GISMOL: A General Intelligent Systems Modelling Language

Harris Wang
School of Computing and Information Systems, Athabasca University, Athabasca, Canada.

Email: harrisw@athabascau.ca
Manuscript submitted February 9, 2026; accepted March 12, 2026; published May 20, 2026.
doi: 10.18178/JAAI.2026.4.2.104-124

Abstract: This paper presents General Intelligent System Modeling Language (GISMOL), a prototype Python-based framework implementing Constrained Object Hierarchies (COH)—a neuroscience-inspired theoretical framework for Artificial General Intelligence (AGI). GISMOL is currently under active development as a research prototype and has not yet been deployed in production environments at scale. COH and GISMOL together provide a unified language for modelling and implementing intelligent systems across diverse domains including healthcare, manufacturing, finance, and governance. The framework bridges symbolic AI and neural computation through its core architecture of constraint-aware objects with embedded neural components, hierarchical reasoning capabilities, and natural language integration. We demonstrate how GISMOL translates COH's formal 9-tuple representation into executable systems with six comprehensive case studies, showing its versatility in modelling complex intelligent behaviors while maintaining theoretical rigor. The implementation includes specialized modules for neural integration, multi-domain reasoning, and natural language processing, all built around the COHObject abstraction that encapsulates intelligence as constrained hierarchical structures.

Keywords: artificial general intelligence, constrained object hierarchies, neuro-symbolic AI, intelligent systems modelling, constraint programming, hierarchical reasoning, Python framework, constraint reasoning

1. Introduction

The pursuit of Artificial General Intelligence (AGI) represents one of the most ambitious goals in computer science and cognitive science [1]. Unlike narrow AI systems optimized for specific tasks, AGI aims to achieve human-level versatility, adaptability, and reasoning across multiple domains [2]. Current approaches to AGI development face fundamental challenges, including the integration of neural learning with symbolic reasoning, the maintenance of safety and ethical constraints, and the creation of systems that can generalize beyond their training distributions [3].

Traditional deep learning approaches, while powerful for pattern recognition, struggle with explicit reasoning, constraint satisfaction, and interpretability [4]. Conversely, symbolic AI systems excel at logical inference and knowledge representation but lack the learning capabilities and robustness to uncertainty of neural approaches [5]. This dichotomy has led to renewed interest in neuro-symbolic integration, which seeks to combine the strengths of both paradigms [6].

GISMOL addresses these challenges through its foundation in Constrained Object Hierarchies (COH), a theoretical framework that models intelligent systems as hierarchical compositions of objects subject to

multi-domain constraints [7]. COH extends beyond typical neuro-symbolic approaches by embedding constraints directly within the system architecture through identity, trigger, and goal constraints monitored by autonomous daemons [8].

Scope and contributions of this paper:

1. A formal presentation of the COH theoretical framework and its neuroscience grounding.

2. A description of the design goals and initial Application Programming Interface (API) of General Intelligent System Modeling Language (GISMOL), a prototype Python library for COH.

3. Six conceptual case studies and code snippets illustrating intended usage patterns and cross-domain applicability.

4. A preliminary comparison with existing approaches to indicate potential advantages and gaps.

5. A development roadmap, implementation strategies, and evaluation plan suitable for future empirical validation.

Disclaimer. GISMOL is not yet a mature production framework. The API is subject to change, and examples provided are prototypes or simulated demonstrations, not deployed systems.

2. Literature Review

2.1. Symbolic AI and Knowledge Representation

Early AI research focused heavily on symbolic approaches, with systems like SOAR [9] and ACT-R [10] providing cognitive architectures for representing knowledge and reasoning. These systems excelled at tasks requiring explicit rule-based reasoning but struggled with perception, learning, and handling uncertainty [11]. More recent approaches like TensorLog [12] have attempted to bridge symbolic and neural representations through differentiable inference but often lack comprehensive constraint management capabilities.

2.2. Neural Networks and Deep Learning

The resurgence of neural networks, particularly through deep learning architectures [13], has revolutionized fields like computer vision, natural language processing, and reinforcement learning [14]. However, these systems typically operate as “black boxes” with limited interpretability and difficulty incorporating explicit constraints or symbolic knowledge [15].

2.3. Neuro-Symbolic Integration

Recent work in neuro-symbolic AI has explored various integration strategies. DeepProbLog [16] combines probabilistic logic programming with neural networks, while Neural Theorem Provers [17] use differentiable reasoning. Neurosymbolic Concept Learners [18] integrate perception with reasoning. However, these approaches often lack the hierarchical organization and comprehensive constraint system provided by COH.

2.4. Cognitive Architectures and AGI

Cognitive architectures like LIDA [19] and CLARION [20] attempt to model human cognition more comprehensively but often lack practical implementation frameworks for real-world applications. GISMOL builds on these ideas while providing a concrete, executable modeling language.

2.5. Constraint Satisfaction and Optimization

Constraint programming [21] and Constraint Satisfaction Problems (CSPs) [22] provide formalisms for representing and solving constrained systems. In traditional CSPs, a problem is defined by variables with domains and constraints that must be satisfied, with solutions found through systematic search or

consistency algorithms. GISMOL extends these classical concepts in several significant ways.

First, COH's goal constraints correspond to the optimization objectives found in Constraint Optimization Problems (COPs), where solutions are evaluated not just on satisfaction but on quality metrics. However, where traditional COPs typically optimize a single objective function, COH goal constraints support multiple competing objectives (e.g., maximize throughput while minimizing energy consumption) that are continuously monitored and balanced during system operation.

