structures, evaluation parameters, and graph-connected lattice representations. Interactions between computational states are established and utilized to construct admissible evaluation paths and neighboring node-state relationships [13] [19] [20].

Following initialization, dependency evaluation and boundary-conditioned assessment activities are performed to determine admissible propagation behavior, state-transition eligibility, ambiguity-preserving conditions, and neighboring state influence. State assignments and state differentials are subsequently derived and utilized for ambiguity-aware evaluation activities [8] [13].

Ambiguity-preserving Q-States are formed through admissibility evaluation and ambiguity derivation procedures [8] [13]. Preservation analysis, interpretation activities, outcome generation, metric generation, verification preparation, validation preparation, and acceptance assessment activities are then performed to support consistent ambiguity-aware computational evaluation across multiple simulation configurations and representative test conditions.

The execution workflow therefore complements the transformation pipeline by providing an operational mechanism through which ambiguity-aware computational evaluation, propagation continuity, path validity assessment, sparsity-aware traversal behavior, and preservation analysis may be consistently evaluated throughout the simulation process.

5.3. Q-State Propagation and Sparsity Observations

Experimental observations demonstrated that ambiguity-aware propagation behavior remained structurally governed under admissible graph-connected evaluation conditions [8] [13].

Under ambiguity-aware propagation models, Q-State persistence influenced:

- admissible path continuity,
- neighboring propagation stability,
- sparsity-aware traversal behavior,
- propagation inhibition behavior,
- boundary-conditioned evaluation consistency.

Relative to the deterministic state-assignment baseline evaluated in this study, ambiguity-aware propagation preserved structurally admissible ambiguity regions without requiring unconditional deterministic collapse [2]-[4].

Experimental observations further indicated structural sparsity consistency under the ambiguity-aware propagation conditions evaluated here, particularly within graph-connected environments containing incomplete, conflicting, or evolving node-state behavior [8] [12] [19] [20].

Repeated seeded propagation observations conducted within IBM Power-based simulation environments demonstrated progressive reduction of ambiguity-

aware propagation regions under constrained evaluation conditions while preserving admissible structural continuity [13].

Representative simulation iterations exhibited decreasing ambiguity-region density and stabilized propagation behavior across the evaluated graph-connected structures, consistent with the boundary-conditioned admissibility constraints established under Sections 3 and 4 [8] [13].

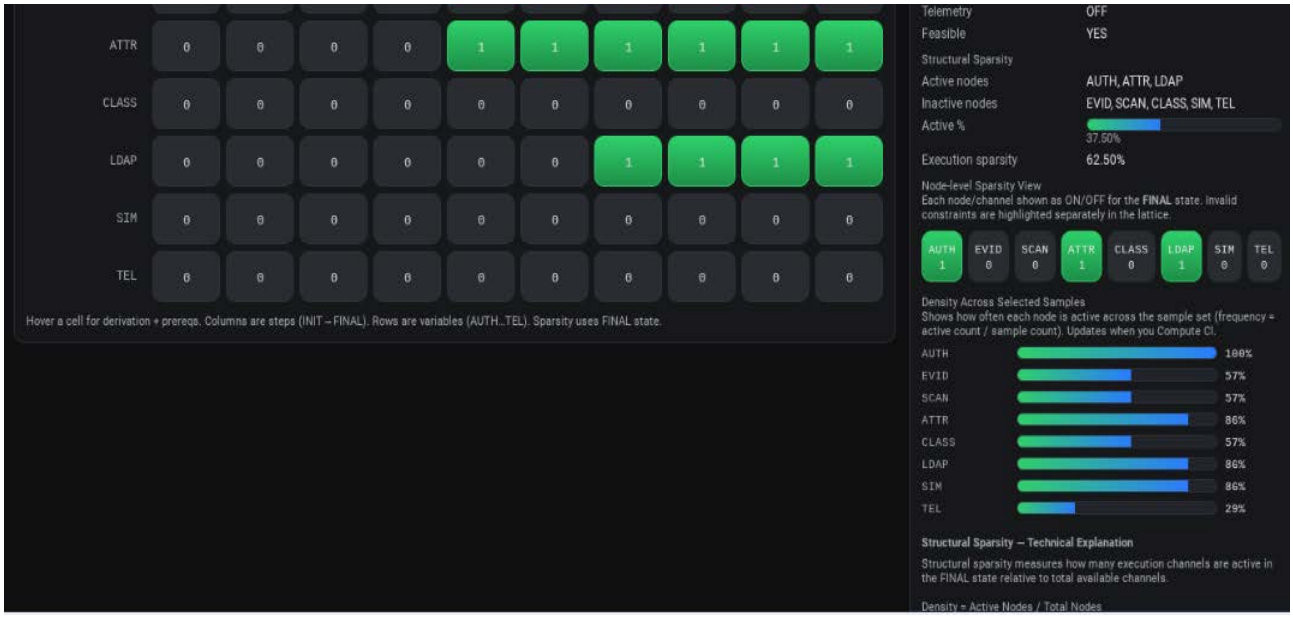


Figure 14. Simulation lattice evaluation illustrating ambiguity-aware propagation behavior, structural sparsity continuity, and admissible graph-connected evaluation conditions across sampled computational paths. Experimental observations demonstrate sparsity-aware propagation behavior and structurally governed evaluation continuity consistent with Sections 3 and 4.

