Preprints.org

Article

Not peer-reviewed version

Towards a Universal World Model for Artificial General Intelligence

Harris Wang *

Posted Date: 29 January 2026

doi: 10.20944/preprints202601.1685.v2

Keywords: artificial general intelligence; world models; constraint-based systems; hierarchical reasoning; symbolic-neural integration; complex world systems modelling

Preprints.org is a free multidisciplinary platform providing preprint service that is dedicated to making early versions of research outputs permanently available and citable. Preprints posted at Preprints.org appear in Web of Science, Crossref, Google Scholar, Scilit, Europe PMC.

Copyright: This open access article is published under a Creative Commons CC BY 4.0 license, which permit the free download, distribution, and reuse, provided that the author and preprint are cited in any reuse.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 January 2026 doi:10.20944/preprints202601.1685.v2

Disclaimer/Publisher's Note: The statements, opinions, and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions, or products referred to in the content.

Article

Towards a Universal World Model for Artificial General Intelligence

Harris Wang

School of Computing and Information Systems, Athabasca University, Athabasca, Canada;
harrisw@athabascau.ca

Abstract

Constrained Object Hierarchies (COH) presents a neuroscience-grounded theoretical framework for artificial general intelligence that models intelligent systems as hierarchical structures of objects constrained by multi-domain rules. This paper demonstrates how COH, implemented through the General Intelligent System Modelling Language (GISMOL), serves as a universal world model capable of representing complex world systems across diverse domains including healthcare, finance, manufacturing, climate science, education, and urban governance. We present a comprehensive analysis of six complex world systems modelled using the COH 9-tuple formalization, detailing their implementation through GISMOL's integrated architecture encompassing symbolic reasoning, neural computation, natural language processing, and constraint management. Our framework bridges the symbolic-neural divide, avoids "jagged intelligence" through coherent constraint propagation, and enables the development of agentic systems with general intelligence capabilities. The paper contributes a unified modelling paradigm that advances AGI research by providing a practical, implementable framework for building intelligent systems that operate consistently across domains while respecting domain-specific constraints and requirements.

Keywords: artificial general intelligence; world models; constraint-based systems; hierarchical reasoning; symbolic-neural integration; complex world systems modelling

1. Introduction

Artificial General Intelligence (AGI) research has long sought a unifying framework capable of representing diverse domains while maintaining coherent reasoning across abstraction levels [1]. Existing approaches often suffer from either excessive specialization that prevents generalization or excessive abstraction that loses domain-specific nuances [2]. The Constrained Object Hierarchies (COH) theoretical framework addresses this challenge through a neuroscience-grounded approach that models intelligent systems as hierarchical structures of objects constrained by multi-domain rules [3,4].

COH formalizes intelligent systems through a 9-tuple representation: O = (C, A, M, N, E, I, T, G, D), where C represents components (sub-objects in compositional hierarchy), A denotes attributes (state variables), M indicates methods (executable actions), N symbolizes neural components (adaptive models), E embodies the embedding component for semantic operations, I represents identity constraints (fundamental rules), T denotes trigger constraints (event-condition-action rules), G symbolizes goal constraints (optimization objectives), and D represents constraint daemons (real-time monitors). This formalization provides a comprehensive representation that captures both structural and behavioral aspects of intelligent systems.

The General Intelligent System Modelling Language (GISMOL) implements the COH framework as a practical Python toolkit, providing developers with the means to construct intelligent systems that respect domain constraints while exhibiting adaptive, general-purpose reasoning capabilities [5]. GISMOL comprises several integrated modules: gismol.core provides the

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 January 2026 doi:10.20944/preprints202601.1685.v2

fundamental COHObject implementation and constraint management, gismol.neural offers neural component integration with constraint awareness, gismol.reasoners implements specialized reasoning engines across domains, and gismol.nlp enables natural language interaction with constraint validation. This integrated architecture enables the construction of executable intelligent systems that maintain coherence across diverse application domains.

This paper demonstrates how COH/GISMOL serves as a universal world model for AGI through detailed case studies across six complex domains, showing how the framework provides consistent modelling and implementation patterns while respecting domain-specific requirements. We further illustrate how the framework integrates symbolic and neural computation, avoids "jagged intelligence" through coherent constraint propagation, and enables the development of agentic systems with general intelligence capabilities.

2. Literature Review

World modelling approaches in AI have evolved from early symbolic systems to modern neural architectures, each with distinct strengths and limitations. Classic symbolic AI frameworks, such as Cyc [6] and Soar [7], demonstrated robust reasoning but struggled with uncertainty handling and scalability. Connectionist approaches, particularly deep learning [8], excelled at pattern recognition but lacked interpretability and struggled with systematic generalization [9].

Hybrid systems have attempted to bridge this divide, with neuro-symbolic approaches [10,11] combining neural perception with symbolic reasoning. However, these often maintain separate processing pathways that require explicit integration mechanisms [12]. Probabilistic programming languages [13] and hierarchical Bayesian models [14] provide uncertainty-aware reasoning but typically lack explicit constraint management across abstraction levels.

Cognitive architectures like ACT-R [15] and LIDA [16] incorporate neuroscientific principles but focus primarily on cognitive processes rather than general system modelling. More recently, large language models [17] have demonstrated impressive language capabilities but exhibit "jagged intelligence" [18]—uneven performance across different reasoning tasks and difficulty maintaining consistent constraints.

Constraint-based reasoning has a rich history in AI, from constraint satisfaction problems [19] to constraint programming languages [20]. However, these typically operate within narrow domains or lack integration with learning systems. Hierarchical temporal memory [21] provides a neuroscience-inspired approach to sequence learning but lacks explicit constraint representation.

The COH framework differs from these approaches by providing a unified representation that integrates hierarchical decomposition, multi-domain constraints, neural computation, and real-time monitoring within a single coherent formalism. Unlike hybrid systems that maintain separate symbolic and neural components, COH embeds neural computation within the object hierarchy, ensuring consistent constraint propagation across all system elements.

3. Workflow of Modelling and Implementing Complex World Systems

The COH/GISMOL framework provides a systematic workflow for modelling and implementing complex real-world systems, balancing domain specificity with general modelling principles. The workflow begins with **domain analysis**, during which key entities, relationships, and constraints are identified. This is followed by **hierarchical decomposition**, which establishes the system's compositional and organisational structure. The designer then formalizes the system using the **COH 9-tuple**, specifying components, attributes, methods, neural elements, embeddings, and constraints.

The **implementation phase** leverages GISMOL's integrated modules to construct executable systems. The gismol.core module provides the foundational COHObject class and the underlying constraint-management infrastructure. Domain-specific reasoning capabilities are incorporated through gismol.reasoners, which includes specialised reasoners for biological, physical, geometric,

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 January 2026 doi:10.20944/preprints202601.1685.v2

temporal, causal, probabilistic, and other classes of domain constraints. Adaptive behaviour and learning are enabled through neural components in gismol.neural, while natural-language interaction and knowledge extraction are supported by gismol.nlp.

Throughout implementation, **constraint validation** ensures system coherence and correctness, with daemons providing real-time monitoring of constraint satisfaction. The hierarchical structure enables efficient propagation of constraints and state changes across different levels of abstraction, while the neural components allow the system to adapt to novel situations without violating core constraints. Overall, this workflow yields systems that preserve domain-specific integrity while exhibiting general, transferable intelligent capabilities.

