Liquid AI: An Architectural Framework for Continuously Self-Improving Artificial Intelligence

Thomas R. Caulfield *, Naeyma N. Islam, Rohit Chitale

Posted Date: 28 July 2025

doi: 10.20944/preprints202507.2283.v1

Keywords: liquid AI; adaptive learning; multi-agent AI; self-improving AI; knowledge integration; continual learning; meta-learning

 doi:10.20944/preprints202507.2283.v1

Article

Liquid AI: A Theoretical Framework for Continuously Self-Improving Artificial Intelligence

Thomas R. Caulfield [1,*], Naeyma N. Islam [2] and Rohit Chitale [3]

[1] Digital Ether Computing, Miami, FL, USA
[2] Mayo Clinic, Jacksonville, FL, USA
[3] Biosciences Division, Los Alamos National Laboratory, Los Alamos, NM, USA
* Correspondence: thomas@digitalethercomputing.com

Abstract: We present Liquid AI as a theoretical framework and mathematical basis for artificial intelligence systems capable of continuous structural adaptation and autonomous capability development. This work explores the conceptual boundaries of adaptive AI by formalizing three interconnected mechanisms: (1) entropy-guided hyperdimensional knowledge graphs that could autonomously restructure based on information-theoretic criteria; (2) a self-development engine using hierarchical Bayesian optimization for runtime architecture modification; and (3) a federated multi-agent framework with emergent specialization through distributed reinforcement learning. We address fundamental limitations in current AI systems through mathematically formalized processes of dynamic parameter adjustment, structural self-modification, and cross-domain knowledge synthesis. While immediate implementation faces substantial computational challenges requiring infrastructure on the scale of current large language model training facilities, we provide architectural specifications, theoretical convergence bounds, and evaluation criteria as a foundation for future research. This theoretical exploration establishes mathematical foundations for a potential new paradigm in artificial intelligence that would transition from episodic training to persistent autonomous development, offering a long-term research direction for the field. A comprehensive supplemental materials document provides detailed technical analysis, computational requirements, and an incremental development roadmap spanning approximately a decade.

Keywords: artificial general intelligence; self-modifying systems; adaptive architectures; autonomous learning; knowledge integration; multi-agent intelligence; information theory; theoretical framework

1. Introduction

Contemporary artificial intelligence has achieved remarkable successes in specialized domains, from game playing to natural language processing and scientific discovery [1–3]. These systems excel at pattern recognition [4], natural language understanding [5], and strategic reasoning [6], yet they remain fundamentally constrained by architectural decisions made before deployment. This limitation contrasts sharply with biological intelligence, which exhibits continuous adaptation through structural plasticity [7]. In this paper, we present Liquid AI as a theoretical framework that explores what becomes possible when we remove these architectural constraints, allowing AI systems to modify their own structure during operation.

The term "liquid" captures the essential property we seek: a system that can reshape itself to fit the contours of any problem space while maintaining coherent function. Inspired by complex systems theory [10] and the adaptive properties of biological neural networks [9], our framework formalizes mechanisms for runtime architectural modification, autonomous knowledge synthesis, and emergent multi-agent specialization. While immediate implementation faces significant challenges (detailed in our supplemental materials) this theoretical exploration provides mathematical foundations for a potential new paradigm in artificial intelligence. Figure 1 illustrates the core architectural components

 doi:10.20944/preprints202507.2283.v1

of Liquid AI, demonstrating how the Knowledge Integration Engine, Self-Development Module, and Multi-Agent Coordinator interact dynamically to enable continuous self-improvement.

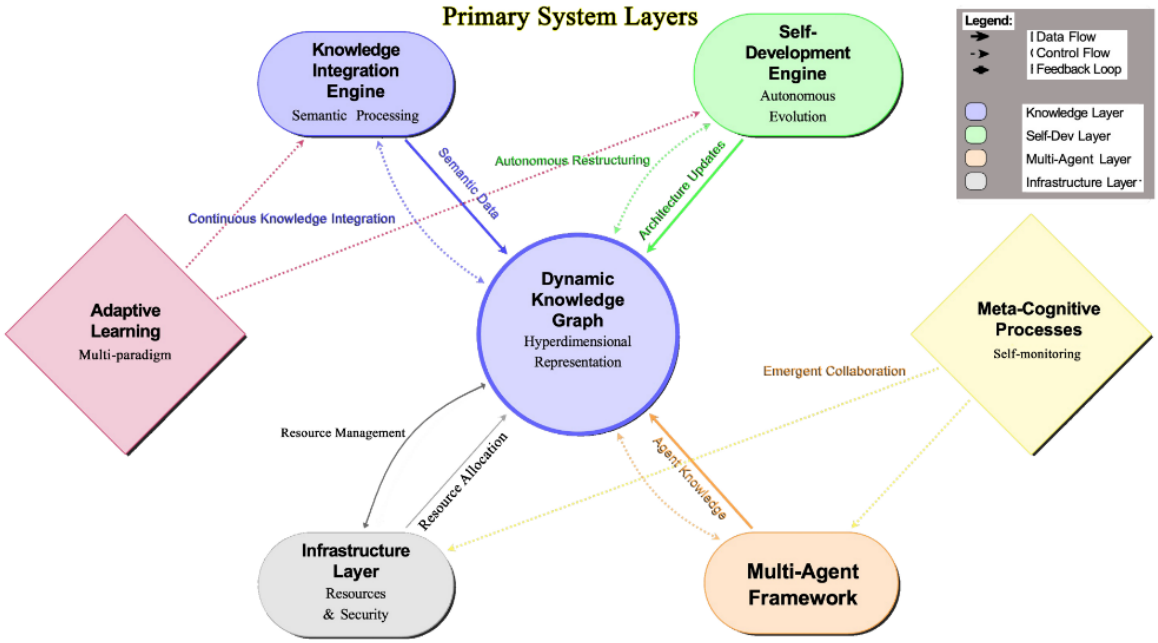

Figure 1. Core architectural components of Liquid AI showing the dynamic interaction between the Knowledge Integration Engine, Self-Development Module, Multi-Agent Coordinator, and supporting infrastructure. Components are color-coded by function with bidirectional data flows indicated by arrows.

2. Current Limitations and Theoretical Opportunities

2.1. Architectural Constraints in Contemporary AI

Modern AI systems operate within predetermined structural boundaries that fundamentally limit their adaptive capacity. Large language models, despite their impressive capabilities, cannot modify their transformer architectures or attention mechanisms after training [5,6]. Computer vision systems remain locked into their convolutional or vision transformer designs [7]. These static architectures create several critical limitations that motivate our theoretical exploration.

Table S1 (Supplemental Material) provides a systematic comparison between traditional AI systems and the proposed Liquid AI framework, highlighting how architectural rigidity constrains current approaches. Figure 2 illustrates these fundamental differences across key dimensions of adaptability, knowledge integration, and autonomous evolution. Figure 2 provides a systematic comparison between traditional AI architectures and the proposed Liquid AI paradigm, highlighting the fundamental shift from static, human-directed systems to dynamic, self-evolving architectures.

 doi:10.20944/preprints202507.2283.v1

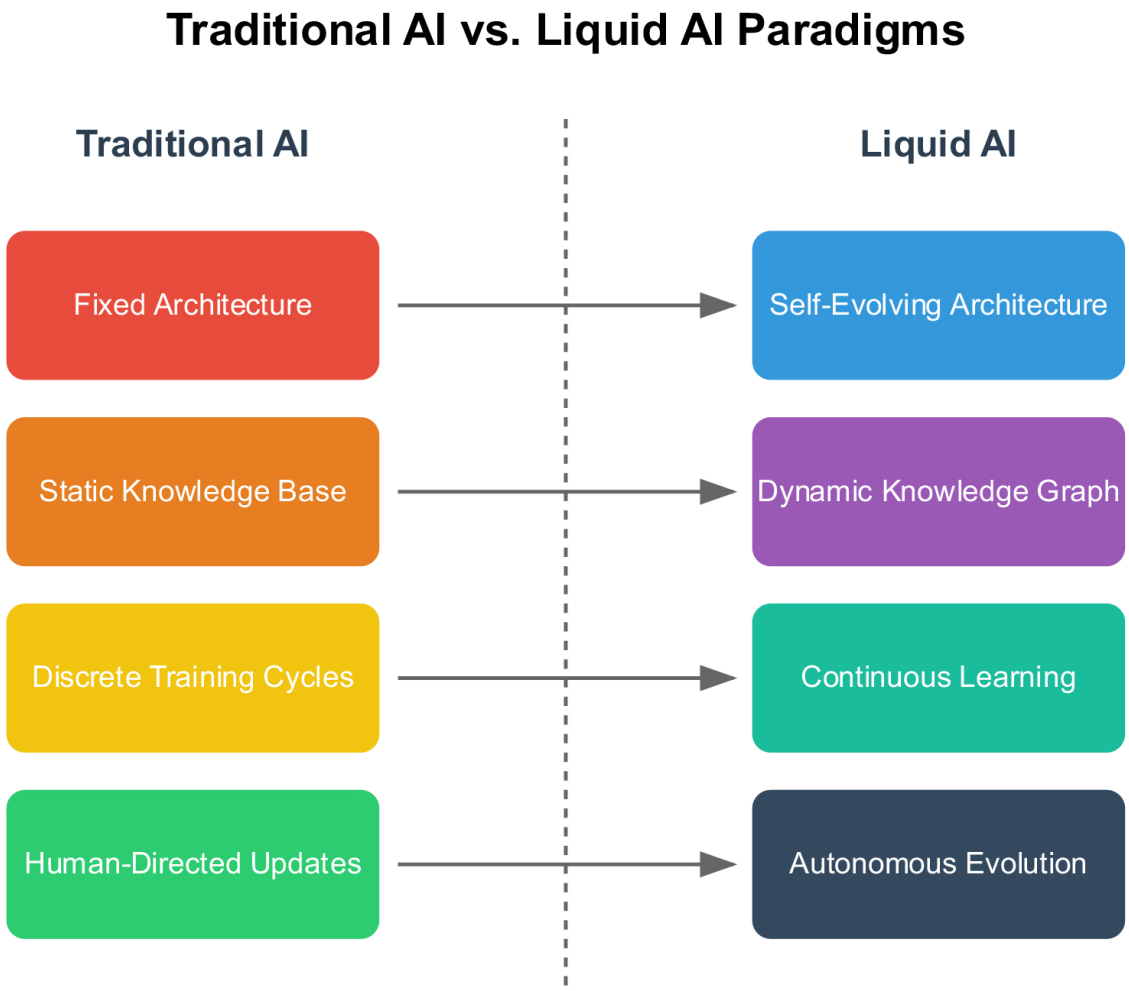

Figure 2. Comparative Analysis of Traditional AI versus Liquid AI Paradigms. Systematic comparison between traditional AI and Liquid AI (right) approaches across four key dimensions: Fixed Architecture vs. Self-Evolving Architecture, Static Knowledge Base vs. Dynamic Knowledge Graph, Discrete Training Cycles vs. Continuous Learning, and Human-Directed Updates vs. Autonomous Evolution. Connecting arrows demonstrate the evolutionary progression from traditional to liquid paradigms.

 doi:10.20944/preprints202507.2283.v1

Table 1. Comprehensive Comparison of Liquid AI with Existing Adaptive AI Methods.