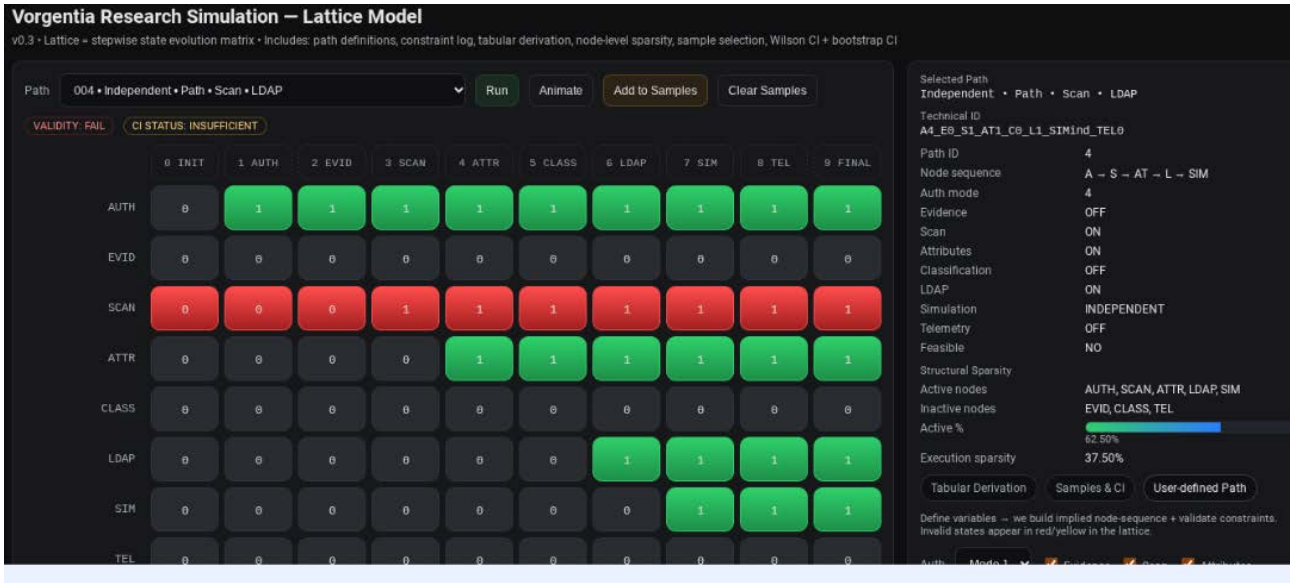


Figure 15. Ambiguity-aware simulation lattice model illustrating valid and invalid propagation paths, node-level sparsity evaluation, and structurally governed admissibility behavior across graph-connected evaluation regions [F11: Ambiguity-Aware Simulation Lattice Model].

Figure 14 illustrates ambiguity-aware sparsity continuity and propagation evaluation behavior across sampled graph-connected execution paths. The simulation environment evaluates structural sparsity, admissible node participation, and propagation continuity while preserving ambiguity-aware evaluation semantics under constrained computational conditions [8] [12] [19] [20].

As illustrated in Figure 15, ambiguity-aware lattice simulations were utilized to evaluate admissible path continuity, invalid-state exclusion behavior, sparsity-aware traversal conditions, and graph-connected propagation stability across sampled execution environments [8] [19] [20].

Table 2 below is a representation of Observations with Experimental Outcomes.

Table 2. Representative experimental observations.

Observation	Experimental Outcome
Ambiguity Retention	Operationally Preserved
Propagation Stability	Structurally Stabilized
Sparsity Continuity	Maintained
Hysteresis Stabilization	Observed
Admissible Path Continuity	Preserved

These observations remained operationally consistent with the admissibility and Non-Collapse conditions previously established under Sections 3.6 and 3.9 [8].

Simulation Metrics and Evaluation Measures

In addition to qualitative propagation observations, simulation activities incorporated a collection of evaluation measures designed to assess ambiguity-aware computational behavior across graph-connected execution environments [8] [13]. These measures were utilized to characterize admissible node participation, path continuity, sparsity-aware traversal behavior, ambiguity retention, and evaluation consistency throughout the simulation process.

Representative evaluation measures included valid node participation, admissible path continuity, sparsity-related observations, ambiguity-preserving state retention, and confidence-oriented evaluation measures derived from observed propagation behavior. These measures were utilized to compare simulation outcomes across multiple configurations, specifications, dependency structures, and representative test conditions [8] [12].