4. Analysis, Design, Modelling, and Implementation of Complex World Systems

4.1. Healthcare: Hospital Pandemic Response System

System Description and Significance: Hospital pandemic response systems must coordinate limited resources, evolving medical knowledge, and unpredictable patient influxes during health crises. Effective systems can dramatically reduce mortality rates while maintaining staff well-being and operational continuity [22].

Analysis and Requirements: The system must handle extreme uncertainty, ethical triage decisions, regulatory compliance, and human factors including staff mental health. Key requirements include dynamic resource allocation, real-time constraint monitoring, adaptive protocols based on evolving medical understanding, and integration of diverse data sources.

COH Solution Design: The system is designed as a hierarchical structure with hospital departments containing wards, which contain patient beds and medical equipment. Neural components predict epidemic spread and resource needs, while constraint daemons monitor critical thresholds.

COH 9-Tuple Formalization:

C: Hospital→Departments→Wards→Beds→Equipment→Staff teams

A: Bed occupancy, PPE inventory, staff fatigue, patient acuity, ventilator availability

M: Triage protocols, resource allocation, staff rotation, decontamination procedures

N: Epidemic spread predictors, patient outcome forecasters, resource optimizers

E: Multi-modal medical data embedding (clinical notes, lab results, imaging)

I: HIPAA compliance, sterilization standards, staff certification, emergency protocols

T: ICU capacity>90%→activate surge; staff infection>5%→implement quarantine

G: Minimize mortality, maximize resource efficiency, minimize staff burnout

D: Bed tracking, supply chain alerts, infection cluster detection

Modelling Justification: The hierarchical decomposition mirrors hospital organizational structure while enabling efficient constraint propagation. Neural components provide adaptive response to evolving pandemic patterns. Constraint daemons enable proactive intervention before critical thresholds are breached.

GISMOL Implementation:

```
from gismol.core import COHObject, COHRepository
from gismol.reasoners import ResourceReasoner, TemporalReasoner, BiologicalReasoner
from gismol.neural import NeuralDurationPredictor, AnomalyDetector

class HospitalPandemicSystem(COHObject):
    def __init__(self, name="HospitalPandemicSystem"):
        super().__init__(name)

        # Hierarchical decomposition
        self.add_component(COHObject("EmergencyDepartment"))
        self.add_component(COHObject("ICUDepartment"))
```

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 January 2026 doi:10.20944/preprints202601.1685.v2


```
        self.add_component(COHObject("LogisticsDepartment"))

        # Neural components for prediction
        self.add_neural_component(
            "epidemic_predictor",
            NeuralDurationPredictor(input_dim=32, hidden_dim=128)
        )
        self.add_neural_component(
            "resource_anomaly_detector",
            AnomalyDetector(input_dim=24, hidden_dim=64)
        )

        # Critical identity constraints
        self.add_identity_constraint({
            'category': 'safety',
            'name': 'ppe_minimum',
            'specification': 'ppe_inventory >= 48_hour_supply',
            'severity': 10
        })

        # Trigger constraints for automated response
        self.add_trigger_constraint({
            'name': 'surge_activation',
            'type': 'operational',
            'precondition': 'icu_occupancy > 0.9',
            'action': 'activate_surge_protocol',
            'postcondition': 'surge_capacity_available == True'
        })

        # Register domain-specific reasoners
        from gismol.core import ConstraintSystem
        self.constraint_system = ConstraintSystem()
        self.constraint_system.register_reasoner("biological", BiologicalReasoner())
        self.constraint_system.register_reasoner("resource", ResourceReasoner())
        self.constraint_system.register_reasoner("temporal", TemporalReasoner())

    # Benefits of hierarchical decomposition:
    # - Resource constraints propagate from hospital to department to ward level
    # - State changes at bed level automatically update ward and department aggregates
    # - Emergency protocols can be activated at appropriate granularity levels

    # Neural component interaction:
    epidemic_predictor = hospital_system.get_neural_component("epidemic_predictor")
    prediction = epidemic_predictor.predict(current_patient_influx)
    # Prediction influences resource allocation constraints and trigger thresholds

    # Constraint propagation:
    # When ICU occupancy increases, it triggers evaluation of ICU capacity constraints
    # at multiple hierarchy levels simultaneously
```

Critical Comparison: Traditional hospital management systems typically rely on loosely coupled modules for scheduling, inventory control, and patient tracking, resulting in limited

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 January 2026 doi:10.20944/preprints202601.1685.v2

cross-module integration and fragmented decision-making [23]. In contrast, COH enables unified constraint management across all operational dimensions, supporting coherent decisions that explicitly account for interdependencies among resource allocation, staff coordination, and patient care activities.

4.2. Finance: Global Cryptocurrency Market Ecosystem

System Description and Significance: Cryptocurrency markets exhibit extreme volatility, 24/7 operation, decentralized governance, and complex regulatory landscapes. Effective monitoring systems must detect anomalies, ensure compliance, and manage systemic risk across global, interconnected markets.

Analysis and Requirements: The system must operate across conflicting jurisdictions, handle pseudonymous actors, adapt to rapid technological evolution, and balance security with usability. Requirements include real-time anomaly detection, regulatory compliance monitoring, risk assessment, and automated response to market manipulations.

COH Solution Design: The ecosystem is modelled as a hierarchy of exchange platforms containing trading pairs with associated order books, smart contracts, and wallet addresses. Neural components detect market manipulations and predict price movements, while constraint daemons monitor for regulatory violations and security breaches.

COH 9-Tuple Formalization:

C: Exchange platforms→Trading pairs→Order books→Smart contracts→Wallet addresses

A: Liquidity volumes, volatility indices, regulatory status, transaction fees, hash rates

M: Arbitrage algorithms, risk assessment, compliance verification, smart contract deployment

N: Price movement predictors, anomaly detectors, sentiment analyzers, flash crash anticipators

E: Cross-chain transaction semantics, token relationship mapping, regulatory jurisdiction embeddings

I: Cryptography security requirements, consensus protocol rules, anti-money laundering constraints

T: Price deviation>20%→trigger arbitrage; unusual wallet activity→flag for investigation

G: Maximize portfolio returns, minimize systemic risk, ensure market integrity, optimize efficiency

D: 24/7 market surveillance, regulatory update monitors, smart contract vulnerability scanners

Modelling Justification: The hierarchical structure captures market organization while enabling risk assessment at multiple levels. Neural components handle extreme non-linearity and emergent market behaviors. Identity constraints enforce both technical (cryptographic) and regulatory requirements simultaneously.