Second, COH introduces hierarchical constraint satisfaction, where constraints at different levels of the object hierarchy interact and propagate. A constraint at a parent object (e.g., "factory throughput ≥ 1000 units/hour") must be decomposed into constraints on child components (e.g., individual production cells) while maintaining global consistency. This mirrors hierarchical constraint logic programming but with dynamic adaptation.

Third, COH's constraint system integrates with neural components to enable predictive constraint satisfaction. Rather than simply checking whether constraints are currently satisfied, neural models predict future constraint violations (e.g., predicting that throughput will fall below threshold in 30 min) and trigger proactive resolution. This moves beyond classical constraint satisfaction into constraint forecasting and prevention.

Finally, the constraint daemons provide continuous runtime verification [23] of constraints, extending the static verification typical of constraint programming into dynamic, real-time monitoring with autonomous resolution capabilities.

3. Introducing Constrained Object Hierarchies (COH)

3.1. Formal Definition

Constrained Object Hierarchies (COH) provides a formal framework for modeling general intelligent systems as 9-tuple structures:

$$O = (C, A, M, N, E, I, T, G, D)$$

where:

- C (Components): Sub-objects forming a compositional hierarchy, enabling complex system decomposition.
- A (Attributes): State variables describing object properties, representing system state.
- M (Methods): Executable actions or behaviors, implementing system functionality.
- N (Neural Components): Adaptive models for learning and inference, providing statistical learning capabilities.
- E (Embedding): Neural components for semantic representation, enabling knowledge integration.
- I (Identity Constraints): Fundamental structural and logical rules, defining system invariants.
- T (Trigger Constraints): Event-condition-action mechanisms, enabling reactive behavior.
- G (Goal Constraints): Optimization objectives guiding intelligent behavior, providing purposeful direction.
- D (Constraint Daemons): Real-time monitors enforcing constraints, ensuring continuous compliance.

3.2. Neuroscience Grounding

COH draws explicit inspiration from hierarchical and regulatory mechanisms in biological neural systems [24]. The framework's design maps specific neuroscientific findings to computational constructs:

Hierarchical cortical organization—The mammalian neocortex is organized hierarchically, with primary sensory areas feeding into association areas that integrate information across modalities, and prefrontal

cortex implementing executive control [24]. In COH, this inspired the component hierarchy (C) where objects can contain sub-objects, enabling compositional decomposition while maintaining global coherence. Just as visual cortex processes edges before shapes before objects, COH objects process information through their component hierarchy.

Prefrontal cortex and executive function—The prefrontal cortex implements working memory, rule-based behavior, and goal maintenance [25]. This directly inspired COH's Identity constraints (I) and Goal constraints (*G*). The prefrontal cortex maintains abstract rules that constrain behavior—analogous to identity constraints that define object invariants. Similarly, it maintains goals that guide action selection—directly parallel to goal constraints that specify optimization objectives.

Basal ganglia and reinforcement learning—The basal ganglia mediate action selection based on reward predictions, implementing a form of reinforcement learning [25]. This inspired the integration of Neural (*N*) components with goal constraints. In COH, neural components learn to predict outcomes, while goal constraints specify the objectives that guide learning—mirroring how basal ganglia use reward signals to shape behavior.

Hippocampus and relational memory—The hippocampus supports episodic memory and relational reasoning, binding together disparate pieces of information into coherent representations [24]. This inspired the Embedding (*E*) component in COH, which creates semantic representations that relate objects based on their properties and relationships, analogous to hippocampal pattern completion and relational binding.

Homeostatic regulation—Biological systems maintain stability through homeostatic mechanisms that continuously monitor internal state and trigger corrective actions when parameters deviate from set points [26]. This directly inspired constraint Daemons (*D*) in COH—autonomous monitoring processes that continuously evaluate constraints and initiate resolution when violations occur, exactly as the body maintains temperature, pH, and glucose levels.

Event-condition-action in basal ganglia—The basal ganglia implement gating mechanisms that trigger action execution when specific conditions are met [25]. This maps directly to Trigger (*T*) constraints in COH, which specify event-condition-action rules for reactive behavior. When a particular event occurs (e.g., obstacle detected) and conditions are satisfied, the trigger executes its associated action.

Neuromodulation and adaptive constraint weighting—Neuromodulators like dopamine and serotonin adjust the relative importance of different behavioral constraints based on context [26]. This inspired COH's adaptive constraint prioritization, where constraint severities and goal weights can be dynamically adjusted based on system state. Table 1 summarizes the correspondence between key neuroscience concepts and the formal elements of the COH framework.

Table 1. Neuroscience Concepts to COH Mapping

Neuroscience Concept	Brain Region/Mechanism	COH Element	Function
Hierarchical processing	Cortical hierarchy	*C* (Components)	Compositional decomposition
Executive control	Prefrontal cortex	*I, G* (Identity, Goal)	Rule maintenance, goal pursuit
Reward-based learning	Basal ganglia	*N* (Neural)	Adaptive prediction
Relational memory	Hippocampus	*E* (Embedding)	Semantic representation
Homeostasis	Regulatory systems	*D* (Daemons)	Continuous monitoring
Action gating	Basal ganglia	*T* (Trigger)	Event-condition-action
Context modulation	Neuromodulation	Constraint weighting	Adaptive prioritization

This neuroscience grounding ensures that COH is not merely an abstract formalism but incorporates principles that have evolved in biological systems to produce robust, adaptive intelligence.

4. Structure of GISMOL

Development status. GISMOL is a prototype library; several modules are partially implemented, and others are in design or stubbed form. APIs described below represent the intended interface and may evolve.

GISMOL implements COH theory as a Python library with four main modules. Fig. 1 illustrates the module architecture and dependencies.