The resulting evaluation measures provided an additional basis for evaluating ambiguity-aware computational behavior beyond qualitative observations alone and supported verification, validation, and outcome assessment across multiple simulation configurations and representative experimental conditions [8] [13].

The simulations, experimental evaluations, and supporting research artifacts described in this work were undertaken and developed by Abhishek Kumar, Vorgentia Research (VORG | Preferido), as part of the ongoing research program conducted during 2025-Present [8] [13].

Table 3 below summarizes the representation of Simulation Evaluation Measures.

Table 3. Representative simulation evaluation measures.

Evaluation Measure	Purpose
Ambiguity Retention and Q-State Persistence	Evaluate preservation and continuity of ambiguity-preserving states during propagation and interpretation.
Valid Node Participation	Evaluate admissible node participation across graph-connected evaluation activities.
Admissible Path Continuity	Evaluate propagation continuity and path-level consistency across graph-connected computational regions.
Sparsity Continuity	Evaluate structural participation density and sparsity-aware traversal behavior.
Propagation Stability and Hysteresis Stabilization	Evaluate stabilization behavior, retention characteristics, and propagation consistency under neighboring-node influence and boundary-conditioned constraints.
Confidence-Oriented Evaluation	Evaluate consistency of observed outcomes across representative simulation configurations, specifications, and test conditions.

5.4. Stabilization, Boundary Condition and Ambiguity Retention

Experimental stabilization observations demonstrated that admissible ambiguity may remain operationally preserved under constrained evaluation conditions without requiring premature deterministic collapse behavior [8] [13].

Boundary-conditioned propagation environments exhibited retained ambiguity continuity, hysteresis stabilization behavior, and structurally governed propagation inhibition across graph-connected evaluation structures [8] [10] [11].

To evaluate ambiguity-aware propagation stabilization behavior, a Python/Matplotlib iterative research simulation was used to evaluate count-based propagation activity and the evolution of a simulation-defined sealing coefficient under boundary-constrained computational conditions [17] [18].

The verified source artifact evaluates the relationship between a qualifying edge-count activity series, denoted here as $C_Q(t)$, and a derived sealing coefficient $m_{seal}(t)$. The count series is reduced by the simulation's integer hysteresis schedule, while the coefficient is computed from the current count, the preceding count, and the initial count. Neither quantity is a Binary State, Absolute State, or Q-State value.

Python/Matplotlib Research Simulation (executed on Linux; no Linux mseal() system call is invoked):

Additional review of the sealing condition. As an additional step in reviewing and analyzing this research simulation, the applicability of sealing was examined in relation to continued propagation, ambiguity evaluation, and applied hysteresis.

Figure 16 should be interpreted together with the Active, Stabilized, and Sealed conditions described in Figure 7. The purpose of this additional analysis is to clarify when suppression of further computational activity is appropriate under the applicable boundary conditions while the established Q-State remains preserved.

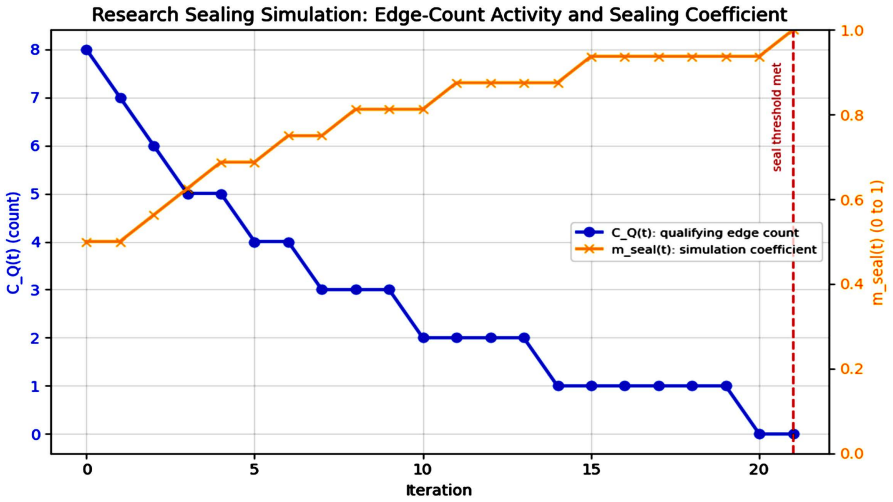


Figure 16. Python/Matplotlib visualization of count-based propagation activity and a simulation-defined sealing coefficient. The blue series $C_Q(t)$ is the integer count of qualifying evaluated node pairs after the source code's hysteresis reduction; it is not the normalized node-participation measure P_Q defined by Equation (60). The orange series $m_{seal}(t)$ is calculated by the research simulation on a 0-to−1 scale. The graph illustrates suppression of further propagation or evaluation activity while an established Q-State can remain preserved under the Non-Collapse Principle in Section 3.9.