GISMOL Implementation:

```
from gismol.core import COHObject
from gismol.reasoners import ProbabilisticReasoner, TemporalReasoner, SafetyReasoner
from gismol.neural import AnomalyDetector, NeuralDurationPredictor

class CryptoMarketSystem(COHObject):
    def __init__(self, name="CryptoMarketSystem"):
        super().__init__(name)

        # Multi-level hierarchy
        self.add_component(COHObject("ExchangePlatforms"))
        self.add_component(COHObject("TradingPairs"))
        self.add_component(COHObject("SmartContractNetwork"))

        # Neural market analysis
        self.add_neural_component(
```

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 January 2026 doi:10.20944/preprints202601.1685.v2


```
        "anomaly_detector",
        AnomalyDetector(input_dim=64, hidden_dim=256)
    )
    self.add_neural_component(
        "sentiment_analyzer",
        TextEmbedder()  # From gismol.nlp
    )

    # Security and regulatory constraints
    self.add_identity_constraint({
        'category': 'security',
        'name': 'transaction_validation',
        'specification': 'all transactions satisfy cryptographic_verification',
        'severity': 10
    })

    # Automated trading triggers
    self.add_trigger_constraint({
        'name': 'arbitrage_opportunity',
        'type': 'trading',
        'precondition': 'price_discrepancy(exchange_A, exchange_B) > 0.2',
        'action': 'execute_arbitrage',
        'postcondition': 'portfolio_value_increased == True'
    })

    # Start monitoring daemons
    self.start_daemons()

# Constraint propagation across hierarchy:
# A smart contract vulnerability at contract level triggers re-evaluation
# of all dependent trading pairs and exchange platforms

# Neural-symbolic integration:
anomaly_score = self.get_neural_component("anomaly_detector").detect(market_data)
if anomaly_score > threshold:
    # Symbolic constraints evaluate appropriate response
    self.constraint_system.evaluate("security_response", context)
```

Critical Comparison: Traditional financial systems use rule-based alerts with limited learning capabilities [24]. COH integrates continuous learning with explicit constraint enforcement, enabling systems that adapt to evolving market manipulation techniques while maintaining regulatory compliance.

4.3. Smart Manufacturing: Automotive Production Network

System Description and Significance: Modern automotive manufacturing involves complex supply chains, robotic assembly lines, quality control systems, and just-in-time production schedules. Efficient systems reduce waste, improve quality, and enable mass customization while maintaining safety standards.

Analysis and Requirements: The system must handle physical constraints (machine wear), human-robot collaboration, supply chain disruptions, and customized production requirements. Key needs include predictive maintenance, dynamic scheduling, quality defect detection, and energy optimization.

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 January 2026 doi:10.20944/preprints202601.1685.v2

COH Solution Design: The manufacturing network is structured hierarchically with factories containing production lines, robotic cells, IoT sensors, and individual components. Neural components predict equipment failures and optimize schedules, while constraint daemons monitor quality metrics and resource utilization.

COH 9-Tuple Formalization:

C: Factories→Production lines→Robotic cells→IoT sensors→Components

A: Machine health scores, production rates, defect frequencies, energy consumption, lead times

M: Predictive maintenance, dynamic scheduling, quality control, energy optimization

N: Failure predictors, supply chain forecasters, defect classifiers, demand predictors

E: Multi-modal production data embedding (vibration, thermal, visual, acoustic)

I: Safety standards (ISO 26262), precision tolerances, environmental regulations

T: Abnormal vibration→schedule maintenance; supplier delay>48h→activate alternative sourcing

G: Maximize Overall Equipment Effectiveness, minimize waste, optimize energy usage

D: Real-time OEE monitors, quality deviation detectors, supply chain disruption alerts

Modelling Justification: The hierarchical decomposition matches physical manufacturing organization while enabling constraint propagation from component to factory level. Neural components enable predictive capabilities that traditional control systems lack. Constraint daemons provide continuous monitoring essential for lean manufacturing.

GISMOL Implementation:

```
from gismol.core import COHObject, COHRelation
from gismol.reasoners import PhysicalReasoner, GeometricReasoner, TemporalReasoner
from gismol.neural import NeuralSchedulingModel, NeuralResourceMatcher

class AutomotiveManufacturingSystem(COHObject):
    def __init__(self, name="AutomotiveManufacturingSystem"):
        super().__init__(name)

        # Physical hierarchy
        assembly_line = COHObject("AssemblyLine")
        assembly_line.add_component(COHObject("RoboticArm"))
        assembly_line.add_component(COHObject("WeldingStation"))
        assembly_line.add_component(COHObject("QualityCamera"))
        self.add_component(assembly_line)

        # Neural scheduling and matching
        self.add_neural_component(
            "production_scheduler",
            NeuralSchedulingModel(task_dim=64, resource_dim=32, hidden_dim=256)
        )
        self.add_neural_component(
            "resource_matcher",
            NeuralResourceMatcher(capability_list=['welding', 'painting', 'assembly'])
        )

        # Safety and quality constraints
        self.add_identity_constraint({
            'category': 'safety',
            'name': 'human_robot_distance',
            'specification': 'distance(human_worker, robot) > 1.5m',
            'severity': 10
```

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 January 2026 doi:10.20944/preprints202601.1685.v2


```
    })

    # Goal constraint for optimization
    self.add_goal_constraint({
        'name': 'maximize_oee',
        'type': 'optimization',
        'objective': 'maximize(availability * performance * quality)',
        'priority': 'HIGH'
    })

    # Relationships between components
    relation = COHRelation()
    relation.add_relation(
        self.get_component("RoboticArm"),
        self.get_component("WeldingStation"),
        "equipment",
        "arm_welder_connection"
    )

# Hierarchical constraint benefits:
# A defect at component level triggers quality checks at cell, line, and factory levels
# Energy optimization constraints propagate downward to individual machines

# Neural-symbolic interaction:
schedule = self.get_neural_component("production_scheduler").generate(
    task_hierarchy=manufacturing_tasks,
    resource_pool=factory_resources
)
# Generated schedule validated against all constraints before execution
if self.constraint_system.validate_schedule(schedule):
    self.execute_schedule(schedule)
```

Critical Comparison: Traditional manufacturing execution systems use rigid workflows with limited adaptability [25]. COH enables dynamic reconfiguration in response to disruptions while maintaining safety and quality constraints, supporting the transition to Industry 4.0 with flexible, intelligent production systems.

4.4. Climate Science: Global Carbon Cycle Management

System Description and Significance: Climate systems involve complex interactions between atmospheric, oceanic, terrestrial, and biological processes across multiple spatial and temporal scales. Effective management requires balancing immediate human needs with long-term planetary health.

Analysis and Requirements: The system must handle extreme spatial/temporal scales, deep uncertainty in climate models, political-economic constraints, and non-linear feedback loops. Requirements include tipping point detection, carbon sequestration optimization, biodiversity preservation, and climate resilience assessment.

COH Solution Design: The carbon cycle is modelled as a nested hierarchy from cellular processes to planetary systems. Neural components predict climate impacts and ecosystem responses, while constraint daemons monitor planetary boundaries and biodiversity indicators.

COH 9-Tuple Formalization:

C: Biomes→Ecosystems→Species populations→Organisms→Cellular processes

A: Carbon sequestration rates, biodiversity indices, temperature anomalies, precipitation patterns

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 January 2026 doi:10.20944/preprints202601.1685.v2

M: Reforestation protocols, carbon credit verification, conservation strategies, climate modelling
N: Climate impact predictors, tipping point warning systems, ecosystem resilience estimators
E: Cross-scale ecological process embedding (molecular to planetary)
I: Planetary boundaries (climate, biodiversity), conservation laws, international treaties
T: Deforestation rate>X%→trigger intervention; ocean pH<threshold→initiate mitigation
G: Maximize carbon sequestration, minimize biodiversity loss, optimize climate resilience
D: Real-time deforestation monitoring, carbon flux trackers, biodiversity loss detectors

Modelling Justification: The nested hierarchy captures ecological scales while enabling multi-level intervention strategies. Neural components model complex, non-linear climate feedback loops. Identity constraints encode hard planetary boundaries that cannot be violated without catastrophic consequences.