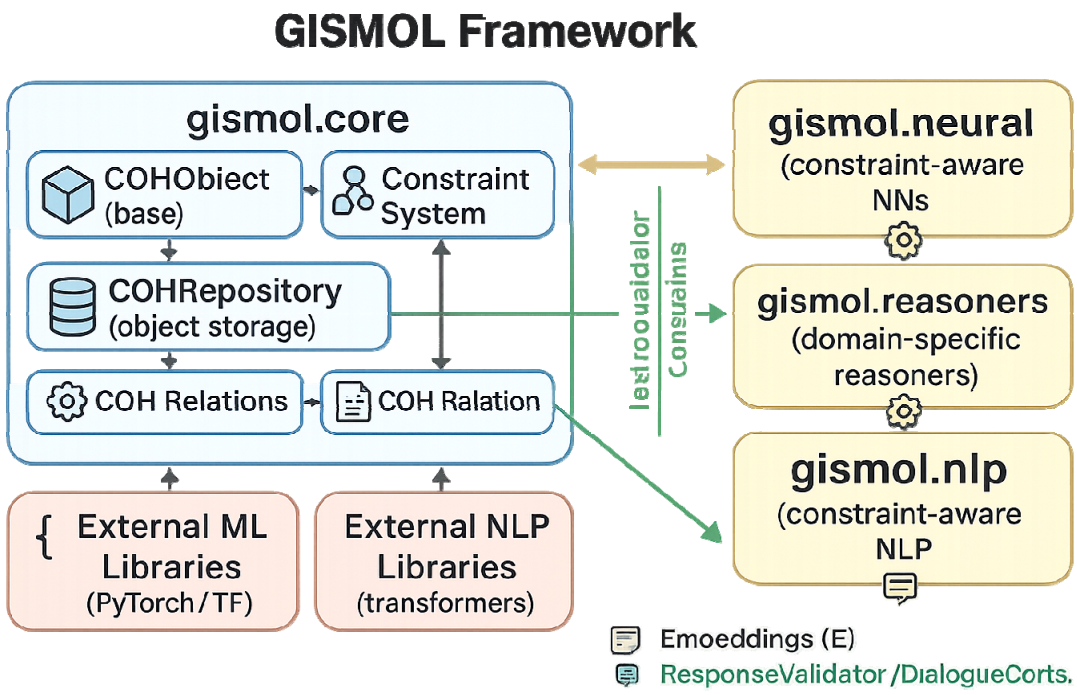

Fig. 1. GISMOL module architecture and dependencies.

4.1. Core Module (gismol.core)

The Core Module establishes the foundational COHObject class and its supporting infrastructure. Current prototypes include:

- COHObject: Base class for intelligent system components.
- ConstraintSystem: Manages constraint evaluation and resolution.
- ConstraintDaemon: Autonomous agents for real-time constraint monitoring.
- COHRepository: Manages object collections and relationships.

4.2. Neural Module (gismol.neural)

The Neural Module designs constraint-aware neural components that integrate with COH objects:

- NeuralComponent: Base class combining nn.Module with COHObject.
- EmbeddingModel: Generates semantic representations of objects and text.
- ConstraintAwareOptimizer: Optimizers that respect constraints.
- Specialized models for classification, regression, and generation (in progress).

4.3. Reasoners Module (gismol.reasoners)

Provides domain-specific engines for constraint evaluation:

- BaseReasoner: Fallback implementation with robust error handling.
- Domain-specific reasoners (Biological, Physical, Geometric, etc.).
- Advanced reasoning systems (Causal, Probabilistic, Temporal, etc.).
- Registry pattern for dynamic reasoner discovery.

4.4. NLP Module (gismol.nlp)

Enables natural language understanding with constraint awareness:

- COHDialogueManager: Manages conversations with constraint validation (prototype).
- EntityRelationExtractor: Extracts knowledge from text into COHObjects (prototype).
- Text2COH: Converts documents into hierarchical object structures (planned).
- ResponseValidator: Ensures language generation respects constraints (planned).

4.5. COH-to-GISMOL Mapping

Table 2 shows the mapping between COH theoretical elements and GISMOL implementation constructs (design intent).

Table 2. COH Elements to GISMOL Implementation Mapping

COH Element	GISMOL Implementation	Primary Module
C (Components)	COHObject hierarchy, COHRepository	gismol.core
A (Attributes)	COHObject.attributes dictionary	gismol.core
M (Methods)	COHObject.methods dictionary	gismol.core
N (Neural Components)	NeuralComponent subclasses	gismol.neural
E (Embedding)	EmbeddingModel classes	gismol.neural
I (Identity Constraints)	Constraint objects with identity category	gismol.core
T (Trigger Constraints)	Constraint objects with trigger type	gismol.core
G (Goal Constraints)	Constraint objects with goal type	gismol.core
D (Constraint Daemons)	ConstraintDaemon instances	gismol.core

5. Constructs of GISMOL

5.1. Classes in the Core

COHObject class: Fundamental Intelligent Unit

The COHObject class implements the core abstraction of intelligent entities in GISMOL:

```
class COHObject:
    """Fundamental unit of intelligence in COH framework"""
    def __init__(self, name):
        self.name = name
        self.identity_constraints = []
        self.trigger_constraints = []
        self.goal_constraints = []
        self.neural_components = {}
        self.daemons = {}
        self.relations = COHRelation()
        self._embedding_model = None
    def add_identity_constraint(self, constraint_spec):
        """Add fundamental structural constraint"""
        constraint = IdentityConstraint.from_spec(constraint_spec)
        self.identity_constraints.append(constraint)
```

```
    def add_neural_component(self, name, component, is_embedding_model=False):
        """Add adaptive neural component"""
        if is_embedding_model:
            self._embedding_model = component
        else:
            self.neural_components[name] = component
    def semantic_distance(self, other):
        """Calculate semantic distance to another object"""
        if self._embedding_model:
            return self._embedding_model.distance(self, other)
        return float('inf')
```

Each COHObject encapsulates a complete intelligent entity with constraints, neural capabilities, and relational context.