Technical interpretation. Sealing suppresses continued propagation and ambiguity evaluation, including further path re-evaluation, while preserving the established Q-State; suppression does not itself imply that ambiguity has been resolved or that Q has been discarded. Applied hysteresis governs stabilization and transition behavior, while the applicable sealing condition determines when suppression takes effect. In this simulation, m_{seal} is a numerical research coefficient bounded from 0 to 1. Its values quantify the coefficient and do not denote Binary Node States, Absolute States, or Q-State representations. The numerical range alone does not establish a universal sealing threshold. The run shown in Figure 15 uses a configured threshold of 0.95.

Terminology and operating-system boundary. The research coefficient mseal is distinct from the Linux mseal() system call introduced in Linux 6.10. The Linux interface seals virtual-memory mappings against specified later memory-management changes; it does not implement Q-State computation, and sealed mappings do not make memory contents universally immutable [28] [29]. The commonality used here is limited to the security concept of preserving an established mapping or state while restricting selected future transitions.

Source-artifact verification. The source file *mseal_active_simulation*(1).*py* and its plotted artifact *mseal_plot*(9).*png* were reviewed together. The source variable named P_Q is semantically a qualifying edge count and is therefore denoted $C_Q(t)$ in this paper. The internally generated source and plot, rather than external literature, are the evidence for the displayed numeric sequence. The illustrated source uses an unseeded random initial node assignment, so reproducing the same trace requires recording the initial states or a random seed.

The verified relationship between decreasing count-based propagation activity and increasing sealing-coefficient values is consistent with the boundary-conditioned admissibility behavior and Non-Collapse semantics established under Sections 3.9 and 4.4 [8]. It does not establish that Q-State membership becomes zero when the activity count reaches zero.

Lattice-Based Experimental Simulations:

Figure 17 illustrates operational ambiguity preservation behavior under Active, Stabilized, and Sealed evaluation conditions. Under sealed boundary constraints, ambiguity propagation behavior becomes structurally constrained while preserving admissible Q-State continuity without uncontrolled recursive propagation or premature deterministic collapse behavior [8] [19] [20].

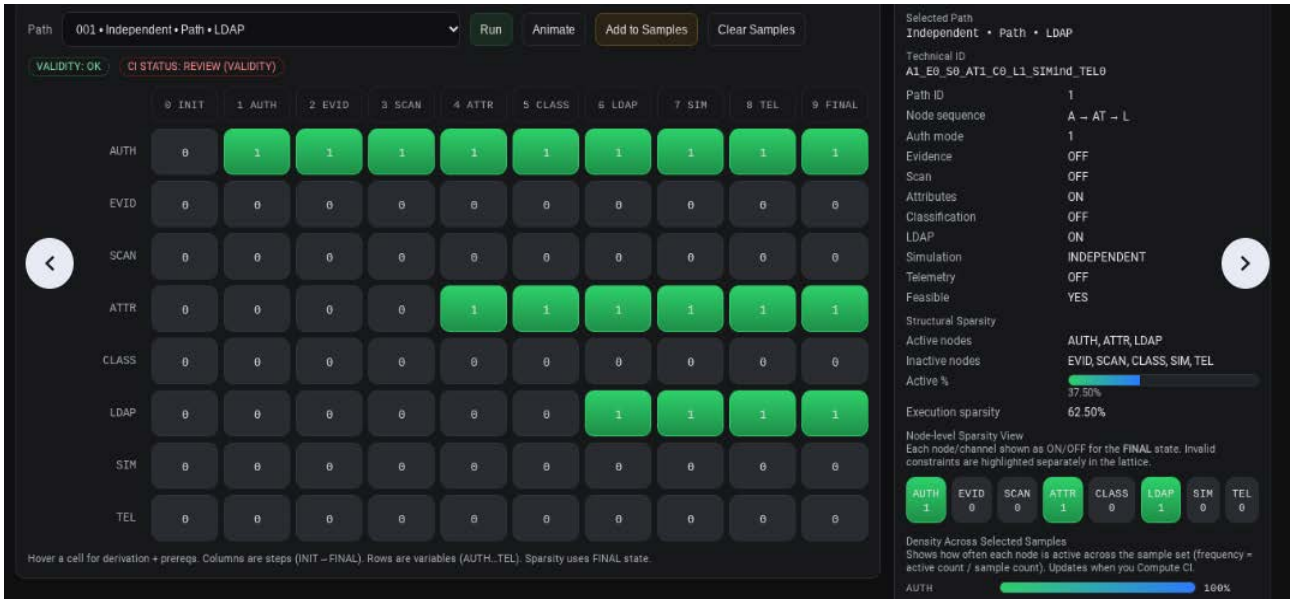


Figure 17. Simulation experimental illustration of ambiguity-aware operational state behavior under constrained system conditions. The framework preserves admissible ambiguity within Q-State regions without forced deterministic collapse behavior while propagation behavior remains governed by Active, Stabilized, and Sealed system constraint conditions.

Research sealing-related stabilization observations further demonstrated ambiguity-retention behavior under sealed evaluation conditions, particularly within propagation regions containing incomplete, conflicting, or evolving node-state conditions [8] [12].