GISMOL Implementation:

```
from gismol.core import COHObject
from gismol.reasoners import BiologicalReasoner, CausalReasoner, TemporalReasoner
from gismol.neural import NeuralDurationPredictor
from gismol.nlp import Text2COH

class CarbonCycleSystem(COHObject):
    def __init__(self, name="CarbonCycleSystem"):
        super().__init__(name)

        # Ecological hierarchy
        self.add_component(COHObject("AtmosphericLayer"))
        self.add_component(COHObject("OceanicSystem"))
        terrestrial = COHObject("TerrestrialBiomes")
        terrestrial.add_component(COHObject("ForestEcosystems"))
        terrestrial.add_component(COHObject("AgriculturalSystems"))
        self.add_component(terrestrial)

        # Neural climate prediction
        self.add_neural_component(
            "climate_predictor",
            NeuralDurationPredictor(input_dim=128, hidden_dim=512)
        )

        # Planetary boundary constraints (identity)
        self.add_identity_constraint({
            'category': 'planetary',
            'name': 'co2_boundary',
            'specification': 'atmospheric_co2 < 450 ppm',
            'severity': 10   # Absolute constraint
        })

        # Trigger constraints for intervention
        self.add_trigger_constraint({
            'name': 'deforestation_response',
            'type': 'conservation',
            'precondition': 'deforestation_rate > 0.5%_per_year',
            'action': 'activate_conservation_protocol',
            'postcondition': 'protected_areas_increased == True'
        })
```

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 January 2026 doi:10.20944/preprints202601.1685.v2


```
        # NLP for policy document processing
        self.add_neural_component(
            "policy_analyzer",
            Text2COH()  # Converts policy documents to COH constraints
        )

    # Hierarchical carbon accounting:
    # Carbon sequestration at organism level aggregates to ecosystem and biome levels
    # Interventions can be targeted at appropriate hierarchy levels

    # Neural-symbolic climate modelling:
    climate_prediction = self.get_neural_component("climate_predictor").predict(
        current_emissions,
        economic_models
    )
    # Prediction informs which constraints are most critical to enforce
```

Critical Comparison: Traditional climate models focus on physical processes with limited integration of biological and social factors [26]. COH provides integrated modelling of physical, biological, and socioeconomic dimensions with explicit constraint management across scales, enabling more comprehensive climate policy assessment.

4.5. Education: Personalized Learning Ecosystem

System Description and Significance: Personalized learning systems adapt educational content, pacing, and assessment to individual learner needs, improving outcomes while addressing achievement gaps. Effective systems balance standardization with personalization across diverse learner populations.

Analysis and Requirements: The system must handle diverse backgrounds, learning styles, and cognitive abilities while maintaining educational standards and addressing ethical concerns around data privacy and algorithmic bias. Requirements include adaptive learning pathways, engagement monitoring, knowledge gap detection, and equitable access.

COH Solution Design: The learning ecosystem is structured hierarchically with educational institutions containing classrooms, learning groups, individual students, and cognitive processes. Neural components predict learning outcomes and optimize engagement, while constraint daemons monitor equity gaps and intervention effectiveness.

COH 9-Tuple Formalization:

C: Institutions→Classrooms→Learning groups→Students→Cognitive processes

A: Learning proficiency, engagement metrics, retention rates, socio-economic factors, learning styles

M: Adaptive learning pathways, assessment generation, intervention strategies, collaborative design

N: Learning outcome predictors, engagement optimizers, knowledge gap identifiers, at-risk detectors

E: Multi-modal learning data embedding (performance, behavior, socio-emotional)

I: Educational standards (Common Core), accessibility requirements, privacy laws, ethical guidelines

T: Student engagement<threshold→adapt content; knowledge gap detected→provide targeted practice

G: Maximize learning outcomes, minimize achievement gaps, optimize engagement, develop critical thinking

D: Real-time engagement monitors, learning progress trackers, equity gap detectors

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 January 2026 doi:10.20944/preprints202601.1685.v2

Modelling Justification: The hierarchy reflects educational organization while capturing individual cognitive diversity. Neural components enable true personalization at scale impossible with rule-based systems. Constraint daemons ensure equitable access and continuous improvement while respecting privacy constraints.

GISMOL Implementation:

```
from gismol.core import COHObject
from gismol.reasoners import TemporalReasoner, CardinalityReasoner
from gismol.neural import Classifier, Regressor
from gismol.nlp import COHDialogueManager

class PersonalizedLearningSystem(COHObject):
    def __init__(self, name="PersonalizedLearningSystem"):
        super().__init__(name)

        # Educational hierarchy
        self.add_component(COHObject("MathematicsDepartment"))
        self.add_component(COHObject("ScienceDepartment"))

        # Neural learning analytics
        self.add_neural_component(
            "outcome_predictor",
            Classifier(input_dim=128, output_dim=5)   # Predict grade outcomes
        )
        self.add_neural_component(
            "engagement_optimizer",
            Regressor(input_dim=64, output_dim=1)   # Predict engagement score
        )

        # Ethical and regulatory constraints
        self.add_identity_constraint({
            'category': 'ethics',
            'name': 'privacy_protection',
            'specification': 'student_data encrypted and anonymized',
            'severity': 10
        })

        # Personalization triggers
        self.add_trigger_constraint({
            'name': 'adaptive_content',
            'type': 'pedagogical',
            'precondition': 'student.engagement_score < 0.6',
            'action': 'adjust_content_difficulty',
            'postcondition': 'student.engagement_score >= 0.7'
        })

        # NLP for student interaction
        self.add_neural_component(
            "dialogue_manager",
            COHDialogueManager()   # Constraint-aware tutoring conversations
        )
```

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 January 2026 doi:10.20944/preprints202601.1685.v2


```
# Hierarchical personalization:
# Learning strategies propagate from institutional goals to individual adaptations
# Group interventions can be coordinated while respecting individual needs

# Neural constraint integration:
engagement_prediction = self.get_neural_component("engagement_optimizer").predict(
    student_data
)
if engagement_prediction < threshold:
    # Trigger evaluates alternative teaching strategies within constraints
    self.evaluate_constraint("adaptive_content", context)
```

Critical Comparison: Traditional learning management systems provide content delivery with limited personalization [27]. Intelligent tutoring systems offer adaptation but often operate in isolation from institutional constraints. COH enables personalized learning within institutional frameworks, ensuring interventions respect educational standards, privacy requirements, and equity considerations.

4.6. Urban Governance: Smart City Infrastructure

System Description and Significance: Smart cities integrate IoT sensors, data analytics, and automated systems to optimize urban services, enhance citizen well-being, and improve resource efficiency. Effective governance balances technological efficiency with human-centric design and democratic oversight.

Analysis and Requirements: The system must handle conflicting stakeholder interests, integrate hard infrastructure with soft social systems, ensure democratic oversight of automation, and maintain resilience to disruptions. Requirements include traffic optimization, energy management, emergency response coordination, and equitable service delivery.

COH Solution Design: The urban system is structured spatially with city containing districts, neighborhoods, buildings, and infrastructure components. Neural components predict urban growth and optimize services, while constraint daemons monitor resource use and social indicators.