COHRepository Class

The COHRepository class manages collections of COHObjects with semantic search:

```
class COHRepository:
    """Repository for COHObjects with semantic operations"""
    def find_semantic_matches(self, query, threshold=0.7):
        """Find objects semantically similar to query"""
        query_embedding = self.embedder.embed_text(query)
        matches = []
        for obj in self.objects.values():
            obj_embedding = obj.get_embedding()
            similarity = cosine_similarity(query_embedding, obj_embedding)
            if similarity >= threshold:
                matches.append((obj, similarity))
        return sorted(matches, key=lambda x: x[1], reverse=True)
```

ConstraintSystem Class

The ConstraintSystem class coordinates constraint evaluation across multiple reasoners:

```
class ConstraintSystem:
    """Orchestrates constraint evaluation with multiple reasoners"""
    def validate_all(self, context):
        """Validate all constraints with appropriate reasoners"""
        results = {}
        for constraint in self.constraints:
            reasoner_type = self._determine_reasoner_type(constraint)
            reasoner = Reasoner.get_reasoner(reasoner_type)()
            results[constraint.name] = reasoner.evaluate(constraint, context)
        return results
```

5.2. Neural Module Classes

NeuralComponent Class

The NeuralComponent class provides the base for constraint-aware neural models:

```
class NeuralComponent(COHObject, nn.Module):
    """Base class for neural components with constraint awareness"""
    def train_component(self, dataset, constraints=None):
```

```
"""Train with constraint penalty terms"""
if constraints:
    loss_fn = self._build_constrained_loss(constraints)
else:
    loss_fn = nn.CrossEntropyLoss()
# Training loop with constraint monitoring
for epoch in range(epochs):
    for batch in dataset:
        loss = self._train_step(batch, loss_fn)
        self._check_training_constraints(loss)
```

5.3. Reasoners Module Classes

Reasoner Class

The Reasoner class is an abstract base for constraint reasoning and resolution:

```
class Reasoner(ABC):
    """Abstract base class for constraint reasoners"""
    _registry = {}    # Global registry of reasoner types
    @classmethod
    def get_reasoner(cls, reasoner_type):
        """Factory method for getting reasoner instances"""
        return cls._registry.get(reasoner_type, BaseReasoner)
    @abstractmethod
    def evaluate(self, constraint, context):
        """Evaluate constraint in given context"""
        pass
```

BiologicalReasoner Class

The BiologicalReasoner handles biomedical constraints:

```
class BiologicalReasoner(BaseReasoner, reasoner_type="biological"):
    """Reasoner for biological and medical constraints"""
    def evaluate(self, constraint, context):
        if "concentration" in constraint.specification:
            return self._evaluate_concentration(constraint, context)
        elif "growth_rate" in constraint.specification:
            return self._evaluate_growth(constraint, context)
        return super().evaluate(constraint, context)
```

5.4. NLP Module Classes

COHDialogueManager Class

The COHDialogueManager class enables constraint-aware conversation:

```
class COHDialogueManager:
    """Manages dialogues with constraint validation"""
    def respond(self, user_input):
        """Generate constraint-aware response"""
        # Recognize intent
        intent = self.intent_recognizer.recognize_intent(user_input)
        # Generate candidate response
        candidate = self.response_generator.generate(intent)
```

```
        # Validate against constraints
        validation = self.response_validator.validate(
            candidate,
            self.repository.objects
        )
        if validation['valid']:
            return candidate
        else:
            # Generate alternative that satisfies constraints
            return self._generate_constraint_compliant_alternative(
                candidate,
                validation['violations']
            )
```

6. Programming Intelligent Systems with GISMOL

We demonstrate GISMOL programming through a healthcare system: the Autonomous Diagnostic Assistant. The code examples focus on the COH 9-tuple structure rather than implementation details.

6.1. Core COH Structure

```
class AutonomousDiagnosticAssistant(COHObject):
"""Healthcare diagnostic system with COH implementation"""
    def __init__(self, name="DiagnosticAssistant"):
        super().__init__(name)
        # 1. Components - hierarchical decomposition
        self.components = {
            'symptom_analyzer': SymptomAnalyzer(),
            'disease_knowledge_base': DiseaseKnowledgeBase(),
'treatment_recommender': TreatmentRecommender()
        }
        # 2. Attributes - state variables
        self.attributes = {
            'diagnostic_confidence': 0.0,
            'current_patient': None,
'active_differential_diagnosis': []
        }
# 3. Neural Components
        self.add_neural_component(
            "diagnostic_classifier",
BayesianClassifier(input_dim=128, output_dim=2000)
        )
# 4. Embedding Model
        self.add_neural_component(
            "medical_embedder",
MedicalConceptEmbedder(embedding_dim=256),
            is_embedding_model=True
        )
```

```
# 5. Identity Constraints
        self.add_identity_constraint({
            'name': 'diagnostic_accuracy',
            'specification': 'diagnostic_confidence >= 0.7 or escalate_to_human',
'severity': 9
        })
        self.add_identity_constraint({
            'name': 'patient_privacy',
'specification': 'patient_data.encrypted == True',
'severity': 10
        })
# 6. Trigger Constraints
        self.add_trigger_constraint({
            'name': 'critical_finding',
'event': 'symptom_severity > 8',
            'action': 'alert_emergency_team',
            'critical': True
        })
        self.add_trigger_constraint({
            'name': 'abnormal_result',
            'event': 'lab_result.out_of_range == True',
'action': 'schedule_followup'
        })
# 7. Goal Constraints
        self.add_goal_constraint({
            'name': 'maximize_accuracy',
            'objective': 'MAXIMIZE diagnostic_accuracy',
'subject_to': ['false_positive_rate < 0.05', 'decision_time < 300 seconds']
        })
        self.add_goal_constraint({
            'name': 'minimize_testing',
            'objective': 'MINIMIZE unnecessary_tests',
'subject_to': ['diagnostic_confidence >= 0.7']
        })
# 8. Constraint Daemons
        self.daemons['patient_safety'] = MedicalSafetyDaemon(
monitoring_interval=0.5,
vital_sign_monitors=['heart_rate', 'blood_pressure']
        )
        self.daemons['diagnostic_quality'] = QualityAssuranceDaemon(
evaluation_interval=60.0,
quality_metrics=['accuracy', 'timeliness']
        )
# Start daemons
for daemon in self.daemons.values():
```

```
        daemon.start()
```