Relative to the deterministic resolution baseline evaluated in this study, the ambiguity-aware evaluation environments preserved admissible ambiguity regions

while maintaining structurally valid propagation continuity and evaluation consistency [3] [4] [8].

These observations remained operationally consistent with the Non-Collapse Principle established under Section 3.9 and the ambiguity-aware evaluation conditions introduced under Section 4 [8].

Propagation and Stabilization Measures

To support interpretation of ambiguity-aware stabilization behavior, representative propagation and stabilization measures were utilized throughout the research simulation studies [8] [13].

The propagation measure P_Q represents the relative extent of ambiguity-preserving Q-State participation within a graph-connected evaluation environment and is expressed as:

$$P_Q = \frac{N_Q}{N}. \tag{61}$$

where N_Q denotes the number of nodes participating in ambiguity-preserving Q-State conditions and N denotes the total number of evaluated nodes within the simulation environment.

Larger values of P_Q indicate greater ambiguity-aware propagation activity, whereas decreasing values indicate progressive reduction of ambiguity propagation regions under constrained evaluation conditions [8] [12].

Figure 16 normalization boundary. Equation (61) defines the node-based proportion $P_Q = \frac{N_Q}{N}$, which is bounded from 0 to 1. **Figure 15** does not plot that proportion. With six nodes, the source evaluates 15 index-ordered pairs ($i < j$) and plots the qualifying-pair count $C_Q(t)$, which begins at 8 in the displayed run. An edge-based normalized quantity could be defined separately as $C_Q/15$; it would not be identical to the node-based P_Q in Equation (61). A node-based value requires an explicit count of participating nodes and must not be inferred merely by relabeling the plotted edge count.

The sealing coefficient m_{seal} represents a stabilization measure associated with boundary-conditioned propagation inhibition and ambiguity retention behavior and is expressed as:

$$0 \le m_{seal} \le 1. \tag{62}$$

Equation (62) states the bounded range $0 \le m_{seal} \le 1$; it is not the calculation formula. In **Figure 16** source, $m_{seal}(t)$ is computed from the current and preceding qualifying-edge counts relative to the initial count, then clamped to this range. Increasing values indicate stronger stabilization under that simulation rule, not an operating-system measurement or a universal security threshold.

Together, P_Q and m_{seal} provide complementary measures for evaluating ambiguity-aware propagation behavior, stabilization continuity, boundary-conditioned evaluation behavior, and admissible ambiguity retention throughout iterative simulation activities [8] [13].

5.5. Experimental Interpretation Summary

The simulation and experimental observations presented throughout Section 5 collectively demonstrated structurally consistent ambiguity-aware evaluative behavior within the evaluated configurations across propagation continuity, admissible state retention, sparsity continuity, and stabilization behavior under constrained computational conditions [8] [13].

Experimental observations, simulations, representative use cases, test cases, and related research activities conducted to date, together with ongoing technical investigations and supporting research activities undertaken by Abhishek Kumar, Vorgentia Research (VORG | Preferido), across IBM Power-based computational experimentation environments, graph-connected lattice structures, ambiguity-aware state-space simulations, Codex-assisted analysis environments, Matplotlib-based propagation visualization systems, and boundary-conditioned stabilization studies, remained operationally consistent with the ambiguity-aware computational framework established under Sections 3 and 4 [8] [13].

The experimental observations further remained operationally consistent with the ambiguity-aware transformation pipeline and evaluative propagation workflow introduced throughout Sections 4 and 5 [8] [13].

Collectively, these observations support the interpretation that ambiguity-aware evaluative computational systems may preserve admissible ambiguity, propagation continuity, and structurally governed evaluative behavior without requiring premature deterministic collapse under incomplete, conflicting, or evolving computational conditions [8].

The experimental interpretation further incorporates observations obtained across multiple simulation specifications, representative test cases, propagation configurations, evaluation measures, verification activities, and validation-oriented simulation studies [8] [13]. The resulting observations remained operationally consistent across ambiguity-aware propagation evaluation, Q-State preservation, sparsity-aware traversal behavior, boundary-conditioned stabilization, and ambiguity retention measures introduced throughout Section 5.

Simulation evaluation measures, propagation-density observations, stabilization measures, admissible path continuity assessments, and ambiguity-retention analyses collectively provided additional support for the consistency of ambiguity-aware evaluative computational behavior across differing simulation environments and representative computational conditions [8] [12] [13].

A partial 4 KiB execution checkpoint dated 14 September 2026 records 16 executable local cases passing all 3573 predefined assertions, with 0 failed cases, 0 failed assertions, and 16 trace files[1] [30]. This result applies only to the executed checkpoint cases. The same record lists 16 unresolved or not-run entries, identifies the Online Build as a static source mapping rather than a completed browser-runtime execution, marks energy as unavailable, and states that the full simulation

[1]Artifact: Vorgentia_4KiB_Simulation_Checkpoint.xlsx; fixture SHA-256: c8f5d0341d54d951a71b136e6e2afcb14d11ed8489a7ae126a8fee0df6ecf193

is not complete.