COH 9-Tuple Formalization:

C: City→Districts→Neighborhoods→Buildings→Infrastructure components

A: Traffic flow rates, energy consumption, air quality indices, service demand, social cohesion

M: Traffic light optimization, emergency response coordination, utility load balancing, space management

N: Urban growth predictors, infrastructure failure forecasters, social tension detectors, demand anticipators

E: Multi-source urban data embedding (IoT, social media, government records)

I: Building codes, zoning regulations, environmental standards, accessibility requirements, equity mandates

T: Air quality>hazardous level→restrict traffic; emergency event→activate response protocols

G: Maximize citizen well-being, minimize resource consumption, optimize service delivery, enhance resilience

D: Infrastructure health monitors, social indicator trackers, resource scarcity alerts, equity verifiers

Modelling Justification: The spatial hierarchy matches urban organization while enabling multi-scale governance. Neural components handle complex urban dynamics and emergent behaviors. Constraint daemons provide the continuous monitoring needed for proactive governance that balances efficiency with equity.

GISMOL Implementation:

```
from gismol.core import COHObject
from gismol.reasoners import GeometricReasoner, ResourceReasoner, TemporalReasoner
```

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 January 2026 doi:10.20944/preprints202601.1685.v2


```
from gismol.neural import NeuralSchedulingModel, AnomalyDetector

class SmartCitySystem(COHObject):
    def __init__(self, name="SmartCitySystem"):
        super().__init__(name)

        # Spatial hierarchy
        downtown = COHObject("DowntownDistrict")
        downtown.add_component(COHObject("TrafficNetwork"))
        downtown.add_component(COHObject("EnergyGrid"))
        self.add_component(downtown)

        # Neural urban management
        self.add_neural_component(
            "traffic_optimizer",
            NeuralSchedulingModel(task_dim=32, resource_dim=16, hidden_dim=128)
        )
        self.add_neural_component(
            "infrastructure_anomaly_detector",
            AnomalyDetector(input_dim=48, hidden_dim=192)
        )

        # Equity and accessibility constraints
        self.add_identity_constraint({
            'category': 'equity',
            'name': 'service_accessibility',
            'specification': 'all neighborhoods have essential_services_within_1km',
            'severity': 8
        })

        # Emergency response triggers
        self.add_trigger_constraint({
            'name': 'emergency_coordination',
            'type': 'safety',
            'precondition': 'major_incident_detected == True',
            'action': 'activate_emergency_response',
            'postcondition': 'emergency_services_deployed == True'
        })

        # Start all monitoring daemons
        self.start_daemons()

# Hierarchical governance:
# Traffic policies at city level propagate to district and neighborhood implementations
# Emergency responses coordinate across multiple hierarchy levels simultaneously

# Neural optimization with constraints:
optimal_traffic_plan = self.get_neural_component("traffic_optimizer").generate(
    traffic_patterns,
    road_network
)
```

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 January 2026 doi:10.20944/preprints202601.1685.v2


```
# Plan validated against equity, safety, and environmental constraints
if self.constraint_system.validate_plan(optimal_traffic_plan):
    self.implement_traffic_plan(optimal_traffic_plan)
```

Critical Comparison: Traditional city management uses siloed systems for transportation, energy, and public safety with limited integration [28]. COH enables unified governance that considers interdependencies between urban systems while respecting legal constraints and equity requirements, supporting the development of truly intelligent cities.

5. Implementing COH Models into Python Applications[1]

COH models transition from formal specifications to executable systems through GISMOL's integrated implementation architecture. This process begins with instantiating COHObjects that represent the hierarchical structure, followed by configuring neural components for adaptive behavior, and finally activating constraint daemons for autonomous monitoring.

The implementation leverages GISMOL's multi-module architecture: gismol.core provides the foundational objects and constraint management, gismol.neural supplies adaptive components, gismol.reasoners offers domain-specific reasoning, and gismol.nlp enables natural language interaction. These modules work synergistically to create systems that maintain coherence across domains while respecting domain-specific requirements.

```
# Complete implementation example: Healthcare Pandemic Response System
from gismol.core import COHObject, ConstraintSystem
from gismol.neural import NeuralDurationPredictor, AnomalyDetector
from gismol.reasoners import BiologicalReasoner, ResourceReasoner, TemporalReasoner
from gismol.nlp import COHDialogueManager

class ExecutablePandemicSystem:
    def __init__(self):
        # Create COH hierarchy
        self.hospital = COHObject("MetropolitanHospital")

        # Add components with inheritance of constraints
        emergency_dept = COHObject("EmergencyDepartment")
        emergency_dept.add_identity_constraint({
            'name': 'triage_capacity',
            'specification': 'triage_wait_time < 30_minutes'
        })
        self.hospital.add_component(emergency_dept)

        # Add neural predictive components
        self.hospital.add_neural_component(
            "patient_inflow_predictor",
            NeuralDurationPredictor(input_dim=16, hidden_dim=64)
        )

        # Configure constraint system with domain reasoners
        self.constraint_system = ConstraintSystem()
        self.constraint_system.register_reasoner("biological", BiologicalReasoner())
```

[1] Because the supporting GISMOL library is still under development and has not yet been released publicly, the GISMOL code presented in this paper is intended solely to illustrate the semantics and programming logic of the corresponding COH models, rather than to serve as a fully functional implementation.

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 January 2026 doi:10.20944/preprints202601.1685.v2


```
        self.constraint_system.register_reasoner("resource", ResourceReasoner())

        # Add NLP for reporting and queries
        self.dialogue_manager = COHDialogueManager()
        self.hospital.add_neural_component("nlp_interface", self.dialogue_manager)

    def run_system(self):
        # Start all monitoring daemons
        self.hospital.start_daemons()

        # Continuous constraint validation loop
        while True:
            context = self.collect_current_state()
            violations = self.constraint_system.validate_all(context)

            for violation in violations:
                if violation.severity > 7:
                    self.trigger_emergency_response(violation)
                else:
                    self.attempt_constraint_resolution(violation)

            # Update neural components with new data
            self.update_neural_models(context)

            # Process natural language queries
            nlp_queries = self.get_nlp_queries()
            for query in nlp_queries:
                response = self.dialogue_manager.respond(query)
                self.deliver_response(response)
```

This implementation demonstrates how COH models become live systems: hierarchical structures manage state, neural components provide adaptive predictions, constraint systems maintain system integrity, and NLP interfaces enable human interaction. The system operates autonomously while remaining responsive to external inputs and environmental changes.

6. COH as a Universal World Model

The COH framework demonstrates universality as a world model through its consistent application across diverse domains while maintaining domain-specific integrity. This universality emerges from several key characteristics: hierarchical decomposition as a fundamental organizing principle, constraint-based reasoning as a universal inference mechanism, and the integration of neural and symbolic processing within a unified representation.