Capability	Liquid AI (Ours)	EWC [81]	MAML [17]	DARTS [82]	PackNet [70]	QMIX [83]
Architectural Adaptation						
Runtime Architecture Modification	✓	×	×	×	×	×
Topological Plasticity	✓	×	×	×	×	×
Autonomous Structural Evolution	✓	×	×	×	×	×
Pre-deployment Architecture Search	N/A	×	×	✓	×	×
Learning Capabilities						
Continual Learning	✓	✓	✓	×	✓	×
Catastrophic Forgetting Prevention	✓	✓	✓	N/A	✓	N/A
Cross-Domain Knowledge Transfer	✓	Limited	✓	×	Limited	×
Zero-Shot Task Adaptation	✓	×	✓	×	×	×
Self-Supervised Learning	✓	×	×	×	×	×
Knowledge Management						
Dynamic Knowledge Graphs	✓	×	×	×	×	×
Entropy-Guided Optimization	✓	×	×	×	×	×
Cross-Domain Reasoning	✓	×	Limited	×	×	×
Temporal Knowledge Evolution	✓	×	×	×	×	×
Multi-Agent Capabilities						
Emergent Agent Specialization	✓	N/A	N/A	N/A	N/A	×
Dynamic Agent Topology	✓	N/A	N/A	N/A	N/A	×
Collective Intelligence	✓	N/A	N/A	N/A	N/A	✓
Autonomous Role Assignment	✓	N/A	N/A	N/A	N/A	×
Performance Characteristics						
Sustained Improvement	✓	×	×	×	×	×
Resource Efficiency	Adaptive	Fixed	Fixed	Fixed	Fixed	Fixed
Scalability	Unlimited	Limited	Limited	Limited	Limited	Moderate
Interpretability	Dynamic	Low	Low	Moderate	Low	Low
Deployment Flexibility						
Online Adaptation	✓	Limited	Limited	×	Limited	Limited
Distributed Deployment	✓	×	×	×	×	✓
Hardware Agnostic	✓	✓	✓	✓	✓	✓
Real-Time Operation	✓	✓	✓	×	✓	✓

Five Fundamental Limitations

Current AI systems face five interconnected limitations that stem from their static nature:

Parameter Rigidity

Once trained, neural architectures cannot evolve their topology. While parameter-efficient adaptation methods like LoRA [9] and fine-tuning [8] enable limited modifications within fixed structures, they cannot add new pathways, remove redundant components, or fundamentally reorganize information flow. This contrasts with biological systems where synaptic plasticity includes both weight modification and structural changes [10].

Knowledge Fragmentation

Information remains isolated within predefined domains. Transfer learning [11] and multi-task learning [12] provide mechanisms for sharing knowledge across related tasks, but these require human-specified task relationships. Current systems lack the ability to autonomously discover and exploit latent connections between disparate knowledge domains.

Human-Dependent Evolution

Improvements require human intervention through architectural redesign, hyperparameter tuning, or complete retraining. While AutoML [13] and Neural Architecture Search [14] automate aspects

 doi:10.20944/preprints202507.2283.v1

of model development, they operate within human-defined search spaces during distinct optimization phases, not during deployment.

Catastrophic Forgetting

Sequential learning in current systems leads to degradation of previously acquired capabilities [15]. Although continual learning methods [16] mitigate this issue through various memory mechanisms, they operate within the constraint of fixed architectures, limiting their ability to expand capabilities without interference.

Limited Meta-Learning

Current meta-learning approaches enable rapid adaptation to new tasks within a distribution [17,18], but cannot modify their fundamental learning algorithms or architectural constraints during deployment. The meta-learning process itself remains static, unable to evolve based on experience.

2.2. Theoretical Foundations from Natural Systems

Nature provides compelling examples of systems exhibiting the properties we seek. Biological neural networks demonstrate remarkable plasticity, continuously forming and pruning connections in response to experience [19]. Social insect colonies exhibit collective intelligence emerging from simple local interactions without central coordination [20]. These natural systems inspire our approach while highlighting the gap between current AI and truly adaptive intelligence.

Complex adaptive systems theory offers insights into how simple components can give rise to sophisticated collective behaviors [10]. Key principles include distributed control without central authority, adaptive feedback loops that modify behavior based on environmental signals, phase transitions where small parameter changes lead to qualitative behavioral shifts, and hierarchical organization where complex behaviors emerge at multiple scales. These principles inform our design of Liquid AI, where architectural evolution emerges from information-theoretic optimization rather than predetermined rules.

2.3. Core Contributions and Article Structure

This paper makes four fundamental contributions to the theoretical foundations of artificial intelligence:

We introduce Liquid AI as a comprehensive theoretical framework for AI systems capable of continuous structural self-modification during deployment. Unlike existing approaches that modify parameters within fixed architectures, our framework formalizes mechanisms for runtime topological evolution. We establish mathematical foundations including convergence bounds for self-modifying systems and formal conditions under which architectural evolution preserves functional coherence while enabling capability expansion.

Our theoretical analysis addresses fundamental questions about the feasibility and behavior of self-modifying AI systems. We develop comprehensive evaluation frameworks for assessing adaptive AI systems, including metrics for temporal evolution, cross-domain integration, and emergent capabilities that extend beyond traditional static benchmarks. Finally, we provide detailed implementation considerations in the supplemental materials, including computational complexity analysis, distributed computing requirements, and a decade-spanning incremental development roadmap.

The computational foundations underlying these mechanisms are summarized in Table S1. Figure 1 illustrates the core architectural components and their theoretical interactions, providing a visual overview of the complete system.

The remainder of this paper is structured as follows: Section 3 details the Liquid AI architecture with its dynamic knowledge graphs and self-modification mechanisms. Section 4 explores self-development processes including hierarchical task decomposition. Section 5 presents the multi-agent collaboration framework. Section 6 covers knowledge integration mechanisms. Section 7 addresses implementation considerations. Section 8 outlines evaluation methodologies. Section 9 explores

 doi:10.20944/preprints202507.2283.v1

potential applications. Section 10 discusses implications and future directions. The supplemental materials provide additional technical depth, baseline comparisons with existing systems, and detailed analysis of computational requirements.

By establishing theoretical foundations for continuously adaptive AI, this work aims to inspire research toward systems that match the flexibility of biological intelligence while leveraging the computational advantages of artificial systems. While immediate implementation faces significant challenges, we believe this theoretical exploration opens important new directions for the field.

3. Liquid AI Architecture

3.1. Architectural Overview

The Liquid AI architecture represents a theoretical framework for dynamically evolving computational systems that extend beyond traditional adaptive approaches. Unlike existing methods that modify parameters within fixed structures [21], Liquid AI explores the mathematical foundations for runtime topological modifications guided by information-theoretic principles. This framework builds on concepts from modular networks [23] and meta-learning [17], but extends them to enable autonomous structural evolution during deployment.

We formalize the architecture as an adaptive system Λ defined by:

$$\Lambda(t) = \{\Omega(t), \Gamma(t), \Phi(t), \Theta(t), \Xi(t)\} \tag{1}$$

where each component represents a functionally distinct subsystem: $\Omega(t)$ implements the dynamic knowledge graph with entropy-guided restructuring, $\Gamma(t)$ represents the self-development engine using hierarchical Bayesian optimization, $\Phi(t)$ encapsulates the multi-agent collaborative framework, $\Theta(t)$ describes adaptive learning mechanisms, and $\Xi(t)$ represents meta-cognitive processes for system-level optimization.

This architecture exhibits three theoretical properties that distinguish it from current systems. First, topological plasticity enables runtime modification of network connectivity based on information flow patterns, extending beyond weight adaptation to structural reconfiguration. Second, compositional adaptivity allows dynamic instantiation and dissolution of functional modules based on task demands. Third, meta-learning autonomy enables self-modification of learning algorithms through nested optimization without external task specification.

Figure 1 illustrates the core architectural components and their interactions, while Table S2 (Supplementary Material) provides a detailed breakdown of each component's primary functions and key innovations.

3.2. Core System Components

3.2.1. Dynamic Knowledge Graph

The Dynamic Knowledge Graph (DKG) forms the foundational knowledge substrate, implementing hyperdimensional representations that evolve according to information-theoretic criteria. Unlike static knowledge graphs [24] or temporal graphs with predetermined update rules [22], our DKG implements autonomous structural evolution through entropy-guided optimization:

$$\Omega(t) = \{V(t), E(t), W(t), A(t)\} \tag{2}$$

The graph evolves through the following update mechanism:

 doi:10.20944/preprints202507.2283.v1

Algorithm 1 Dynamic Knowledge Graph Update

Require: Current graph $\Omega(t)$, new information $I(t)$, threshold τ
Ensure: Updated graph $\Omega(t+1)$
Compute information gain: $IG(v) = H(V) - H(V|v)$ for all vertices v
Identify high-entropy regions: $R = \{v : H(v) > \tau\}$
for each region $r \in R$ **do**
  Generate candidate modifications: $\Delta\Omega_r = f_{\text{propose}}(r, I(t))$
  Evaluate via information bottleneck: $IB(\Delta\Omega_r) = I(X;Z) - \beta I(Z;Y)$
  if $IB(\Delta\Omega_r) > IB(\Omega(t))$ **then**
    Apply modification: $\Omega(t) \leftarrow \Omega(t) + \Delta\Omega_r$
  end if
end for
Prune low-information edges: $E(t+1) = \{e \in E(t) : I(e) > \epsilon\}$
return $\Omega(t+1)$

The update dynamics follow gradient flow on an information-theoretic objective:

$$\frac{d\Omega}{dt} = f_{\text{update}}(\Omega, \nabla_\Omega \mathcal{L}, I(t)) \tag{3}$$

where f_{update} incorporates both gradient-based optimization and entropy-guided structural modifications.