6.2. Core Diagnostic Method

```
def diagnose_patient(self, patient_data, symptoms):
"""Main diagnostic method with constraint validation"""
# Pre-condition check
if not self._check_diagnostic_preconditions(patient_data):
return {'success': False, 'error': 'Preconditions not met'}
# Symptom analysis using neural component
    symptom_embedding = self.get_neural_component(
'diagnostic_classifier'
    ).encode(symptoms)
# Query knowledge base
    similar_cases = self.components['disease_knowledge_base'].query(
        symptom_embedding,
        similarity_threshold=0.7
    )
# Generate diagnosis
    differential = self._generate_differential_diagnosis(
symptoms, similar_cases
    )
# Validate against medical constraints
    validation = self._validate_diagnosis(differential)
    if not validation['valid']:
# Attempt resolution or escalate
return self._handle_diagnostic_failure(differential, validation)
# Update state
    self.attributes['active_differential_diagnosis'] = differential
self.attributes['diagnostic_confidence'] = self._calculate_confidence(differential)
    return {
        'success': True,
        'diagnosis': differential,
'confidence': self.attributes['diagnostic_confidence']
    }
```

In the code snippet above symptom analysis using neural component and semantic embedding models allows the system to identify diagnostically similar cases and generate candidate differentials, an approach aligned with recent large-scale applications of deep learning to electronic health records for clinical prediction and decision support [27]

6.3. System Usage

```
# Instantiate and use the diagnostic assistant
diagnostic_assistant = AutonomousDiagnosticAssistant(
    name="HospitalDiagnosticAI"
)
# Process a patient case
patient_data = {
```

```
        'id': 'PAT_789',
        'age': 45,
    'symptoms': ['chest_pain', 'shortness_of_breath']
    }
    result = diagnostic_assistant.diagnose_patient(
        patient_data=patient_data,
        symptoms=patient_data['symptoms']
    )
    if result['success']:
        print(f"Diagnosis: {result['diagnosis']}")
        print(f"Confidence: {result['confidence']:.2f}")
        # Check constraint satisfaction
        validation = diagnostic_assistant.validate_current_state()
        if validation['all_satisfied']:
            print("All constraints satisfied")
```

This implementation demonstrates how GISMOL enables programming of complex intelligent systems with built-in constraint management, neural integration, and hierarchical organization. The Autonomous Diagnostic Assistant showcases all elements of the COH 9-tuple in practice, with the code focused on structural relationships rather than implementation details. The inclusion of critical identity constraints and patient-safety daemons is particularly important in healthcare contexts, where preventable medical errors remain a leading cause of mortality and necessitate continuous monitoring and escalation mechanisms [28].

7. Modeling and Programming Examples

The following examples illustrate how various intelligent systems can be modelled with COH and programmed with GISMOL. Due to space limitations, we present only the essential COH structure for each example.

7.1. Smart Manufacturing: Adaptive Production Orchestrator

System Description: Modern manufacturing requires real-time adaptation to disruptions while maintaining efficiency and quality, a challenge increasingly addressed using deep learning-based methods for intelligent scheduling, quality control, and predictive optimization in smart manufacturing systems [29].

COH-based Design:

```
class AdaptiveProductionOrchestrator(COHObject):
    def __init__(self, name="ProductionOrchestrator"):
        super().__init__(name)
# Components
        self.components = {
            'scheduler': NeuralSchedulingModel(),
            'quality_controller': QualityController(),
            'resource_manager': ResourceManager()
        }
# Neural Components
        self.add_neural_component(
            "dynamic_scheduler",
NeuralSchedulingModel(task_dim=64, resource_dim=32)
```

```
        )
# Identity Constraints
        self.add_identity_constraint({
            'name': 'throughput_target',
            'specification': 'units_per_hour >= 1000',
            'reasoner': 'resource'
        })
        self.add_identity_constraint({
'name': 'quality_standard',
'specification': 'defect_rate < 0.01',
'reasoner': 'statistical'
        })

# Trigger Constraints
        self.add_trigger_constraint({
            'name': 'machine_failure',
            'event': 'machine_status == "failed"',
            'action': 'reallocate_tasks',
            'postcondition': 'throughput_loss < 5%'
        })
# Goal Constraints
        self.add_goal_constraint({
'name': 'optimize_production',
'objective': 'MAXIMIZE throughput, MINIMIZE energy',
'subject_to': ['quality_standard', 'safety_protocols']
        })
```

Key Insight: Quality deviations trigger rescheduling while maintaining throughput goals. Neural predictions inform constraint adjustments.

7.2. Social Science: Policy Impact Simulator

System Description: Social policy analysis requires modeling complex interactions between demographic groups and institutions.