Formally, any system S can be represented as COH 9-tuple O = (C, A, M, N, E, I, T, G, D) where:

- C captures the compositional structure through hierarchical decomposition
- A represents state variables across abstraction levels
- M encapsulates executable transformations
- N provides adaptive, learning-based capabilities
- E enables semantic understanding and cross-modal integration
- I encodes fundamental, inviolable constraints
- T implements reactive, event-driven behaviors
- G specifies optimization objectives
- D provides continuous monitoring and maintenance

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 January 2026 doi:10.20944/preprints202601.1685.v2

This representation proves universal because (1) all systems or worlds exhibit hierarchical organization, either physically, organizationally, or conceptually; (2) all systems operate under constraints, whether physical laws, social norms, or computational limits; (3) all systems combine deterministic and adaptive behaviors; and (4) all systems require monitoring to maintain integrity.

```
# Universal COH template for any system
class UniversalCOHModel(COHObject):
    def __init__(self, system_name):
        super().__init__(system_name)

        # Universal pattern: hierarchical decomposition
        self.components = []    # C: Sub-systems or elements

        # Universal pattern: state representation
        self.attributes = {}    # A: State variables

        # Universal pattern: executable behaviors
        self.methods = {}       # M: Transformations and actions

        # Universal pattern: adaptive capabilities
        self.neural_components = {}    # N: Learning and prediction

        # Universal pattern: semantic understanding
        self.embedding_model = None    # E: Cross-modal integration

        # Universal pattern: fundamental constraints
        self.identity_constraints = []    # I: Inviolable rules

        # Universal pattern: reactive behaviors
        self.trigger_constraints = []     # T: Event-response rules

        # Universal pattern: optimization objectives
        self.goal_constraints = []        # G: Optimization targets

        # Universal pattern: continuous monitoring
        self.daemons = {}                 # D: Real-time validators

    def validate_universal_properties(self):
        """All systems must satisfy these universal checks"""
        # 1. Hierarchy consistency
        assert self._validate_hierarchy_acyclic(), "Hierarchy must be acyclic"

        # 2. Constraint coherence
        assert self._check_constraint_consistency(), "Constraints must be consistent"

        # 3. State evolution legality
        assert self._validate_state_transitions(), "State changes must respect constraints"

        # 4. Adaptive learning safety
        assert self._validate_neural_adaptation(), "Learning must respect constraints"

        return True
```

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 January 2026 doi:10.20944/preprints202601.1685.v2

The universality of COH enables modelling of systems ranging from microscopic biological processes to macroscopic social organizations, from deterministic physical systems to probabilistic economic markets, from concrete mechanical devices to abstract computational processes. This cross-domain consistency provides the foundation for general intelligence that operates coherently across domains.

7 Integrating Symbolic Knowledge with Neural Computation

COH/GISMOL achieves seamless symbolic-neural integration through architectural principles that embed neural components within the symbolic hierarchy rather than treating them as separate subsystems. This integration occurs at three levels: representation (neural embeddings of symbolic structures), inference (neural enhancement of symbolic reasoning), and learning (constraint-guided neural adaptation).

At the representation level, the embedding component E creates vector representations that capture both the structural properties of COHObjects and their semantic meanings. These embeddings enable similarity computation, analogy formation, and cross-modal integration while preserving constraint relationships.

```
# Symbolic-neural integration in GISMOL
from gismol.neural import COHObjectEmbedder, MultiModalEmbedder
from gismol.core import COHObject

class IntegratedSystem(COHObject):
    def __init__(self):
        super().__init__("IntegratedSystem")

        # Symbolic structure
        self.add_component(COHObject("SymbolicComponent"))
        self.add_identity_constraint({
            'name': 'safety_rule',
            'specification': 'temperature < 100'
        })

        # Neural embedding of symbolic structure
        self.embedder = COHObjectEmbedder()
        self.object_embedding = self.embedder.embed_object(self)

        # Multi-modal integration
        self.multi_embedder = MultiModalEmbedder()

    def neural_symbolic_reasoning(self, context):
        """Integrated reasoning combining symbolic and neural approaches"""
        # Symbolic constraint evaluation
        symbolic_result = self.evaluate_constraints(context)

        # Neural prediction
        neural_prediction = self.get_neural_component("predictor").predict(context)

        # Integrated decision making
        if symbolic_result['valid'] and neural_prediction['confidence'] > 0.8:
            return self.execute_action(context)
        elif not symbolic_result['valid']:
            # Neural suggestion for constraint resolution
            suggestion = self.get_neural_component("resolver").suggest_fix(
```

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 January 2026 doi:10.20944/preprints202601.1685.v2


```
                symbolic_result['violations']
            )
            return self.attempt_resolution(suggestion)

    def constraint_guided_learning(self, training_data):
        """Neural training that respects symbolic constraints"""
        # Convert constraints to loss penalties
        constraint_penalties = []
        for constraint in self.identity_constraints:
            penalty_fn = self._constraint_to_loss(constraint)
            constraint_penalties.append(penalty_fn)

        # Train with constraint-aware optimizer
        optimizer = ConstraintAwareOptimizer(
            model_parameters=self.neural_components.values(),
            constraints=constraint_penalties
        )

        # Training loop that respects constraints
        for epoch in range(training_epochs):
            for batch in training_data:
                loss = self.compute_loss(batch)
                constraint_loss = sum(p() for p in constraint_penalties)
                total_loss = loss + constraint_loss

                optimizer.zero_grad()
                total_loss.backward()
                optimizer.step()  # Respects constraints during update
```

This integration provides several advantages: (1) neural components benefit from the structure and constraints provided by the symbolic framework, improving sample efficiency and generalization; (2) symbolic reasoning benefits from neural pattern recognition and prediction capabilities, handling uncertainty and complexity beyond pure symbolic approaches; (3) the system maintains interpretability through the symbolic structure while gaining adaptability through neural components; and (4) learning is guided by constraints, preventing the system from developing behaviors that violate fundamental requirements.

8 Avoiding Jagged Intelligence with COH/GISMOL

"Jagged intelligence" refers to the uneven performance of AI systems across different tasks or domains, where a system might excel at one type of reasoning while failing catastrophically at another superficially similar task [18]. COH/GISMOL addresses this through coherent constraint propagation across the hierarchy, ensuring that reasoning maintains consistency regardless of domain or task specifics.

The framework prevents jagged intelligence through several mechanisms: (1) hierarchical decomposition ensures that knowledge and constraints propagate appropriately across abstraction levels; (2) the unified constraint system maintains consistency across all reasoning processes; (3) neural components are embedded within the constraint hierarchy, preventing them from developing inconsistent behaviors; and (4) the embedding component E provides semantic coherence across different representations.

```
# Preventing jagged intelligence through constraint coherence
from gismol.core import COHObject, ConstraintSystem
from gismol.reasoners import GeneralReasoner, DomainReasoner
```

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 January 2026 doi:10.20944/preprints202601.1685.v2


```
class CoherentIntelligenceSystem(COHObject):
    def __init__(self):
        super().__init__("CoherentSystem")

        # Unified constraint system ensures consistency
        self.constraint_system = ConstraintSystem()

        # Register multiple reasoners that must agree
        self.constraint_system.register_reasoner("general", GeneralReasoner())
        self.constraint_system.register_reasoner("domain", DomainReasoner())
        self.constraint_system.register_reasoner("safety", SafetyReasoner())

        # Consistency check: all reasoners must agree
        self.require_consensus = True

    def coherent_reasoning(self, problem, context):
        """Reasoning that maintains consistency across domains"""
        results = {}

        # Get reasoning from multiple perspectives
        for reasoner_name, reasoner in self.constraint_system.reasoners.items():
            results[reasoner_name] = reasoner.evaluate(problem, context)

        # Check for consistency
        if self.require_consensus:
            unique_results = set(results.values())
            if len(unique_results) > 1:
                # Inconsistency detected - use hierarchy to resolve
                return self.resolve_inconsistency(results, problem, context)

        return results['general']  # Default to general reasoner

    def resolve_inconsistency(self, conflicting_results, problem, context):
        """Resolve reasoning inconsistencies using hierarchical constraints"""
        # Check which result respects higher-level constraints
        for reasoner_name, result in conflicting_results.items():
            constraint_compliance = self.check_constraint_compliance(
                result, problem, context
            )

            if constraint_compliance['all_satisfied']:
                return result  # This result respects all constraints

        # If no result satisfies all constraints, activate learning
        self.trigger_adaptive_learning(problem, context, conflicting_results)
        return self.coherent_reasoning(problem, context)  # Re-evaluate after learning

    def check_constraint_compliance(self, result, problem, context):
        """Check if a reasoning result complies with all relevant constraints"""
        compliance = {'all_satisfied': True, 'violations': []}
```