3.2.2. Self-Development Engine

The Self-Development Engine $\Gamma(t)$ implements autonomous architectural evolution through hierarchical Bayesian optimization. Building on advances in neural architecture search [25,26], our theoretical approach extends these concepts to enable runtime modifications:

$$\Gamma(t) = \{\Psi(t), \Delta(t), \Upsilon(t)\} \tag{4}$$

The self-development process operates through the following mechanism:

Algorithm 2 Self-Development Process

Require: Performance history $\mathcal{P}_{1:t}$, current architecture $\Lambda(t)$
Ensure: Modified architecture $\Lambda(t+1)$
Initialize Gaussian Process: $GP(\mu, k)$ over architecture space
Compute acquisition function: $\alpha(x) = \mu(x) + \kappa\sigma(x)$
Sample candidate architectures: $\{\Lambda_i\} \sim \alpha(x)$
for each candidate Λ_i **do**
  Estimate performance via surrogate model: $\hat{P}_i = f_{\text{surrogate}}(\Lambda_i)$
  Compute modification cost: $C_i = d(\Lambda(t), \Lambda_i)$
  Score candidate: $S_i = \hat{P}_i / (1 + \lambda C_i)$
end for
Select best candidate: $\Lambda^* = \arg\max_i S_i$
Apply gradual transformation: $\Lambda(t+1) = (1-\alpha)\Lambda(t) + \alpha\Lambda^*$
return $\Lambda(t+1)$

3.2.3. Multi-Agent Collaborative Framework

The framework implements distributed intelligence through specialized agents that theoretically emerge via interaction. This extends multi-agent reinforcement learning [27,28] by enabling agents to modify their own architectures based on specialization needs:

$$\Phi(t) = \{A_1(t), A_2(t), ..., A_n(t), C(t)\} \tag{5}$$

Agent communication follows information-theoretic principles:

 doi:10.20944/preprints202507.2283.v1

$$M_{ij}(t) = h_{\text{comm}}(A_i(t), A_j(t), S(t)) \tag{6}$$

3.2.4. Adaptive Learning Mechanisms

Our adaptive learning layer integrates multiple learning paradigms that operate synergistically:

$$\Theta(t) = \{\alpha(t), \beta(t), \gamma(t), \delta(t)\} \tag{7}$$

where components represent reinforcement learning, unsupervised learning, transfer learning, and meta-learning respectively.

3.2.5. Meta-Cognitive Processes

Meta-cognitive processes implement system-level awareness and strategic planning, extending ideas from cognitive architectures [29,30] to self-modifying systems:

$$\Xi(t) = \{S(t), E(t), R(t), O(t)\} \tag{8}$$

3.3. Information Flow and System Dynamics

3.3.1. Temporal Evolution

The complete system evolves according to coupled differential equations that capture interdependencies between components:

$$\frac{d\Lambda}{dt} = F(\Lambda, E, t) \tag{9}$$

where the evolution function F creates complex feedback dynamics enabling emergent behaviors.

3.3.2. Information Propagation

Information flows through the architecture via learnable transfer functions that adapt based on utility, implementing attention-like mechanisms [3] at the architectural level:

$$I_{ij}(t) = T_{ij}(O_i(t), \Lambda(t)) \tag{10}$$

3.3.3. Stability and Convergence

To prevent chaotic behavior while enabling growth, we implement theoretical Lyapunov-based stability constraints addressing key challenges in self-modifying systems [31]:

$$\mathcal{S}(\Lambda) = \sum_i w_i \cdot \text{Lyapunov}_i(\Lambda) \tag{11}$$

3.3.4. Information-Theoretic Optimization

Knowledge graph evolution follows entropy minimization principles that enable automatic discovery of knowledge hierarchies:

$$\min_{\mathcal{G}} \mathcal{H}[\mathcal{G}] = -\sum_{i,j} p(e_{ij}) \log p(e_{ij}) + \lambda \mathcal{R}(\mathcal{G}) \tag{12}$$

3.3.5. Adaptive Computational Graphs

Computational graphs restructure dynamically based on task requirements, extending dynamic neural architecture approaches [32] to runtime adaptation:

$$G_{\text{comp}}(t+1) = G_{\text{comp}}(t) + \Delta G_{\text{task}}(t) \tag{13}$$

 doi:10.20944/preprints202507.2283.v1

3.4. System Boundaries and Theoretical Guarantees

3.4.1. Environmental Interaction

Liquid AI interfaces with its environment through adaptive channels implementing principles from active inference [33]:

$$\Phi_{\text{env}}(t) = \{I_{\text{in}}(t), O_{\text{out}}(t), F_{\text{feedback}}(t)\} \tag{14}$$

3.4.2. Secure Containment

Given the system's self-modification capabilities, we propose formal containment measures addressing AI safety concerns [34,35]:

$$\mathcal{B}(\Lambda, A) = \{P(A|\Lambda), C(\Lambda), V(\Lambda, A)\} \tag{15}$$

3.4.3. Theoretical Performance Bounds

We establish bounds on system capabilities and the rate of capability improvement through self-modification:

$$\frac{d\mathcal{P}_{\max}}{dt} = \eta \cdot \nabla_{\Lambda}\mathcal{P} \cdot \frac{d\Lambda}{dt} \tag{16}$$

The self-development cycle, illustrated in Figure 3, demonstrates the continuous feedback loop of assessment, planning, execution, and reflection that enables autonomous capability expansion.

 doi:10.20944/preprints202507.2283.v1

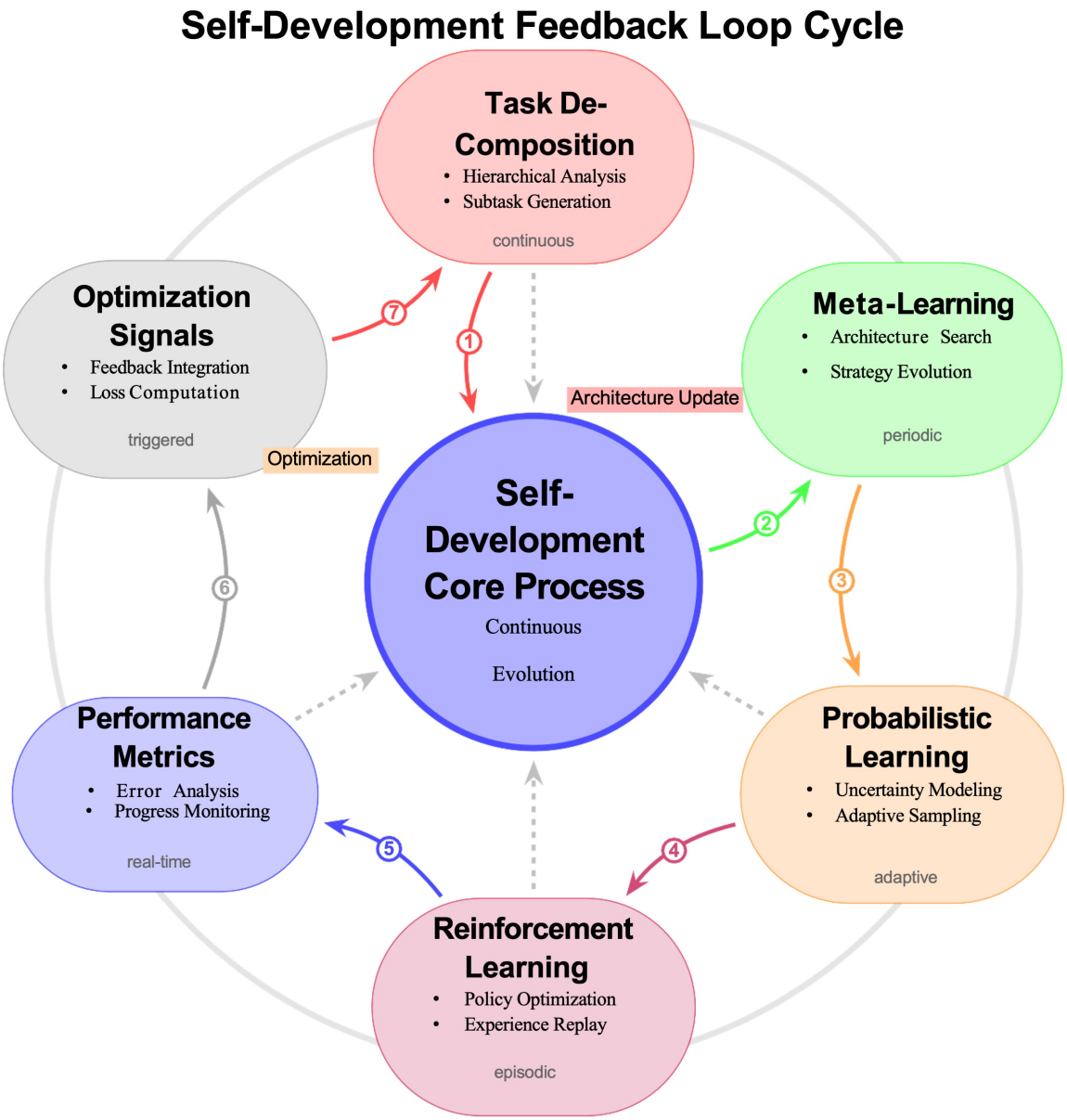

Figure 3. Self-Development Mechanisms and Feedback Loops in Liquid AI showing the continuous cycle of assessment, planning, execution, and reflection.

4. Self-Development Mechanisms

4.1. Foundational Principles of Self-Development

The Self-Development Engine represents our core theoretical innovation for enabling autonomous capability evolution. Unlike AutoML systems that operate in discrete optimization phases [36,37], Liquid AI explores continuous evolution during deployment through internally-driven processes. Figure 3 illustrates these self-development mechanisms and their feedback loops.