COH-based Design:

```
class PolicyImpactSimulator(COHObject):
def __init__(self, name="PolicySimulator"):
        super().__init__(name)
# Components (demographic groups as objects)
        self.components = {
'population_groups': self._create_demographic_groups(),
'institutions': self._create_institutions(),
'social_networks': SocialNetworkModel()
        }
# Neural Components
        self.add_neural_component(
"agent_model",
GraphNeuralNetwork(hidden_dim=128)
```

```
            )
    # Identity Constraints
            self.add_identity_constraint({
    'name': 'population_conservation',
    'specification': 'total_population == constant',
    'reasoner': 'mathematical'
            })
            self.add_identity_constraint({
    'name': 'budget_constraint',
    'specification': 'total_cost <= policy_budget',
                'reasoner': 'resource'
            })
    # Trigger Constraints
            self.add_trigger_constraint({
    'name': 'inequality_alert',
    'event': 'gini_coefficient > 0.45',
    'action': 'trigger_policy_review'
            })
    # Goal Constraints
            self.add_goal_constraint({
    'name': 'maximize_welfare',
    'objective': 'MAXIMIZE social_welfare_index',
    'subject_to': ['budget_constraint', 'political_feasibility']
            })
        def simulate_policy(self, policy, steps=100):
    """Run policy simulation"""
            # Apply policy as constraint modifications
            modified_constraints = self._policy_to_constraints(policy)
    # Run simulation
            for step in range(steps):
    # Agent decisions using neural model
    decisions = self.get_neural_component('agent_model').predict()
                # Update with constraint satisfaction
    self._update_with_constraints(decisions)
            return self._collect_outcomes()
```

Key Insight: Graph neural networks model social network effects while constraints ensure conservation laws and budget limits.

7.3. Psychology: Personalized Mental Health Companion

System Description: Mental health support requires personalized interventions that maintain therapeutic boundaries.

COH-based Design:

```
class MentalHealthCcompanion(COHObject):
    def __init__(self, client_profile):
        super().__init__(name=f"Companion_{client_profile['id']}")
# Components
```

```
        self.components = {
'mood_tracker': MoodTracker(),
'coping_strategies': CopingStrategyLibrary(),
'crisis_detector': CrisisDetector()
        }
# Neural Components
        self.add_neural_component(
"emotion_recognizer",
EmotionRecognitionModel()
        )
# Identity Constraints (ethical boundaries)
        self.add_identity_constraint({
            'name': 'client_autonomy',
'specification': 'never_override_client_choice',
            'critical': True,
'reasoner': 'ethical'
        })
        self.add_identity_constraint({
'name': 'therapeutic_boundary',
'specification': 'maintain_professional_relationship',
            'critical': True
        })
# Trigger Constraints
        self.add_trigger_constraint({
            'name': 'crisis_alert',
            'event': 'suicidal_ideation_detected',
            'action': 'escalate_to_human',
'postcondition': 'human_contact_established within 15min',
            'critical': True
        })
        self.add_trigger_constraint({
'name': 'progress_stagnation',
'event': 'no_improvement for 30 days',
'action': 'suggest_therapy_adjustment'
        })
# Goal Constraints
        self.add_goal_constraint({
'name': 'improve_wellbeing',
'objective': 'MAXIMIZE wellbeing_score',
'subject_to': ['client_autonomy', 'evidence_based_practices']
        })
```

Key Insight: Therapy is decomposed into techniques with efficacy constraints. Ethical boundaries are enforced as critical identity constraints.

7.4. Finance: Adaptive Portfolio Management System

System Description: Financial markets require balancing risk and return across multiple time horizons.

COH-based Design:

```
class AdaptivePortfolioManager(COHObject):
def __init__(self, name="PortfolioManager"):
        super().__init__(name)
# Components
        self.components = {
'asset_allocator': AssetAllocator(),
'risk_manager': RiskManager(),
'execution_engine': ExecutionEngine()
        }
# Neural Components
        self.add_neural_component(
"regime_detector",
MarketRegimeDetector(hidden_dim=64)
        )
# Identity Constraints
        self.add_identity_constraint({
'name': 'diversification',
'specification': 'max_position_size <= 0.05 of portfolio',
'reasoner': 'financial'
        })
        self.add_identity_constraint({
'name': 'regulatory_compliance',
'specification': 'sector_exposure within_limits',
'reasoner': 'regulatory'
        })
# Trigger Constraints
        self.add_trigger_constraint({
'name': 'risk_breach',
'event': 'VaR > limit',
'action': 'reduce_risk_exposure'
        })
# Goal Constraints
        self.add_goal_constraint({
'name': 'optimize_returns',
'objective': 'MAXIMIZE sharpe_ratio',
'subject_to': ['diversification', 'max_drawdown < 0.15']
        })
    def rebalance_portfolio(self, market_data):
"""Constraint-aware rebalancing"""
# Predict regime
regime = self.get_neural_component('regime_detector').predict(market_data)
        # Adjust constraints based on regime
        self._adjust_risk_constraints(regime)
        # Optimize with constraint-aware optimizer
```

```
        optimizer = ConstraintAwareOptimizer(
constraints=self._get_all_constraints()
        )
        return self._execute_rebalancing(optimizer.solution)
```

Key Insight: Attention mechanisms identify cross-asset relationships while risk limits propagate through portfolio hierarchy.

7.5. Governance: Participatory Policy Co-creation Platform

System Description: Democratic governance requires integrating diverse perspectives while maintaining procedural fairness.