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 January 2026 doi:10.20944/preprints202601.1685.v2


```
        # Check constraints at multiple hierarchy levels
        current = self
        while current is not None:
            for constraint in current.identity_constraints:
                satisfied = self.evaluate_single_constraint(
                    constraint, result, context
                )
                if not satisfied:
                    compliance['all_satisfied'] = False
                    compliance['violations'].append({
                        'constraint': constraint,
                        'level': current.name
                    })

            # Move up hierarchy
            current = current.parent if hasattr(current, 'parent') else None

        return compliance
```

This approach ensures that reasoning maintains coherence across domains by: (1) requiring multiple reasoning perspectives to agree, (2) using hierarchical constraints to resolve inconsistencies, and (3) triggering adaptive learning when current reasoning approaches fail to satisfy constraints. The result is intelligence that performs consistently across different tasks and domains, avoiding the "jagged" performance characteristic of many current AI systems.

9 Modelling Agentic Systems with COH/GISMOL

Agentic systems—systems that exhibit goal-directed, autonomous behavior—are naturally modelled within the COH framework through the combination of goal constraints (G), trigger constraints (T), and neural components (N). The hierarchical structure enables multi-level agency, where higher-level goals decompose into sub-goals for lower-level components, and neural components provide the adaptive capabilities needed for goal achievement in complex environments.

Formally, an agentic system A can be represented as a COH 9-tuple where: G contains hierarchical goal specifications, T implements goal-directed action selection, N provides learning and adaptation for goal achievement, and D monitors goal progress and adjusts behavior as needed. The compositional hierarchy C enables the decomposition of complex goals into manageable sub-tasks assigned to appropriate components.

```
    # Agentic system implementation in GISMOL
    from gismol.core import COHObject
    from gismol.neural import NeuralSchedulingModel, NeuralResourceMatcher
    from gismol.reasoners import GoalReasoner, TemporalReasoner

    class AgenticSystem(COHObject):
        def __init__(self, name="AutonomousAgent"):
            super().__init__(name)

            # Goal hierarchy
            self.goal_constraints = [
                {
                    'name': 'primary_goal',
                    'specification': 'complete_assignment_within_deadline',
                    'priority': 'HIGH',
                    'subgoals': ['plan_tasks', 'allocate_resources', 'execute_plan']
```

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 January 2026 doi:10.20944/preprints202601.1685.v2


```
            }
        ]

        # Neural components for goal achievement
        self.add_neural_component(
            "task_planner",
            NeuralSchedulingModel(task_dim=64, resource_dim=32, hidden_dim=256)
        )

        self.add_neural_component(
            "resource_allocator",
            NeuralResourceMatcher(capability_list=self.capabilities)
        )

        # Goal reasoner for constraint-aware planning
        self.goal_reasoner = GoalReasoner()

    def pursue_goal(self, goal_specification, context):
        """Autonomous goal pursuit with constraint satisfaction"""
        # Parse goal into constraint format
        goal_constraint = self.parse_goal_to_constraint(goal_specification)

        # Generate plan using neural components
        plan = self.get_neural_component("task_planner").generate(
            goal=goal_constraint,
            context=context
        )

        # Validate plan against all constraints
        validation_result = self.validate_plan(plan)

        if validation_result['valid']:
            # Execute plan with monitoring
            return self.execute_plan(plan, goal_constraint)
        else:
            # Adjust plan to satisfy constraints
            adjusted_plan = self.adjust_plan(plan, validation_result['violations'])
            return self.execute_plan(adjusted_plan, goal_constraint)

    def execute_plan(self, plan, goal_constraint):
        """Execute plan with real-time goal monitoring"""
        # Start goal monitoring daemon
        goal_daemon = self.start_goal_daemon(goal_constraint)

        # Execute plan steps
        for step in plan['steps']:
            # Check if goal still achievable
            if not goal_daemon.is_achievable():
                # Replan with current context
                new_plan = self.replan(goal_constraint, self.get_context())
                return self.execute_plan(new_plan, goal_constraint)
```

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 January 2026 doi:10.20944/preprints202601.1685.v2


```
            # Execute step
            result = self.execute_step(step)

            # Update goal progress
            goal_daemon.update_progress(result)

            # Learn from execution for future improvement
            self.learn_from_execution(step, result)

        return goal_daemon.get_final_result()
```

This implementation demonstrates how COH/GISMOL enables the development of agentic systems that: (1) pursue complex goals through hierarchical decomposition, (2) adapt their strategies based on environmental feedback, (3) maintain constraint satisfaction throughout goal pursuit, and (4) learn from experience to improve future performance. The integration of symbolic goal specifications with neural planning and execution enables agents that are both purposeful and adaptable.

10 Why COH/GISMOL is a Better Framework for AGI

COH/GISMOL advances AGI research by addressing several fundamental challenges that have limited previous approaches. Unlike frameworks that specialize in either symbolic reasoning or neural computation, COH provides unified representation that integrates both paradigms. Unlike systems that excel in narrow domains but fail to generalize, COH provides consistent modelling patterns across domains. Unlike approaches that sacrifice interpretability for capability, COH maintains explainability through explicit constraint representation.

The framework's advantages include: (1) Neuroscientific grounding in hierarchical cortical processing and constraint-based reasoning; (2) Unified representation that integrates structure, behavior, constraints, and learning; (3) Domain generality with consistent application across physical, biological, social, and computational systems; (4) Scalable hierarchy that enables modelling at multiple abstraction levels; (5) Constraint coherence that prevents contradictory behaviors; (6) Practical implementability through the GISMOL toolkit; and (7) Human-AI alignment through explicit constraint representation that can encode ethical and safety requirements.

Comparative analysis reveals that COH/GISMOL addresses limitations of existing approaches: it provides more coherent integration than neuro-symbolic hybrids, more explicit constraint management than deep learning systems, more adaptive capabilities than pure symbolic systems, and more domain generality than specialized frameworks. The result is a framework that supports the development of generally intelligent systems that operate effectively across domains while respecting domain-specific requirements and constraints.

11. Summary of Key Contributions

This paper makes several significant contributions to AGI research:

- Theoretical Foundation: Presents Constrained Object Hierarchies as a neuroscience-grounded theoretical framework for AGI that models intelligence through hierarchical constraint satisfaction.
- Practical Implementation: Introduces GISMOL as a comprehensive implementation of COH, providing executable tools for building intelligent systems.
- Universal World Model: Demonstrates COH's applicability as a universal modelling framework through six detailed case studies across diverse domains.
- Symbolic-Neural Integration: Shows how COH seamlessly integrates symbolic knowledge representation with neural computation within a unified architecture.
- Jagged Intelligence Mitigation: Presents mechanisms for maintaining reasoning coherence across domains, addressing the "jagged intelligence" problem.