We formalize self-development as a nested optimization problem:

$$\mathcal{A}^* = \arg\max_{\mathcal{A}} \mathbb{E}_{\mathcal{T}\sim p(\mathcal{T})}[\mathcal{P}(\mathcal{A},\mathcal{T}) - \lambda\mathcal{C}(\mathcal{A})] \tag{17}$$

This formulation enables the theoretical discovery of architectural innovations that improve performance across diverse tasks while maintaining computational efficiency.

 doi:10.20944/preprints202507.2283.v1

4.2. Hierarchical Task Decomposition

Complex objectives naturally decompose into hierarchical structures through information-theoretic analysis:

$$\mathcal{S} = \arg\max_{\mathcal{S}} I(\mathcal{S}; \mathcal{G}) - \beta H(\mathcal{S}) \tag{18}$$

Policies organize hierarchically with high-level controllers selecting among low-level primitives:

$$\pi(a|s) = \sum_{o} \pi_{\text{high}}(o|s)\pi_{\text{low}}(a|s,o) \tag{19}$$

4.3. Meta-Learning for Architectural Adaptation

Building on meta-learning principles [38,39], we extend adaptation to architectural parameters. We distinguish between object-level parameters θ and meta-parameters ϕ that control architectural properties:

$$\mathcal{L}(\theta, \phi) = \mathbb{E}_{\mathcal{T}}[\mathcal{L}_{\mathcal{T}}(f_\theta(\phi)) + \gamma R(\phi)] \tag{20}$$

4.3.1. Bilevel Optimization

Architectural adaptation follows a bilevel optimization scheme [40]:

$$\phi^* = \arg\min_{\phi} \mathcal{L}_{\text{val}}(\theta^*(\phi), \phi) \tag{21}$$

$$\text{s.t. } \theta^*(\phi) = \arg\min_{\theta} \mathcal{L}_{\text{train}}(\theta, \phi) \tag{22}$$

Gradients propagate through the bilevel optimization using implicit differentiation [41].

Algorithm 3 Online Bayesian Architecture Optimization

Require: Initial architecture θ_0, performance function f
Ensure: Optimized architecture trajectory $\{\theta_t\}$
1: Initialize GP prior: $f \sim \mathcal{GP}(0, k(\theta, \theta'))$
2: **while** system is deployed **do**
3: Observe performance: $y_t = f(\theta_t) + \epsilon_t$
4: Update posterior: $p(f|D_{1:t}) \propto p(y_t|f, \theta_t)p(f|D_{1:t-1})$
5: Compute acquisition: $\alpha(\theta) = \text{EI}(\theta|D_{1:t})$
6: Select next architecture: $\theta_{t+1} = \arg\max_\theta \alpha(\theta)$
7: Apply smooth transition: $\theta_{t+1} \leftarrow \lambda\theta_t + (1-\lambda)\theta_{t+1}$
8: **end while**

4.4. Probabilistic Program Synthesis

Liquid AI theoretically synthesizes new computational modules through probabilistic programming. New modules are sampled from a learned distribution:

$$m \sim p(m|\mathcal{C}, \mathcal{H}) \tag{23}$$

Modules compose through typed interfaces ensuring compatibility:

$$M_{\text{composite}} = \lambda x.m_2(m_1(x, \theta_1), \theta_2) \tag{24}$$

4.5. Reinforcement Learning for Architectural Evolution

The system treats architectural modifications as actions in a Markov Decision Process with state space encoding current architecture and performance:

 doi:10.20944/preprints202507.2283.v1

$$s_t = [\mathcal{A}_t, \mathcal{P}_t, \mathcal{M}_t, \mathcal{H}_t] \tag{25}$$

Rewards balance multiple objectives:

$$r_t = \alpha_1 \Delta\mathcal{P}_t + \alpha_2 \mathcal{E}_t - \alpha_3 \mathcal{C}_t - \alpha_4 \mathcal{I}_t \tag{26}$$

4.6. Optimization Algorithms and Theoretical Analysis

The system combines multiple optimization methods through a hybrid approach [44]:

$$\Delta\phi = \alpha_g \Delta\phi_{\text{gradient}} + \alpha_e \Delta\phi_{\text{evolution}} + \alpha_r \Delta\phi_{\text{RL}} \tag{27}$$

Under mild assumptions, the self-development process converges to locally optimal architectures. Given Lipschitz continuous performance function $\mathcal{P}$ and bounded architecture space Φ, the sequence $\{\phi_t\}$ converges to a stationary point ϕ^* such that $||\nabla_\phi \mathcal{P}(\phi^*)|| < \epsilon$ in $O(1/\epsilon^2)$ iterations.

Self-modifications preserve system stability through Lyapunov analysis:

$$V(\mathcal{A}_{t+1}) - V(\mathcal{A}_t) \leq -\alpha||\Delta\mathcal{A}_t||^2 + \beta||\xi_t||^2 \tag{28}$$

4.7. Integrated Self-Development Framework

The complete self-development framework operates through multiple feedback loops driving continuous improvement:

Algorithm 4 Adaptive Architecture Evolution

Input: Initial architecture $\mathcal{A}_0$, task distribution $p(\mathcal{T})$
Output: Evolved architecture $\mathcal{A}^*$
Initialize meta-parameters ϕ_0, performance history $\mathcal{H} = \emptyset$
while not converged **do**
  Sample batch of tasks $\{\mathcal{T}_i\} \sim p(\mathcal{T})$
  for each task $\mathcal{T}_i$ **do**
    Train parameters: $\theta_i = \arg\min_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta(\phi))$
    Evaluate: $p_i = \mathcal{P}(f_{\theta_i}(\phi), \mathcal{T}_i)$
  end for
  Update history: $\mathcal{H} = \mathcal{H} \cup \{(\phi, \{p_i\})\}$
  Compute architecture gradient: $g = \nabla_\phi \sum_i p_i$
  Sample evolutionary perturbations: $\{\epsilon_j\} \sim \mathcal{N}(0, I)$
  Evaluate perturbations: $\{q_j\} = \{\mathcal{P}(\phi + \sigma\epsilon_j)\}$
  Compute evolution update: $\Delta\phi_e = \sum_j w_j \epsilon_j$
  Combined update: $\phi = \phi + \eta_g g + \eta_e \Delta\phi_e$
  Synthesize new modules: $m \sim p(m|\mathcal{H})$
  Integrate promising modules into architecture
end while
return $\mathcal{A}^* = f(\phi^*)$

This comprehensive self-development framework provides the theoretical foundation for AI systems that could transcend the limitations of static architectures, continuously evolving to meet new challenges without human intervention. The mathematical foundations and operational characteristics are detailed in Table S6 (Supplemental Material).

5. Multi-Agent Collaboration Framework

5.1. Theoretical Foundations of Multi-Agent Systems

Our multi-agent framework builds on established principles from game theory [43], distributed optimization [44], and swarm intelligence [45]. Unlike traditional multi-agent reinforcement learning that assumes fixed agent architectures [46], Liquid AI explores theoretical mechanisms for emergent

 doi:10.20944/preprints202507.2283.v1

specialization without predefined roles. This framework enables agents to autonomously develop specialized capabilities through adaptive coordination mechanisms, as illustrated in Figure 4, which uses federated multi-agent architecture; showing how heterogeneous agents self-organize into specialized communities while maintaining global coherence through shared protocols.

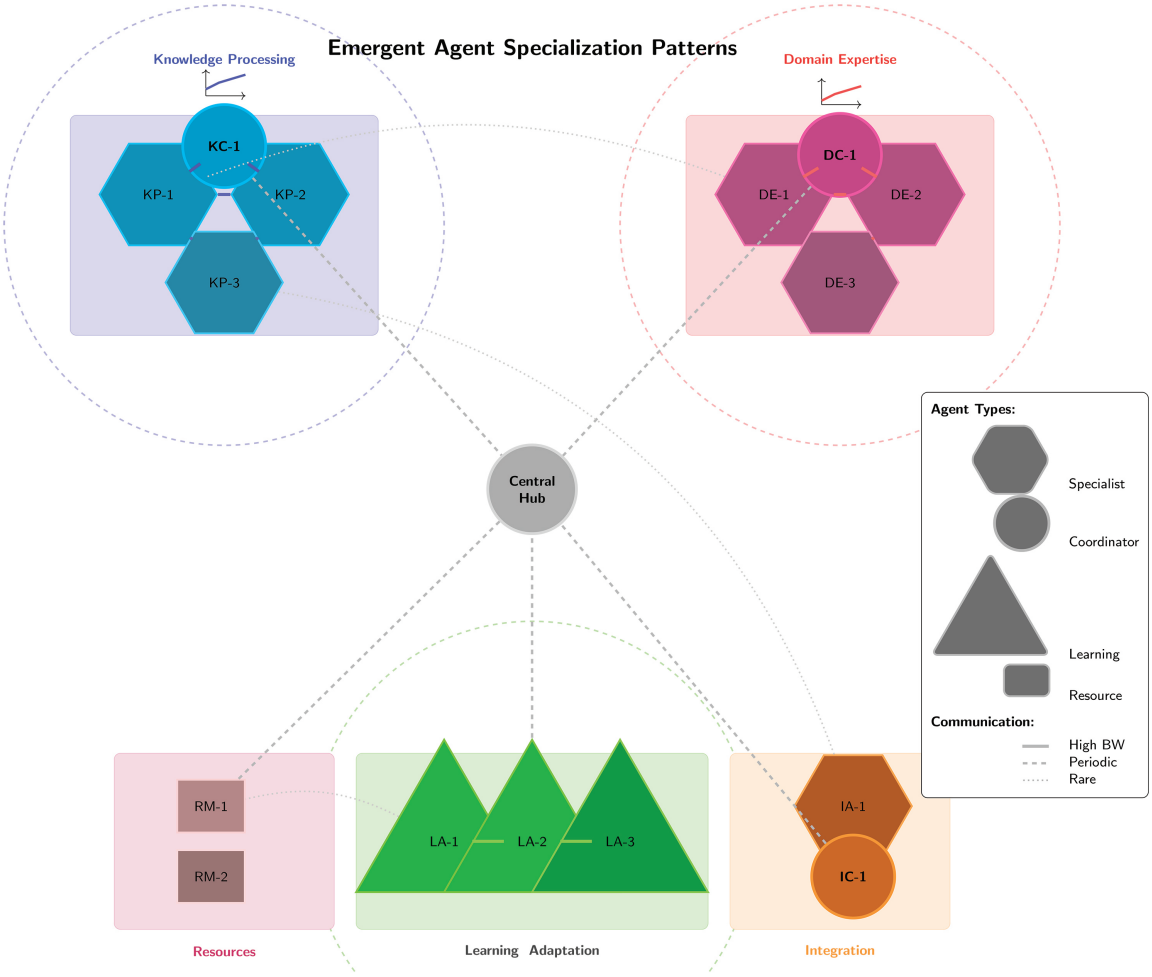

Figure 4. Federated Multi-Agent Architecture showing emergent specialization patterns and communication pathways between heterogeneous agents.