Related external resources on applied scientific computing, large-scale simulation, software assurance, and verification provide broader methodological context for future extensions of the Section 5 workflow [21] [23]-[26] [31]. These sources are not presented as experimental validation of the Section 5 results.

6. Conclusion and Future Research Directions

6.1. Research Summary

This research introduced an ambiguity-aware computational framework for evaluating structurally admissible ambiguity within graph-connected computational systems under constrained evaluation conditions [8].

The framework established operational distinctions between Computing State and Computational State while introducing ambiguity-aware Q-State evaluation, boundary-conditioned admissibility, sparsity-aware propagation continuity, Non-Collapse semantics, and structurally constrained evaluative interpretation across graph-connected computational environments [8].

The framework further introduced an ambiguity-aware transformation pipeline supporting evaluative propagation, Q-State preservation, boundary-conditioned interpretation, and structurally constrained evaluative transition behavior under ambiguity-aware computational conditions [8].

Relative to the deterministic evaluation baseline considered in this study, the proposed framework preserves structurally admissible ambiguity while maintaining admissible propagation continuity, evaluative consistency, and stabilization behavior under the incomplete, conflicting, or evolving computational conditions evaluated here [3] [4] [8].

Simulation and experimental observations conducted across lattice-based evaluation environments, ambiguity-aware state-space simulations, Matplotlib-based stabilization studies, IBM Power-based propagation experimentation environments, Codex-assisted simulation analysis workflows, graph-connected computational evaluation systems, multiple simulation specifications, representative test cases, evaluation measures, verification activities, and validation-oriented simulation studies demonstrated operational continuity consistent with the ambiguity-aware computational framework established throughout Sections 3-5 [8] [13].

Collectively, the framework establishes a foundation for ambiguity-aware evaluative computational interpretation, admissibility-aware propagation evaluation, graph-connected state continuity, and structurally governed computational evaluation within graph-based computational systems [8] [19] [20].

6.2. Research Contributing Differentiators

The research presented throughout this paper introduces a collection of interconnected contributions spanning ambiguity-aware evaluation, graph-connected computational interpretation, simulation-oriented evaluation, and verification-oriented computational studies [8].

The differentiating contributions may be broadly grouped into three complementary areas.

Evaluation-Oriented Contributions

These contributions include Computing State and Computational State separation, Binary-Oriented Evaluation considerations, Absolute State evaluation, Q-State evaluation, Absolute State preservation, and Q-State preservation under ambiguity-aware computational conditions [8].

Structural and Interpretive Contributions

These contributions include Node, Node Object, and Object representation, Path Validity and Path Invalidity evaluation, Identity Preservation, Class Preservation, dependency-aware interpretation, and graph-connected evaluative continuity [8] [19] [20].

Simulation and Validation Contributions

These contributions include simulation specifications, multi-scenario evaluation activities, sparsity-derived confidence measures, simulation-driven verification and validation activities, acceptance-oriented evaluation, and counter-simulation response assessment [8] [13].

Collectively, these contributions support ambiguity-aware computational interpretation, admissibility-driven evaluation behavior, graph-connected continuity, identity-preserving evaluation, and simulation-supported verification activities under constrained computational conditions [8].

6.3. Future Applied and Cross-Domain Extensions

Future research continuity may further evaluate ambiguity-aware computational behavior across distributed evaluation environments, graph-connected computational systems, state-aware interpretive architectures, knowledge graph systems, probabilistic and graph-based computational models, and boundary-conditioned operational evaluation systems [7] [19] [20].

Additional research continuity may further examine ambiguity-aware evaluation behavior within AI/LLM interpretive systems, graph neural network environments, propagation-aware computational systems, hardware-constrained evaluation architectures, ambiguity-aware graph traversal systems, and computational state continuity under evolving operational conditions [32]-[35].

Future applied studies may additionally evaluate ambiguity-aware interpretation across incomplete, conflicting, uncertain, or evolving computational environments while preserving admissible ambiguity and structurally governed propagation continuity under constrained evaluative conditions [8].

Future research studies may additionally incorporate larger graph-connected evaluation environments, expanded simulation campaigns, additional validation studies, representative worked examples, reproducibility-oriented simulation configurations, and comparative evaluations against deterministic baseline approaches. Such studies may further clarify ambiguity-aware propagation behavior, Q-State preservation characteristics, stabilization measures, and evaluation con-

sistency across increasingly complex computational environments [8] [13].

Continued research studies may further examine Node, Node Object, and Object-oriented evaluative behavior, Path Validity and Path Invalidity assessment, Identity Preservation and Class Preservation characteristics, Absolute State preservation, Q-State preservation, and confidence-oriented evaluation measures derived from sparsity-aware computational behavior. Additional studies may further evaluate simulation-driven verification and validation workflows, specification-oriented simulation environments, and counter-simulation response assessment activities across graph-connected, ambiguity-aware computational systems [8].