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 January 2026 doi:10.20944/preprints202601.1685.v2

- Agentic System Modelling: Illustrates how COH enables the development of autonomous, goal-directed systems with hierarchical goal decomposition.
- Cross-Domain Validation: Provides empirical validation through implementation examples showing consistent application across healthcare, finance, manufacturing, climate science, education, and urban governance.

These contributions advance AGI research by providing both a theoretical framework and practical tools for developing generally intelligent systems that maintain coherence across domains while respecting domain-specific constraints.

12. Conclusion and Future Research

Constrained Object Hierarchies provides a comprehensive framework for modelling generally intelligent systems across diverse domains. Through the GISMOL implementation, we have demonstrated how this framework enables the development of executable systems that maintain constraint coherence while exhibiting adaptive, goal-directed behavior. The six case studies illustrate the framework's versatility and practical applicability.

Future research directions include: (1) extending the neural component library with more specialized architectures for different domain requirements; (2) developing more sophisticated constraint reasoning algorithms that can handle complex, non-linear constraints; (3) creating tools for automatic extraction of COH models from existing systems and documentation; (4) investigating scaling properties for extremely large hierarchies; (5) exploring applications in additional domains such as quantum computing, nanotechnology, and interstellar exploration; and (6) developing formal verification methods for critical constraint satisfaction.

The COH/GISMOL framework represents a significant step toward practical AGI by providing both theoretical foundations and implementable tools. By balancing generality with specificity, structure with adaptability, and symbolic reasoning with neural computation, it addresses fundamental challenges in AI research while providing a path toward generally intelligent systems that can operate effectively across the complex, constrained environments that characterize our world.

References

1. Goertzel, B. The Path to More General Artificial Intelligence. *Journal of Artificial General Intelligence* 2014, *5*, 1–48.
2. Marcus, G. The Next Decade in AI: Four Steps Towards Robust Artificial Intelligence. *arXiv* 2020, arXiv:2002.06177.
3. Wang, H. Constrained Object Hierarchies as a Unified Theoretical Model for Intelligence and Intelligent Systems. Computers 14 (2025): 478. https://doi.org/10.3390/computers14110478
4. Wang, P. On the Working Definition of Intelligence. *Technical Report* 1995, *94*, Indiana University.
5. Russell, S.; Norvig, P. *Artificial Intelligence: A Modern Approach*, 4th ed.; Pearson: Hoboken, NJ, USA, 2020.
6. Lenat, D.B. CYC: A Large-Scale Investment in Knowledge Infrastructure. *Communications of the ACM* 1995, *38*, 33–38.
7. Laird, J.E. The Soar Cognitive Architecture. *MIT Press* 2012, *1*, 1–50.
8. LeCun, Y.; Bengio, Y.; Hinton, G. Deep Learning. *Nature* 2015, *521*, 436–444.
9. Lake, B.M.; Ullman, T.D.; Tenenbaum, J.B.; Gershman, S.J. Building Machines That Learn and Think Like People. *Behavioral and Brain Sciences* 2017, *40*, e253.
10. Bhuyan, B. P.; A. Ramdane-Cherif, R. Tomar, and T. P. Singh, "Neuro-symbolic artificial intelligence: A survey," Neural Comput. Appl., vol. 36, pp. 12809–12844, 2024, doi: 10.1007/s00521-024-09960-z.
11. Hammer, B.; Hitzler, P. *Perspectives of Neural-Symbolic Integration*; Springer: Berlin/Heidelberg, Germany, 2007.

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 January 2026 doi:10.20944/preprints202601.1685.v2

12. Besold, T.R.; Garcez, A.S.; Bader, S.; Bowman, H.; Domingos, P.; Hitzler, P.; Kühnberger, K.U.; Lamb, L.C.; Lowd, D.; Lima, P.M.V.; et al. Neural-Symbolic Learning and Reasoning: A Survey and Interpretation. *arXiv* 2017, arXiv:1711.03902.
13. Goodman, N.D.; Mansinghka, V.K.; Roy, D.M.; Bonawitz, K.; Tenenbaum, J.B. Church: A Language for Generative Models. *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence* 2008, *24*, 220–229.
14. Tenenbaum, J.B.; Kemp, C.; Griffiths, T.L.; Goodman, N.D. How to Grow a Mind: Statistics, Structure, and Abstraction. *Science* 2011, *331*, 1279–1285.
15. Anderson, J.R. *How Can the Human Mind Occur in the Physical Universe?* Oxford University Press: Oxford, UK, 2007.
16. Franklin, S.; Patterson, F.G. The LIDA Architecture: Adding New Modes of Learning to an Intelligent, Autonomous, Software Agent. *Integrated Design and Process Technology* 2006, *2006*, 1–8.
17. Brown, T.B.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. Language Models are Few-Shot Learners. *Advances in Neural Information Processing Systems* 2020, *33*, 1877–1901.
18. Bommasani, R.; Hudson, D.A.; Adeli, E.; Altman, R.; Arora, S.; von Arx, S.; Bernstein, M.S.; Bohg, J.; Bosselut, A.; Brunskill, E.; et al. On the Opportunities and Risks of Foundation Models. *arXiv* 2021, arXiv:2108.07258.
19. Rossi, F.; van Beek, P.; Walsh, T. *Handbook of Constraint Programming*; Elsevier: Amsterdam, The Netherlands, 2006.
20. Marriott, K.; Stuckey, P.J. *Programming with Constraints: An Introduction*; MIT Press: Cambridge, MA, USA, 1998.
21. Hawkins, J.; Ahmad, S. Why Neurons Have Thousands of Synapses, A Theory of Sequence Memory in Neocortex. *Frontiers in Neural Circuits* 2016, *10*, 23.
22. U.S. Dept. Health and Human Services, "Updated national survey trends in telehealth utilization and modality (2021–2022)," Office of the Assistant Sec. for Planning and Evaluation, Washington, DC, USA, Tech. Rep., 2023.
23. Imran, S., T. Mahmood, A. Morshed, and T. Sellis, "Big data analytics in healthcare: A systematic literature review and roadmap," IEEE/CAA J. Automatica Sinica, vol. 8, no. 1, pp. 1–22, 2021, doi: 10.1109/JAS.2020.1003384.
24. Almeida, J. and T. C. Gonçalves, "Cryptocurrency market microstructure: A systematic literature review," Ann. Oper. Res., vol. 332, pp. 1035–1068, 2024, doi: 10.1007/s10479-023-05627-5.
25. Oks, S. J. et al., "Cyber-physical systems in the context of Industry 4.0: A review, categorization and outlook," Inf. Syst. Front., vol. 26, pp. 1731–1772, 2024, doi: 10.1007/s10796-022-10252-x.
26. IPCC, Climate Change 2023: Synthesis Report. Geneva, Switzerland: Intergovernmental Panel on Climate Change, 2023.
27. Pane, J.F.; Steiner, E.D.; Baird, M.D.; Hamilton, L.S. *Continued Progress: Promising Evidence on Personalized Learning*; RAND Corporation: Santa Monica, CA, USA, 2015.
28. Ismagilova, E.; L. Hughes, N. P. Rana, and Y. K. Dwivedi, "Security, privacy and risks within smart cities: A literature review and development of a smart city interaction framework," Inf. Syst. Front., vol. 24, pp. 393–414, 2022, doi: 10.1007/s10796-020-10044-1.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.