We model the multi-agent system as a Decentralized Partially Observable Markov Decision Process (Dec-POMDP):

$$\mathcal{M} = \langle \mathcal{I}, \mathcal{S}, \{\mathcal{A}_i\}, \{\mathcal{O}_i\}, P, \{R_i\}, \{\Omega_i\}, \gamma \rangle \tag{29}$$

where $\mathcal{I}$ denotes the set of agents, $\mathcal{S}$ represents the joint state space, $\mathcal{A}_i$ and $\mathcal{O}_i$ are action and observation spaces for agent i, P defines transition dynamics, R_i specifies individual reward functions, Ω_i represents observation functions, and γ is the discount factor.

5.2. Agent Architecture and Capabilities

Each agent in the theoretical Liquid AI framework possesses modular neural architecture consisting of specialized components:

$$A_i = \{M_{\text{perception}}, M_{\text{reasoning}}, M_{\text{action}}, M_{\text{communication}}, M_{\text{adaptation}}\} \tag{30}$$

Each agent maintains a local knowledge graph that interfaces with the global knowledge system:

 doi:10.20944/preprints202507.2283.v1

$$\mathcal{G}_i^{t+1} = \mathcal{U}(\mathcal{G}_i^t, \mathcal{K}_{\text{local}}^t, \mathcal{K}_{\text{shared}}^t) \tag{31}$$

Agent capabilities evolve through experience using gradient-based updates:

$$C_i^{t+1} = C_i^t + \eta \nabla_{C_i} J_i(C_i^t, \mathcal{E}_i^t) \tag{32}$$

5.3. Emergent Specialization and Dynamic Topology

Specialization emerges through competitive-collaborative dynamics without explicit role assignment. The system discovers efficient task allocation through mutual information maximization:

$$\max_{\pi_1,\ldots,\pi_n} I(\mathcal{T}; \mathcal{A}) - \lambda H(\mathcal{A}|\mathcal{T}) \tag{33}$$

Agents develop complementary skills through diversity bonuses in their reward structure:

$$R_i^{\text{total}} = R_i^{\text{task}} + \alpha D(S_i, \{S_j\}_{j \neq i}) \tag{34}$$

The agent topology evolves to minimize communication overhead while maximizing task performance:

Algorithm 5 Adaptive Agent Topology Evolution

Require: Current topology $G(t)$, performance metrics $\mathcal{P}(t)$, task characteristics $\mathcal{T}(t)$
Ensure: Updated topology $G(t+1)$
Compute agent relevance: $r_{ij} = I(A_i; A_j|\mathcal{T})$ for all agent pairs
Identify communication bottlenecks: $B = \{(i,j) : C_{ij} > \tau_{\text{comm}}\}$
for each bottleneck $(i,j) \in B$ **do**
  Evaluate direct connection utility: $U_{ij} = \mathcal{P}_{\text{with}} - \mathcal{P}_{\text{without}}$
  if $U_{ij} > \tau_{\text{utility}}$ **then**
    Add edge: $E(t+1) \leftarrow E(t) \cup \{(i,j)\}$
  end if
end for
Prune low-utility connections: $E(t+1) = \{e \in E(t) : U(e) > \epsilon\}$
Update edge weights via gradient ascent: $W(t+1) = W(t) + \alpha \nabla_W \mathcal{P}$
return $G(t+1) = (V(t), E(t+1), W(t+1))$

5.4. Coordination Mechanisms

5.4.1. Decentralized Consensus

Agents reach consensus through iterative belief propagation extending classical distributed consensus [47] with learned aggregation functions:

$$x_i(t+1) = x_i(t) + \sum_{j \in \mathcal{N}_i} f_{ij}(x_j(t) - x_i(t), \text{context}) \tag{35}$$

 doi:10.20944/preprints202507.2283.v1

Algorithm 6 Adaptive Consensus Protocol

Require: Agent states $\{x_i\}$, reliability scores $\{r_i\}$
Ensure: Consensus state x^*
while not converged **do**
  for each agent i **do**
    Compute influence weights: $w_{ij} = \frac{r_j \cdot \exp(-d(x_i,x_j)/\tau)}{\sum_k r_k \cdot \exp(-d(x_i,x_k)/\tau)}$
    Update state: $x_i \leftarrow x_i + \alpha \sum_j w_{ij}(x_j - x_i)$
    Update reliability: $r_i \leftarrow \beta r_i + (1-\beta)\text{accuracy}_i$
  end for
end while
return $x^* = \sum_i r_i x_i / \sum_i r_i$

5.4.2. Hierarchical Organization

The system can theoretically self-organize into hierarchical structures when beneficial. Meta-agents form when groups develop stable collaboration patterns:

$$M_j = \{A_{j1}, A_{j2}, \ldots, A_{jm}, \mathcal{C}_j\} \tag{36}$$

Algorithm 7 Meta-Agent Formation

Require: Agent population $\mathcal{A}$, interaction history $\mathcal{H}$
Ensure: Meta-agent assignments $\mathcal{M}$
Compute interaction strength: $S_{ij} = \sum_t w(t) \cdot I_{ij}(t)$
Apply spectral clustering to S to identify communities
for each community C_k **do**
  Evaluate cohesion: $\text{coh}(C_k) = \frac{\sum_{i,j \in C_k} S_{ij}}{|C_k|^2}$
  if $\text{coh}(C_k) > \tau_{\text{meta}}$ **then**
    Form meta-agent: $M_k = (C_k, f_{\text{aggregate}})$
    Learn aggregation function: $f_{\text{aggregate}} = \arg\min_f \mathcal{L}_{\text{coord}}$
  end if
end for
return $\mathcal{M} = \{M_1, M_2, \ldots\}$

5.5. Distributed Learning and Credit Assignment

5.5.1. Collaborative Policy Optimization

We extend multi-agent policy gradient methods [42] with dynamic credit assignment:

$$\pi^*_{\text{joint}} = \arg\max_{\pi} \mathbb{E}_{\tau \sim \pi} \left[R_{\text{global}}(\tau) + \sum_i \lambda_i R_i(\tau) \right] \tag{37}$$

Individual policy updates incorporate learned credit assignment through counterfactual reasoning:

$$A^t_i = Q(s^t, a^t) - \sum_{a'_i} \pi_i(a'_i | s^t_i) Q(s^t, (a'_i, a^t_{-i})) \tag{38}$$

5.5.2. Multi-Agent Credit Assignment

We implement a learnable credit assignment mechanism based on Shapley values [46]:

$$\phi_i = \sum_{S \subseteq \mathcal{A} \setminus \{i\}} \frac{|S|!(n - |S| - 1)!}{n!} [v(S \cup \{i\}) - v(S)] \tag{39}$$

where $v(S)$ is approximated through learned value functions.

 doi:10.20944/preprints202507.2283.v1

6. Knowledge Integration Engine

The Knowledge Integration Engine orchestrates cross-domain knowledge synthesis and maintains the system's episodic and semantic memory. Unlike traditional knowledge bases that store static information [48,49], our theoretical approach explores dynamic knowledge restructuring based on usage patterns and information-theoretic optimization.

6.1. Dynamic Knowledge Representation

6.1.1. Hyperdimensional Graph Neural Networks

Our framework extends graph neural networks [51,52] to hypergraph structures with temporal dynamics. Knowledge is encoded in high-dimensional continuous spaces:

$$\mathbf{k} = f_{\text{encode}}(c, r, t) \in \mathbb{R}^d \tag{40}$$

where c represents concept content, r denotes relational context, and t encodes temporal information.

The hyperdimensional graph structure evolves through:

Algorithm 8 Hyperdimensional Graph Evolution

Require: Current graph $\mathcal{G}(t)$, new information $I(t)$, threshold τ
Ensure: Updated graph $\mathcal{G}(t+1)$
Embed new information: $\mathbf{i} = \text{encode}(I(t))$
Identify insertion points: $V_{\text{cand}} = \{v \in V : \text{sim}(\mathbf{v}, \mathbf{i}) > \tau\}$
if $|V_{\text{cand}}| = 0$ **then**
  Create new vertex: $v_{\text{new}} = \text{init}(\mathbf{i})$
  $V(t+1) \leftarrow V(t) \cup \{v_{\text{new}}\}$
else
  Form hyperedge: $h_{\text{new}} = V_{\text{cand}} \cup \{v_{\text{new}}\}$
  $H(t+1) \leftarrow H(t) \cup \{h_{\text{new}}\}$
end if
Update embeddings via gradient flow: $\mathbf{V}(t+1) = \mathbf{V}(t) - \eta \nabla_V \mathcal{L}_{\text{info}}$
Prune redundant structures: $\mathcal{G}(t+1) = \text{prune}(\mathcal{G}(t+1), \epsilon)$
return $\mathcal{G}(t+1)$

6.2. Information-Theoretic Knowledge Organization

6.2.1. Transformer-Based Relational Reasoning

We extend transformer architectures [3] to knowledge graph reasoning with structural masks:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T + \mathbf{M}_{\text{struct}}}{\sqrt{d_k}}\right)\mathbf{V} \tag{41}$$

Multi-hop reasoning aggregates evidence along paths:

$$p(a|q) = \sum_{\pi \in \Pi(q,a)} p(\pi|q) \cdot p(a|\pi, q) \tag{42}$$

6.2.2. Cross-Domain Knowledge Synthesis

Knowledge from different domains is aligned through learned mappings that preserve semantic relationships:

$$\phi_{AB} : \mathcal{K}_A \to \mathcal{K}_B \tag{43}$$

as illustrated in Figure 5, showing hierarchical organization from raw data through semantic concepts to abstract reasoning, which enables seamless integration from capabilities.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 July 2025 doi:10.20944/preprints202507.2283.v1