Cross-domain operational continuity may additionally evaluate ambiguity-aware propagation behavior across simulation-driven evaluation environments, distributed graph systems, graph-connected state-transition systems, hardware-connected propagation architectures, and structurally governed computational interpretation systems [8] [19] [20].

Future cross-domain studies may additionally consider decision-support and conflict-simulation contexts [36] [37], responsible AI governance [38], and hardware-infrastructure constraints illustrated by high-performance computing cooling systems [22]. These sources identify possible application contexts and do not validate the framework.

Future computing-research studies may also examine state-dependent information entropy separately from thermodynamic entropy and record power, energy, heat load, cooling energy, temperature, and monetary cost as distinct observables [39]-[44]. In that research context, the AI Export Index is reserved for a disclosed cost-per-output measure; it is not an entropy, power, temperature, trade, or general AI-capability score. A proposed cost measure is $AEI_{cost} = C_{total} / U_{export}$ within a declared cost and output boundary; no validated numeric value is reported here because the required metered and normalized records are unavailable.

6.4. Supplemental Research Continuity

Ongoing supplemental research continuity may further extend the framework through additional simulation environments, applied ambiguity-aware interpretive examples, propagation-aware operational studies, cross-domain validation environments, graph-connected computational systems, and boundary-conditioned computational evaluation architectures [8] [13].

Additional supplemental studies may further examine ambiguity-aware state continuity, dynamic sparsity behavior, graph-connected evaluation dynamics, propagation continuity across probabilistic and ambiguity-aware environments, ambiguity retention behavior, and computational state-transition evaluation across specialized operational environments and interdisciplinary computational domains [8] [12].

The research presented throughout this paper establishes a continuing foundation for scientific, educational, computational, interdisciplinary, simulation-ori-

ented, verification-oriented, and operational research continuity under ambiguity-aware and constrained computational evaluation conditions [8].

Additional supplemental continuity may further examine Computing State and Computational State relationships, Absolute State and Q-State preservation behavior, Node, Node Object, and Object-oriented evaluation activities, Path Validity and Path Invalidity characteristics, confidence-oriented measures derived from sparsity-aware computational behavior, simulation-driven verification and validation activities, and specification-oriented simulation evaluation across representative ambiguity-aware computational environments [8] [13].

Research communication and public-facing dissemination may also draw on institutional scientific-computing communication resources [45].

Conflicts of Interest

The author declares no conflicts of interest regarding the publication of this paper.

References

[1] Patterson, D. and Hennessy, J. (1993) Computer Organization and Design: The Hardware/Software Interface. Morgan Kaufmann.

[2] Manna, Z. (1974) Mathematical Theory of Computation. McGraw-Hill.

[3] Hopcroft, J., Motwani, R. and Ullman, J. (2001) Introduction to Automata Theory, Languages, and Computation. Pearson.

[4] Gill, A. (1962) Introduction to the Theory of Finite-State Machines. McGraw-Hill.

[5] Pearl, J. (1988) Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann.

[6] Livi, L. and Rizzi, A. (2013) Graph Ambiguity. *Fuzzy Sets and Systems*, **221**, 24-47. https://doi.org/10.1016/j.fss.2013.01.001

[7] Koller, D. and Friedman, N. (2009) Probabilistic Graphical Models: Principles and Techniques. MIT Press. https://mitpress.mit.edu/9780262013192/probabilistic-graphical-models/

[8] Kumar, A., Abhishek, V. and Kumar, A. (2026) System and Method for Multi-State Ambiguity-Aware Computation Using Q-States and Sparsity Dynamics in Graph-Based Computational Systems. U.S. Provisional Patent Application No. 64/024,116.

[9] Strogatz, S. (1994) Nonlinear Dynamics and Chaos. Westview Press.

[10] Ogata, K. (1970) Modern Control Engineering. Prentice Hall.

[11] Khalil, H. (1996) Nonlinear Systems. Prentice Hall.

[12] Ren, L. (2024) Dynamic Sparsity Enabling Efficient, Interpretable, and Generalizable Sequence Models. Ph.D. Dissertation, University of Illinois. https://hdl.handle.net/2142/125587

[13] Kumar, A. (2026) VORG—Preferido—USA Ambiguity-Aware Multi-State Computation Framework. Research and Simulation Continuity Artifact.

[14] Hennessy, J. and Patterson, D. Computer Architecture: A Quantitative Approach. Morgan Kaufmann.

[15] Vorgentia Research (2026) Vorgentia Research Public Research Access and Ongoing Research Builds, Including Research Artifacts, Disclosure, Simulations, Participation

Pathways, and Vorgentia Research Milestones—2026. Site Build v16. https://vorgentia.github.io/

[16] Murialdo, M.R. (2025) Differentiation between Human-Generated and AI-Generated Text. U.S. Patent No. 12,476,822 B2. https://patents.google.com/patent/US12476822B2/en

[17] Hunter, J. (2007) Matplotlib: A 2D Graphics Environment. *IEEE Computing in Science and Engineering*, **9**, 90-95.