COH-based Design:

```
class PolicyCoCreationPlatform(COHObject):
def __init__(self, name="PolicyPlatform"):
        super().__init__(name)
# Components (stakeholder groups)
        self.components = {
'citizens': CitizenGroup(),
'experts': ExpertPanel(),
'government': GovernmentAgency()
        }
# Neural Components
        self.add_neural_component(
"argument_analyzer",
ArgumentMiningModel()
        )
        self.add_neural_component(
"consensus_detector",
ConsensusDetectionModel()
        )
# Identity Constraints (democratic principles)
        self.add_identity_constraint({
'name': 'inclusion',
'specification': 'all_stakeholders_represented',
'reasoner': 'democratic'
        })
        self.add_identity_constraint({
'name': 'transparency',
'specification': 'all_decisions_logged_and_auditable',
'reasoner': 'governance'
        })
# Trigger Constraints
        self.add_trigger_constraint({
'name': 'polarization_alert',
'event': 'opinion_divergence > threshold',
'action': 'facilitate_deliberation'
        })
```

```
# Goal Constraints
        self.add_goal_constraint({
'name': 'maximize_legitimacy',
'objective': 'MAXIMIZE stakeholder_satisfaction',
'subject_to': ['inclusion', 'decision_within_timeframe']
        })
def facilitate_deliberation(self, policy_draft):
        """Facilitate deliberation with constraint preservation"""
# Analyze arguments
arguments = self.get_neural_component('argument_analyzer').process()
        # Detect consensus areas
consensus = self.get_neural_component('consensus_detector').detect(arguments)
        # Generate revised policy satisfying constraints
revised = self._revise_policy(policy_draft, consensus)
        # Validate against democratic constraints
        validation = self.daemons['democratic'].validate(revised)
        return revised, validation
```

Key Insight: Multiple competing objectives balanced through Pareto optimization with constraint satisfaction. Deliberation quality monitored by daemons.

8. Summary of COH/GISMOL Advantages

8.1. Comparison with Existing Frameworks

Table 3 provides a preliminary, qualitative comparison with existing AI frameworks.

Table 3. Comparison with Existing AI Frameworks

Framework	Type	Strengths	Limitations	GISMOL (anticipated)
SOAR [9]	Cognitive architecture	Symbolic reasoning, goal-driven	Limited learning, no neural integration	Neuro-symbolic integration, constraint system
ACT-R [10]	Cognitive architecture	Cognitive modeling, production rules	Complex implementation, limited scalability	Python implementation, hierarchical organization
TensorLog [12]	Neuro-symbolic	Differentiable inference, probabilistic	Limited constraint types, no real-time monitoring	Comprehensive constraint system, daemon monitoring
DeepProbLog [16]	Neuro-symbolic	Probabilistic logic, neural networks	No hierarchical constraints, limited domains	Multi-domain constraints, hierarchical organization
PyTorch/TensorFlow	Deep learning	Neural network flexibility, GPU acceleration	No symbolic reasoning, black-box nature	Symbolic constraint integration, explainability
CLIPS [30]	Expert system	Rule-based reasoning, pattern matching	No learning capabilities, static knowledge	Adaptive neural components, continuous learning

8.2. Unique Contributions of COH/GISMOL

Integrated Constraint System: Combines identity, trigger, and goal constraints with neural components.

1. Hierarchical Organization: Natural decomposition of complex systems while maintaining coherence.
2. Multi-domain Reasoning: Unified handling of biological, physical, temporal, and other constraints.
3. Real-time Monitoring: Constraint daemons provide continuous safety guarantees.
4. Practical Implementation (prototype): Python library enables rapid prototyping of intelligent

systems.

5. Cross-domain Applicability: Single framework applicable to healthcare, manufacturing, finance, etc.

8.3. Limitations

1. Learning Curve: Requires understanding of both symbolic and neural approaches.
2. Computational Overhead: Constraint monitoring adds runtime overhead.
3. Domain Knowledge Requirement: System designers must specify appropriate constraints.
4. Early Development Stage: Limited real-world deployment compared to mature frameworks; API subject to change.

9. Conclusion and Future Research Directions

9.1. Summary

GISMOL provides both a theoretical framework (COH) and a prototype Python library for developing intelligent systems. By integrating neural learning with symbolic constraints in a hierarchical organization, GISMOL addresses key challenges in current AI approaches. The case studies demonstrate GISMOL's potential across healthcare, manufacturing, social science, finance, and governance domains.

9.2. Future Research Directions

1. Scalability Optimization: Improving performance for large constraint systems.
2. Automated Constraint Learning: Learning constraints from data rather than manual specification.
3. Formal Verification: Mathematical proofs of constraint satisfaction under certain conditions.
4. Cognitive Science Validation: Testing COH models against human cognitive performance.
5. Distributed Implementation: Scaling GISMOL systems across multiple computing nodes.
6. Quantum Integration: Exploring quantum computing for constraint satisfaction problems.
7. Ethical Constraint Formalization: Developing frameworks for encoding ethical principles.
8. Cross-modal Learning: Integrating vision, language, and other modalities more seamlessly.
9. Lifelong Learning: Systems that accumulate knowledge over extended periods.
10. Human-AI Collaboration: Improved interfaces for human guidance of GISMOL systems.

9.3. Concluding Remarks

COH/GISMOL represents a promising step toward practical AGI research by providing a unified framework that combines the strengths of neural and symbolic approaches. While challenges remain, the COH theory and evolving GISMOL implementation offer a path forward for creating intelligent systems that are capable, safe, and adaptable across multiple domains.

Conflict of Interest

The author declares no conflict of interest.