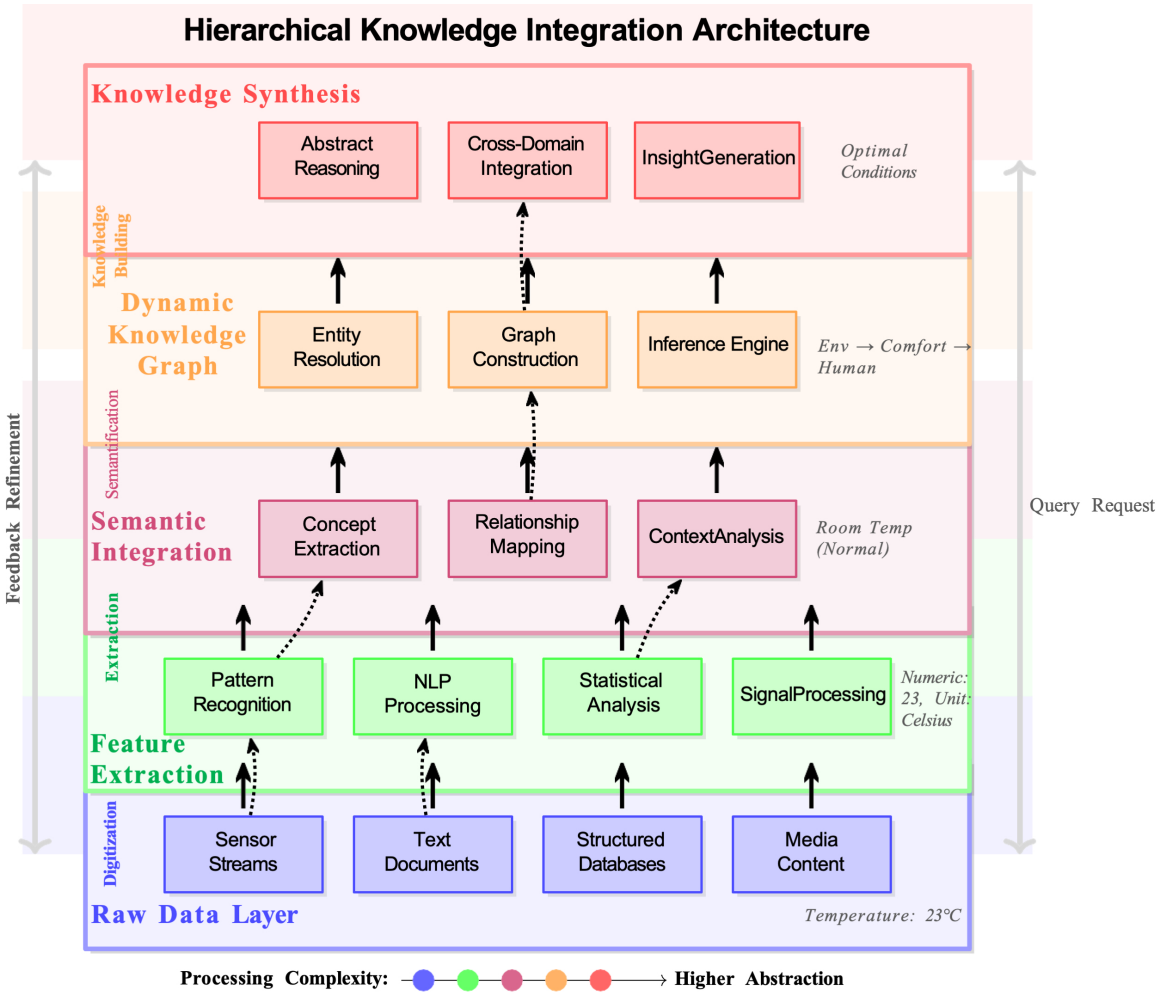

Figure 5. Knowledge Integration Layers showing the hierarchical organization from raw data through semantic concepts to abstract reasoning.

6.3. Distributed Knowledge Management

6.3.1. Federated Knowledge Aggregation

Distributed knowledge nodes collaborate without centralization, extending distributed consensus [50]:

$$\mathcal{K}_{\text{global}} = \bigoplus_{i=1}^{n} w_i \mathcal{K}_i \tag{44}$$

where $\oplus$ represents a learned aggregation operator.

6.3.2. Semantic Memory and Retrieval

The system implements content-addressable memory with temporal context:

$$m_t = f_{\text{encode}}(c_t, m_{t-1}, \tau_t) \tag{45}$$

This enables time-aware reasoning and historical analysis while maintaining semantic coherence.

6.4. Uncertainty Quantification

The framework maintains uncertainty through Bayesian neural networks [53], decomposing total uncertainty into aleatoric and epistemic components:

 doi:10.20944/preprints202507.2283.v1

$$\mathcal{U}_{\text{total}}(x) = \underbrace{\mathcal{H}[\mathbb{E}_\theta[p(y|x,\theta)]]}_{\text{aleatoric}} + \underbrace{\mathbb{E}_\theta[\mathcal{H}[p(y|x,\theta)]] - \mathcal{H}[\mathbb{E}_\theta[p(y|x,\theta)]]}_{\text{epistemic}} \tag{46}$$

6.5. Computational Infrastructure for Knowledge Processing

Liquid AI's knowledge integration requires distributed computing infrastructure extending architectures used in large-scale training [57,58]. The system implements:

Dynamic workload distribution based on node capabilities:

$$\mathcal{W}_i = f_{\text{allocate}}(\mathcal{W}, \mathcal{P}_i, \mathcal{U}, \mathcal{L}) \tag{47}$$

System reliability through redundancy and checkpointing [59]:

$$R(\mathcal{N}) = 1 - \prod_{i=1}^{n} (1 - r_i)^{k_i} \tag{48}$$

Hardware acceleration leveraging TPU [60] and GPU [61] architectures with mixed precision training [62]:

Algorithm 9 Adaptive Load Balancing

Require: Node utilizations $\{\mathcal{U}_i\}$, workloads $\{\mathcal{W}_i\}$, threshold τ
Ensure: Balanced workload distribution
1: **while** $\max_i \mathcal{U}_i - \min_i \mathcal{U}_i > \tau$ **do**
2: $i^* = \arg\max_i \mathcal{U}_i$
3: $j^* = \arg\min_j \mathcal{U}_j$
4: Compute transfer amount: $\Delta = \min(\mathcal{W}_{i^*} \cdot \alpha, \text{capacity}_j - \mathcal{W}_{j^*})$
5: Transfer workload: $\mathcal{W}_{i^*} \leftarrow \mathcal{W}_{i^*} - \Delta$
6: $\mathcal{W}_{j^*} \leftarrow \mathcal{W}_{j^*} + \Delta$
7: Update utilizations based on new workloads
8: **end while**

7. Implementation Considerations

7.1. Computational Complexity Analysis

Implementing Liquid AI presents significant computational challenges requiring careful analysis. Unlike static neural architectures with fixed complexity [54], Liquid AI's dynamic nature introduces variable computational requirements.

7.1.1. Asymptotic Complexity

We analyze the computational complexity of core components extending results from graph neural networks [55] and multi-agent systems [56]:

$$\mathcal{C}_{\text{DKG}} = O(|V|^2 \cdot d + |E| \cdot d + |H| \cdot k \cdot d) \tag{49}$$

$$\mathcal{C}_{\text{SDE}} = O(|\mathcal{T}| \cdot |\mathcal{A}| \cdot \log(|\mathcal{A}|)) \tag{50}$$

$$\mathcal{C}_{\text{MACF}} = O(|\mathcal{A}|^2 \cdot |\mathcal{M}|) \tag{51}$$

Knowledge graph queries can be optimized through algorithmic improvements:

 doi:10.20944/preprints202507.2283.v1

Algorithm 10 Efficient Knowledge Graph Query

Require: Query q, graph $\mathcal{G}$, beam width k
Ensure: Answer set $\mathcal{A}$
Encode query: $\mathbf{q} = \mathcal{E}(q)$
Initialize priority queue: $Q \leftarrow \{(\text{start}, 0)\}$
Initialize visited set: $V \leftarrow \emptyset$
while $|Q| > 0$ and $|\mathcal{A}| < k$ **do**
  $(v, \text{score}) \leftarrow Q.\text{pop\_max}()$
  if $v \in V$ **then**
    continue
  end if
  $V \leftarrow V \cup \{v\}$
  if is_answer(v, q) **then**
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{v\}$
  end if
  for $u \in \text{neighbors}(v)$ **do**
    $s = \text{score} + \text{sim}(\mathbf{q}, \mathbf{u})$
    $Q.\text{push}((u, s))$
  end for
end while
return $\mathcal{A}$

7.2. Distributed Architecture and Resource Management

7.2.1. Hierarchical Processing

The theoretical system organizes into processing layers for efficient computation distribution. Edge layers handle low-latency local processing, fog layers manage regional aggregation and coordination, while cloud layers perform global optimization and heavy computation.

7.2.2. Dynamic Resource Allocation

Resources scale elastically with demand through control-theoretic approaches:

$$r(t+1) = r(t) + k_p e(t) + k_i \int e(\tau) d\tau + k_d \frac{de(t)}{dt} \tag{52}$$

where $e(t) = \text{demand}(t) - \text{capacity}(t)$ represents resource deficit.
Algorithm selection adapts to input characteristics [63]:

$$\mathcal{A}^* = \arg\min_{\mathcal{A} \in \mathcal{F}} \mathbb{E}_{D \sim p(D)}[\mathcal{C}(\mathcal{A}, D)] \tag{53}$$

7.3. Security and Privacy Considerations

Knowledge integration preserves privacy through differential privacy [64]:

$$\mathcal{M}(\mathcal{D}) = f(\mathcal{D}) + \text{Lap}\left(\frac{\Delta f}{\epsilon}\right) \tag{54}$$

Agents collaborate without revealing private data through secure multi-party computation [65]:

$$f(x_1, ..., x_n) = \text{decrypt}\left(\prod_{i=1}^{n} \text{encrypt}(f_i(x_i))\right) \tag{55}$$

The system implements certified defenses against adversarial inputs [66,67]:

$$\|\delta\|_p < \epsilon \Rightarrow f(x + \delta) = f(x) \tag{56}$$

 doi:10.20944/preprints202507.2283.v1

7.4. Deployment and Monitoring

The theoretical framework includes mechanisms for gradual capability deployment with continuous monitoring. System health is tracked through statistical anomaly detection, while debug information includes causal traces for explainable failure analysis. Performance optimization occurs through adaptive batch sizing, dynamic pruning of unnecessary computations, and computation reuse across similar inputs.

Table S5 (Supplemental Material) provides detailed implementation requirements for different deployment scales, addressing the substantial computational infrastructure needed for practical realization of these theoretical concepts.

8. Evaluation Methodology

Evaluating continuously evolving AI systems requires fundamentally different approaches than traditional static benchmarks. We present theoretical methodologies that could capture temporal dynamics, emergent behaviors, and long-term evolution of Liquid AI systems. Our empirical validation methodology, visualized in Figure 6, follows an iterative cycle of system configuration, baseline establishment, performance evaluation, and comparative analysis.