[18] Python Software Foundation. Python Computational Environment. https://www.python.org

[19] Diestel, R. (2017) Graph Theory. Springer.

[20] Barabási, A.-L. (2016) Network Science. Cambridge University Press.

[21] Lawrence Livermore National Laboratory (2026) CASC at 30: Advancing Discovery, Shaping the Future. Center for Applied Scientific Computing. https://computing.llnl.gov/about/newsroom/casc30-part1

[22] Lawrence Livermore National Laboratory (2026) Meet the Machines That Matter: How El Capitan Keeps Its Cool. https://www.llnl.gov/article/54496/meet-machines-matter-how-el-capitan-keeps-its-cool

[23] National Aeronautics and Space Administration (2026) Computing. https://www.nasa.gov/computing/

[24] NASA Advanced Supercomputing Division (2026) Supercomputing Facility at NAS. NASA Ames Research Center. https://www.nas.nasa.gov/supercomputing/overview.html

[25] Lawrence Berkeley National Laboratory. National Energy Research Scientific Computing Center (NERSC). DOE Office of Science User Facility and Scientific Computing Center. https://www.nersc.gov/

[26] Lawrence Berkeley National Laboratory. Applied Mathematics and Computational Research Division (AMCR). Mathematical Modeling, Simulation, Algorithm Design, Computer Architecture, and High-Performance Software Research. https://amcr.lbl.gov/

[27] Hagberg, A.A., Schult, D.A. and Swart, P.J. (2008) Exploring Network Structure, Dynamics, and Function Using NetworkX. Proceedings of the 7th Python in Science Conference, 11-15. https://doi.org/10.25080/TCWV9851

[28] Linux Kernel Documentation (2024-2025) Introduction of Mseal. Userspace API. https://docs.kernel.org/userspace-api/mseal.html

[29] Linux Kernel Newbies (2024) Linux 6.10 Release Summary, Including Mseal(2). https://kernelnewbies.org/Linux_6.10

[30] Vorgentia Research (2026) Vorgentia 4 KiB Simulation Checkpoint. Internal Execution-and-Test Evidence Workbook.

[31] Lawrence Livermore National Laboratory. Software Assurance. Global Security. https://gs.llnl.gov/defense/software-assurance

[32] Russell, S. and Norvig, P. (2010) Artificial Intelligence: A Modern Approach. Pearson.

[33] Goodfellow, I., Bengio, Y. and Courville, A. (2016) Deep Learning. MIT Press.

[34] Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C. and Yu, P.S. (2021) A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, **32**, 4-24. https://doi.org/10.1109/tnnls.2020.2978386

[35] Qu, M., Bengio, Y. and Tang, J. (2019) GMNN: Graph Markov Neural Networks.

Proceedings of the 36th International Conference on Machine Learning, **97**, 5241-5250. https://proceedings.mlr.press/v97/qu19a.html

[36] Lawrence Livermore National Laboratory (2026) Decision Support. Global Security. https://gs.llnl.gov/defense/decision-support

[37] Lawrence Livermore National Laboratory (2026) Conflict Simulation. Global Security. https://gs.llnl.gov/defense/conflict-simulation

[38] National Aeronautics and Space Administration (2026) NASA Framework for the Ethical Use of Artificial Intelligence. https://www.nasa.gov/nasa-artificial-intelligence-ethics/

[39] Shannon, C.E. (1948) A Mathematical Theory of Communication. *Bell System Technical Journal*, **27**, 379-423. https://doi.org/10.1002/j.1538-7305.1948.tb01338.x

[40] Landauer, R. (1961) Irreversibility and Heat Generation in the Computing Process. *IBM Journal of Research and Development*, **5**, 183-191. https://doi.org/10.1147/rd.53.0183

[41] Bennett, C.H. (1982) The Thermodynamics of Computation—A Review. *International Journal of Theoretical Physics*, **21**, 905-940. https://doi.org/10.1007/bf02084158

[42] International Energy Agency (2025) Energy and AI. https://www.iea.org/reports/energy-and-ai

[43] Stanford Institute for Human-Centered Artificial Intelligence (2026) Research and Development. The 2026 AI Index Report. https://hai.stanford.edu/ai-index/2026-ai-index-report/research-and-development

[44] Tschand, A., Rajan, A.T.R., Idgunji, S. *et al.* (2024) MLPerf Power: Benchmarking the Energy Efficiency of Machine Learning Systems from Microwatts to Megawatts for Sustainable AI. arXiv:2410.12032. https://arxiv.org/abs/2410.12032

[45] Computing Sciences Communications Group (2024) For the Media. https://cs.lbl.gov/news-and-events/for-the-media/