References

[1] Anderson, J. R., & Lebiere, C. (1998). *The Atomic Components of Thought*. Psychology Press. https://doi.org/10.4324/9781315805696

[2] Goertzel, B. (2014). Artificial general intelligence: Concept, state of the art, and future prospects. *Journal of Artificial General Intelligence, 5(1)*, 1–46. https://doi.org/10.2478/jagi-2014-0001

[3] Marcus, G. (2020). The next decade in AI: Four steps towards robust artificial intelligence. arXiv preprint. arXiv:2002.06177.

[4] Rudin, C. (2019). Stop explaining black box machine learning models for high stakes decisions and use

interpretable models instead. *Nature Machine Intelligence, 1(5)*, 206–215. https://doi.org/10.1038/s42256-019-0048-x

[5] Levesque, H. J. (2017). *Common Sense, the Turing Test, and the Quest for Real AI*. MIT Press.

[6] Garcez, A.d., & Lamb, L. C. (2023). Neurosymbolic AI: The third wave. *Artificial Intelligence Review, 53*, 12387–12406. https://doi.org/10.1007/s10462-019-09720-w

[7] Garnelo, M., & Shanahan, M. (2019). Reconciling deep learning with symbolic artificial intelligence: Representing objects and relations. *Current Opinion in Behavioral Sciences, 29*, 17–23. doi: 10.1016/j.cobeha.2018.12.010

[8] Besold, T. R. *et al.* (2017). Neural-symbolic learning and reasoning: A survey and interpretation, arXiv preprint, arXiv:1711.03902.

[9] Laird, J. E. (2012). *The SOAR Cognitive Architecture*. MIT Press.

[10] Langley, P. (2019). An integrative framework for artificial intelligence education. *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 33, No. 1, pp. 9670–9677). https://doi.org/10.1609/aaai.v33i01.33019670

[11] Langley, P. (2012). The cognitive systems paradigm. *Advances in Cognitive Systems, 1*, 3–13.

[12] De Raedt, L., Dumančić, S., Manhaeve, R., & Marra, G. (2020). From statistical relational to neuro-symbolic artificial intelligence. *Artificial Intelligence, 287*, 103291. https://doi.org/10.1016/j.artint.2020.103291

[13] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521(7553)*, 436–444. https://doi.org/10.1038/nature14539

[14] Mnih, V. *et al.*, (2015). Human-level control through deep reinforcement learning. *Nature*, *518 (7540)*, 529–533. https://doi.org/10.1038/nature14236

[15] Molnar, C. (2020). *Interpretable machine learning*. Retrieved form https://christophm.github.io/interpretable-ml-book/

[16] Manhaeve, R., Dumančić, S., Kimmig, A., Demeester, T., & De Raedt, L. (2018). DeepProbLog: Neural probabilistic logic programming. *Proceedings of Advances in Neural Information Processing Systems 31 (NeurIPS 2018)*.

[17] Rocktäschel, T., & Riedel, S. (2017). End-to-end differentiable proving. *Proceedings of Advances in Neural Information Processing Systems 30 (NeurIPS 2017)*.

[18] Yi, K., Wu, J., Gan, C., Torralba, A., Kohli, P., & Tenenbaum, J. (2018). Neural-symbolic VQA: Disentangling reasoning from vision and language understanding. *Proceedings of Advances in Neural Information Processing Systems 31 (NeurIPS 2018)*.

[19] Rosenbloom, P. S. (2019). The Sigma cognitive architecture. *International Journal of Machine Learning and Cybernetics*, *10(11)*, 2941–2957. https://doi.org/10.1007/s13042-018-0890-1

[20] Langley, P., Laird, J. E., & Rogers, S. (2019). Cognitive architectures: Research issues and challenges. *Cognitive Systems Research*, *56*, 36–49. https://doi.org/10.1016/j.cogsys.2018.12.009

[21] Rossi, F., van Beek, P., & Walsh, T. (Eds.). (2006). *Handbook of Constraint Programming*. Elsevier.

[22] Dechter, R. (2003). *Constraint Processing*. Morgan Kaufmann. https://doi.org/10.1016/B978-1-55860-890-0.X5000-8

[23] Leucker, M., & Schallhart, C. (2009). A brief account of runtime verification. *Journal of Logic and Algebraic Programming*, *78(5)*, 293–303. https://doi.org/10.1016/j.jlap.2008.08.004

[24] Hassabis, D., Kumaran, D., Summerfield, C., & Botvinick, M. (2017). Neuroscience-inspired artificial intelligence. *Neuron*, *95(2)*, 245–258. https://doi.org/10.1016/j.neuron.2017.09.038

[25] Bengio, Y. (2017). *The Consciousness Prior*. arXiv preprint, arXiv:1709.08568, 2017.

[26] Hawkins, J., Lewis, M., Klukas, M., Purdy, S., & Ahmad, S. (2019). A framework for intelligence and

cortical function based on grid cells in the neocortex. *Frontiers in Neural Circuits, 12(121).* doi: 10.3389/fncir.2018.00121

[27] A. Rajkomar *et al.*, Scalable and accurate deep learning with electronic health records. *NPJ Digital Medicine*, *1(1)*, 18. https://doi.org/10.1038/s41746-018-0029-1

[28] Makary, M. A., & Daniel, M. (2016). Medical error—The third leading cause of death in the US. *BMJ*, *353*, i2139. https://doi.org/10.1136/bmj.i2139

[29] Wang, J., Ma, Y., Zhang, L., Gao, R. X., & Wu, D. (2018). Deep learning for smart manufacturing: Methods and applications. *Journal of Manufacturing Systems*, *48(Part C)*, 144–156. doi: 10.1016/j.jmsy.2018.01.003

[30] Giarratano, J., & Riley, G. (2005). *Expert systems: Principles and Programming* (4th ed.). Thomson/Course Technology.