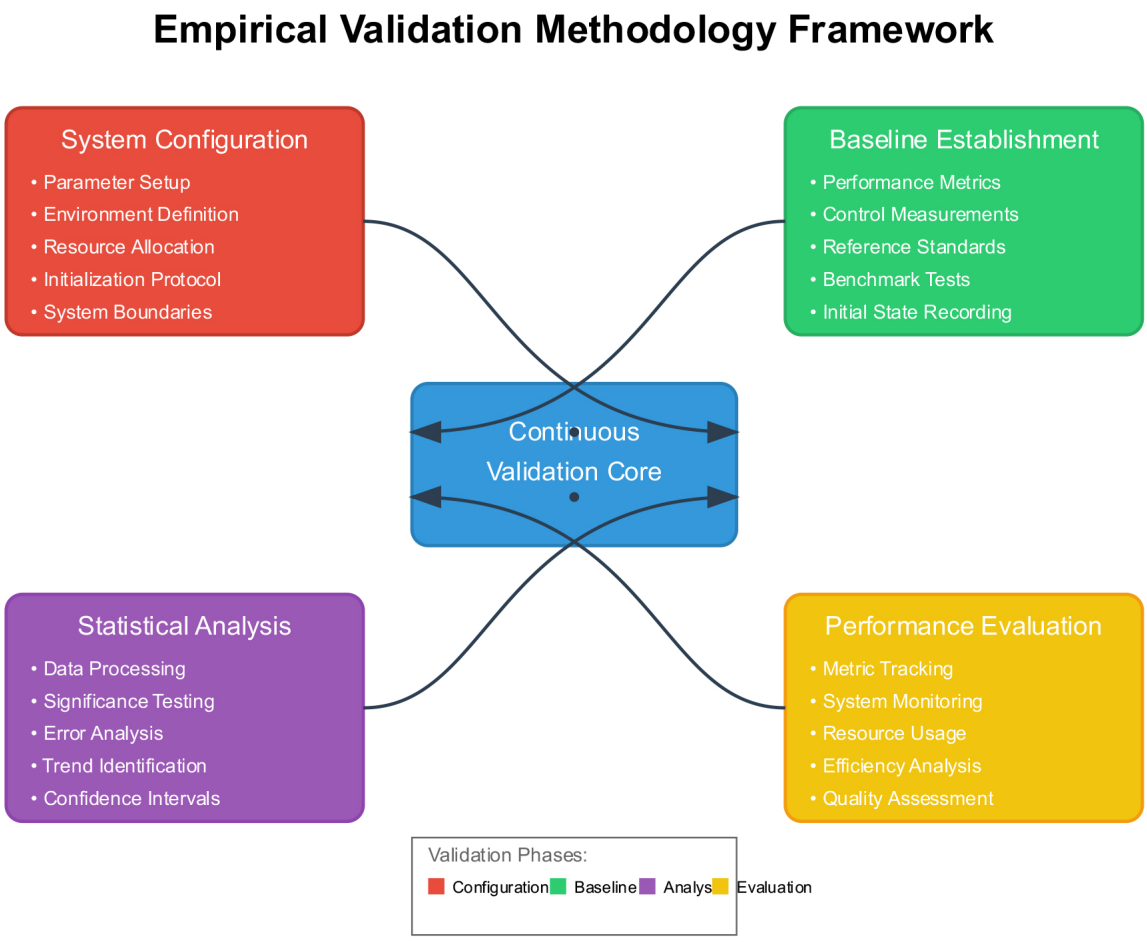

Figure 6. Empirical Validation Methodology Framework showing the iterative cycle of system configuration, baseline establishment, performance evaluation, and comparative analysis.

8.1. Challenges in Evaluating Adaptive Systems

Traditional AI evaluation assumes fixed architectures and capabilities [68,69]. Liquid AI's theoretical continuous evolution introduces unique evaluation challenges. System capabilities would change during evaluation, requiring metrics that account for temporal evolution. Novel capabilities could

 doi:10.20944/preprints202507.2283.v1

emerge unpredictably outside the span of initial capabilities. Evolution would occur across multiple timescales, from rapid parameter updates to slower architectural modifications.

Figure 7 illustrates the proposed iterative validation methodology, showing how system configuration, baseline establishment, performance evaluation, and comparative analysis would form a continuous cycle rather than discrete evaluation points.

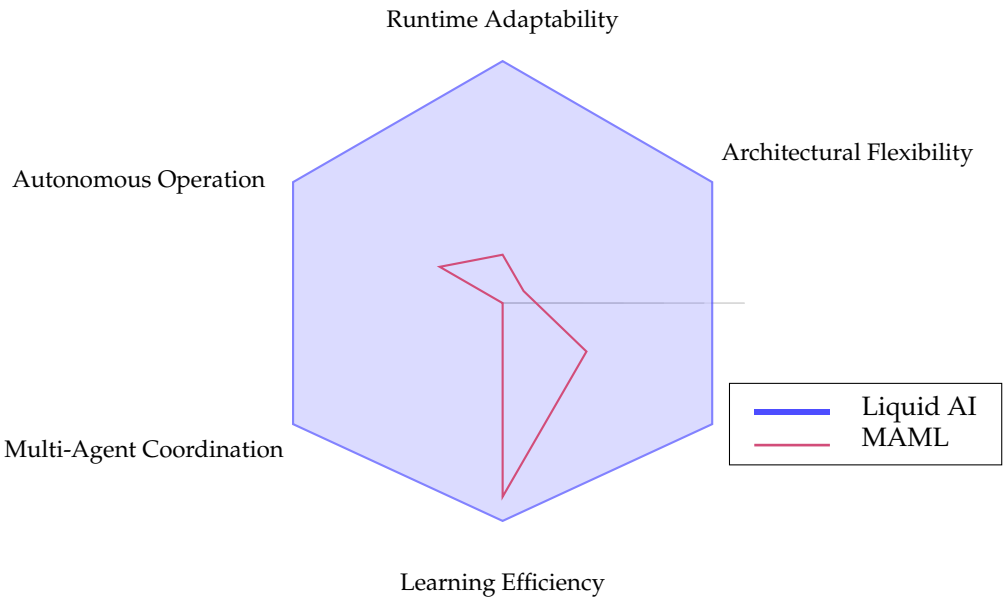

Figure 7. Comparative analysis of adaptive AI systems across six key capability dimensions. Targeted capability dimensions for Liquid AI compared to existing approaches. Values represent theoretical design goals rather than measured performance.

8.2. Temporal Performance Metrics

8.2.1. Capability Evolution Tracking

We propose measuring autonomous improvement through capability growth rate:

$$gc = \frac{d\mathcal{C}}{dt} = \lim_{\Delta t \to 0} \frac{\mathcal{C}(t + \Delta t) - \mathcal{C}(t)}{\Delta t} \tag{57}$$

This would quantify the system's theoretical self-improvement velocity. Adaptation efficiency could be measured as the ratio of performance improvement to computational resources consumed.

8.2.2. Multi-Domain Evaluation

Cross-domain tasks would be constructed to test knowledge integration:

Algorithm 11 Multi-Domain Task Construction

Require: Domains $\{D_i\}$, complexity level c
Ensure: Multi-domain task T
1: Select domains: $\mathcal{D} \sim p(\mathcal{D}|c)$
2: Extract concepts: $\mathcal{K}_i = \text{core\_concepts}(D_i)$ for $D_i \in \mathcal{D}$
3: Identify bridges: $\mathcal{B} = \{(k_i, k_j) : \text{related}(k_i, k_j), k_i \in \mathcal{K}_i, k_j \in \mathcal{K}_j\}$
4: Construct task requiring bridges: $T = f_{\text{task}}(\mathcal{B}, c)$
5: Verify solvability: ensure T requires knowledge from all $D_i \in \mathcal{D}$
6: **return** T

Adaptive learning assessment would track how quickly the system adapts to new domains:

 doi:10.20944/preprints202507.2283.v1

Algorithm 12 Adaptive Learning Assessment

Require: System $\mathcal{S}$, task sequence $\{T_i\}$, evaluation interval Δt
Ensure: Adaptive learning metrics $\mathcal{M}_{\text{adapt}}$
Initialize: $\mathcal{M}_{\text{adapt}} = \{\}$
for $t = 0$ to $T_{\max}$ step Δt **do**
  Sample task: $T \sim p(T|t)$
  Measure performance: $P(t) = \text{evaluate}(\mathcal{S}, T)$
  Compute learning rate: $r(t) = \frac{dP}{dt}$
  Assess transfer: $\tau(t) = P_{\text{new}}(t) - P_{\text{baseline}}$
  Record architecture: $\Lambda(t) = \text{get\_architecture}(\mathcal{S})$
  $\mathcal{M}_{\text{adapt}} \leftarrow \mathcal{M}_{\text{adapt}} \cup \{(t, P(t), r(t), \tau(t), \Lambda(t))\}$
end for
Compute summary statistics: adaptation rate, transfer efficiency, stability
return $\mathcal{M}_{\text{adapt}}$

8.3. Human-AI Interaction Evaluation

Human-in-the-loop evaluation would measure adaptation to feedback:

Algorithm 13 Interactive Adaptation Assessment

Require: System $\mathcal{S}$, human evaluator H, task set $\mathcal{T}$
Ensure: Adaptation metrics $\mathcal{M}_{\text{interact}}$
for $t = 1$ to T **do**
  Present task: $T_t \sim p(T|\text{history})$
  System response: $R_t = \mathcal{S}(T_t)$
  Human feedback: $F_t = H(T_t, R_t)$
  System update: $\mathcal{S} \leftarrow \text{adapt}(\mathcal{S}, F_t)$
  Measure adaptation: $\Delta_t = d(\mathcal{S}_t, \mathcal{S}_{t-1})$
end for
Compute adaptation trajectory and human satisfaction
return $\mathcal{M}_{\text{interact}}$

8.4. Safety and Deployment Validation

Given the theoretical self-modification capabilities, validation would require novel safety protocols. We extend algorithm selection approaches [70] to online settings for continuous validation:

Algorithm 14 Safe Deployment for Self-Modifying Systems

Require: New version v_{new}, current version v_{current}, safety threshold τ
Ensure: Safe deployment decision
Run compatibility tests: $c = \text{test\_compat}(v_{\text{new}}, v_{\text{current}})$
Evaluate in sandbox: $p_{\text{sandbox}} = \text{eval\_sandbox}(v_{\text{new}})$
Compute safety score: $s = \alpha \cdot c + \beta \cdot p_{\text{sandbox}} + \gamma \cdot \text{similarity}(v_{\text{new}}, v_{\text{current}})$
if $s > \tau$ **then**
  Deploy with canary: $\text{deploy\_canary}(v_{\text{new}}, 0.01)$
  Monitor metrics: $m = \text{monitor}(\Delta t)$
  if $m > m_{\text{baseline}}$ **then**
    Gradual rollout: $\text{increase\_traffic}(v_{\text{new}})$
  else
    Rollback: $\text{revert}(v_{\text{current}})$
  end if
else
  Reject deployment
end if

Table S3 (Supplemental Material) summarizes the complete set of evaluation metrics for adaptive AI systems. Figure 10 demonstrates theoretical analysis of sustained capability improvement, showing

 doi:10.20944/preprints202507.2283.v1

performance trajectories, architectural evolution, and feedback mechanism contributions over time. Complete evaluation across all scenarios are compiled in Table S7.

9. Applications and Use Cases

Liquid AI's theoretical ability to continuously evolve and adapt would make it particularly suited for complex, dynamic domains where traditional AI systems struggle. We explore potential applications across healthcare, scientific discovery, and industrial systems, while acknowledging these represent theoretical possibilities rather than immediate implementations.

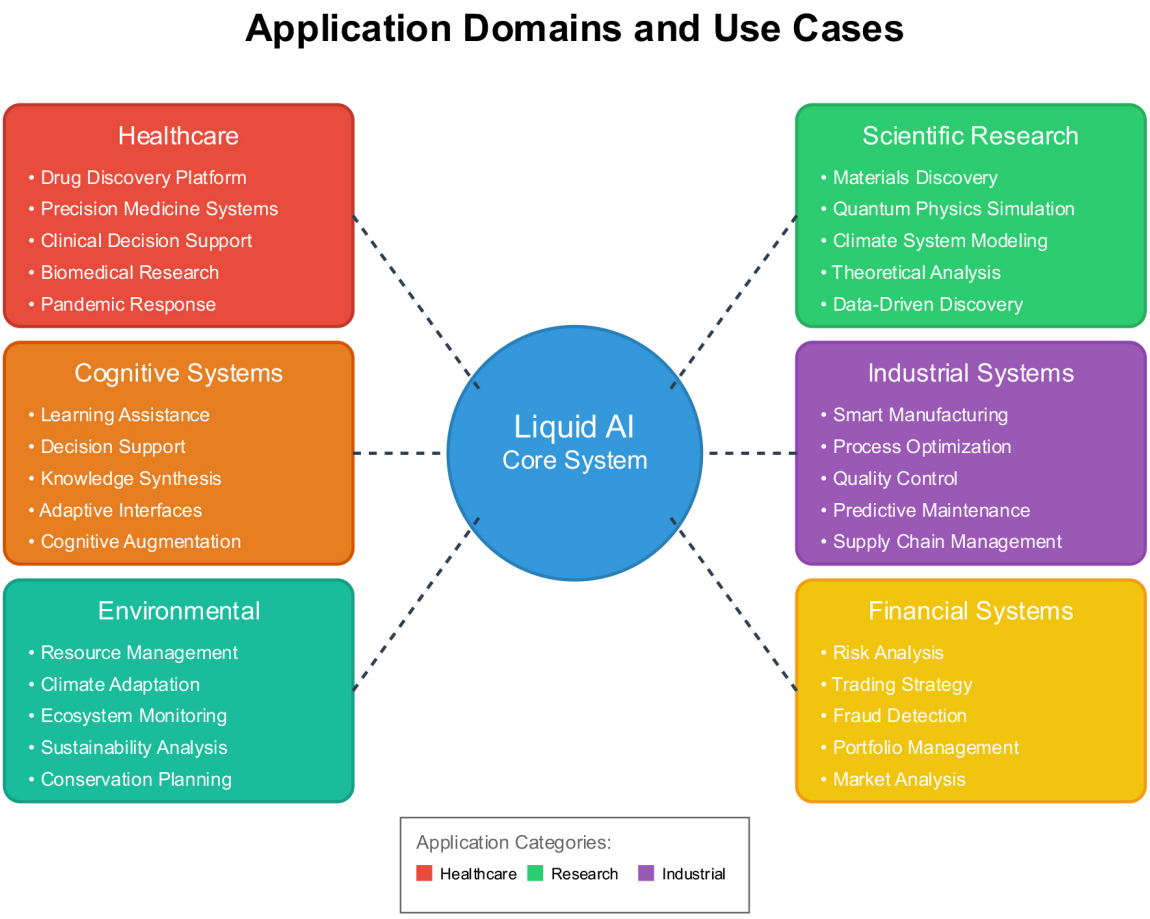

Figure 8. Application Domain and Use Cases of Liquid AI. Systematic representation of six primary application domains: Healthcare (showing drug discovery, precision medicine, clinical analytics), Scientific Research (materials discovery, physics simulation, climate modeling), Industrial Systems (smart manufacturing, quality control, process optimization), Financial Systems (risk analysis, trading strategies, fraud detection), Environmental Management (resource management, climate adaptation, ecosystem monitoring), and Cognitive Systems (learning assistance, decision support, knowledge synthesis).

9.1. Healthcare and Biomedical Applications

In personalized medicine, Liquid AI could theoretically enable treatment optimization that evolves with patient responses [71,72]. The system would continuously refine treatment policies based on individual patient history, medical knowledge, and treatment responses, potentially achieving significant improvements in efficacy while reducing adverse reactions.

For epidemic modeling and response, adaptive capabilities could enable real-time modeling that evolves with emerging data [73]. The system would continuously refine predictions and recommendations as situations develop, integrating genomic surveillance data, mobility patterns, healthcare capacity, and intervention effectiveness. The versatility of the Liquid AI framework across diverse

 doi:10.20944/preprints202507.2283.v1

application domains is illustrated in Figure 8, spanning healthcare, scientific research, industrial systems, financial services, environmental management, and cognitive assistance.

9.2. Scientific Discovery

In materials science, Liquid AI could accelerate discovery through integrated modeling and synthesis [74]. The theoretical framework would enable inverse design capabilities through continuous optimization, multi-fidelity modeling integrating theory, simulation, and experiment, and autonomous experimental planning and execution.

For Earth systems modeling, the framework's multi-scale integration capabilities could enhance climate science [75]. Potential improvements include better regional climate predictions through adaptive model refinement, seamless integration across scales from molecular to global processes, and adaptive scenario analysis with evolving uncertainty quantification.

9.3. Industrial and Infrastructure Systems

Adaptive manufacturing systems could continuously optimize production [76]. The theoretical framework would enable real-time adaptation to supply chain disruptions, predictive maintenance with evolving models, quality optimization through continuous learning, and significant energy efficiency improvements.

In energy grid management, adaptive control of complex energy systems could improve reliability and efficiency [77]. The system could learn to integrate renewable sources optimally, predict demand patterns with increasing accuracy, coordinate distributed energy resources, and maintain grid stability during transitions. Performance metrics estimates for specific application domains, including quantitative improvements and resource utilization, are detailed in Table S8.

Figure 8 illustrates the systematic representation of these application domains, showing how Liquid AI's adaptive capabilities could theoretically benefit diverse fields. Table S4 (Supplemental Materials) demonstrates quantitative performance improvements projected across these application domains based on theoretical analysis.

10. Future Directions and Implications

The Liquid AI paradigm opens theoretical possibilities for artificial intelligence while raising important questions about the nature of intelligence, consciousness, and human-AI collaboration. We explore implications, technical challenges, and potential societal impacts of self-evolving AI systems. Figure 9 outlines a comprehensive research roadmap, identifying six key areas requiring further investigation: theoretical foundations, technical implementation challenges, safety and ethics considerations, societal impact assessment, interdisciplinary studies, and emerging applications.

 doi:10.20944/preprints202507.2283.v1

Figure 9. Future Research Directions and Implications. Comprehensive roadmap showing six key research areas arranged along a temporal progression: Theoretical Foundations (mathematical frameworks, convergence properties), Technical Implementation (scalable architecture, resource management), Safety and Ethics (value alignment, governance), Societal Impact (economic effects, policy frameworks), Interdisciplinary Studies (cognitive science, complex systems), and Future Applications (emerging technologies, new domains).

10.1. Philosophical and Theoretical Implications

Liquid AI challenges traditional boundaries between programmed and emergent intelligence. As systems theoretically develop capabilities beyond their initial design, questions arise about the nature of machine consciousness and intentionality [78,79]. Self-modifying systems could exhibit degrees of autonomy previously theoretical, with self-directed decisions and internally generated goals.

The framework extends concepts of distributed cognition and the extended mind hypothesis [80], where the boundary between human and artificial cognition becomes increasingly fluid. This raises fundamental questions about agency, responsibility, and the nature of intelligence itself.

10.2. Technical Research Challenges

Several critical technical challenges require resolution before practical implementation becomes feasible. Understanding fundamental scaling constraints involves examining thermodynamic limits of computation, potential quantum advantages for adaptive systems, and distributed intelligence scaling laws. The computational requirements align with current trends toward nuclear-powered data centers, as detailed in Supplemental Materials Section 3.

Ensuring correctness in self-modifying systems presents unique verification challenges. Runtime verification of evolved architectures, formal methods for adaptive systems, and probabilistic correctness guarantees all require novel theoretical frameworks. Maintaining interpretability as complexity grows necessitates new approaches for hierarchical explanation generation, causal reasoning in evolved systems, and human-comprehensible abstractions.

 doi:10.20944/preprints202507.2283.v1

10.3. Societal Considerations

The potential societal impact of self-evolving AI systems warrants careful consideration. Economic restructuring could result from widespread deployment of adaptive AI, affecting labor markets, productivity patterns, and wealth distribution. New governance frameworks would be needed for managing self-evolving systems, requiring regulatory adaptation, international coordination, and democratic participation in AI governance.

Educational transformation would be necessary to prepare humanity for collaboration with adaptive AI systems. This includes developing human-AI collaboration skills, ethical reasoning capabilities, and continuous learning mindsets. Table 1 outlines key ethical considerations and corresponding governance mechanisms necessary for responsible Liquid AI deployment.

Table 2. Ethical Considerations for Liquid AI Deployment.

Aspect	Challenge	Mitigation Strategy	Research Needs
Autonomy	Self-modification may lead to unintended behaviors	Bounded modification spaces, continuous monitoring	Formal verification methods for dynamic systems
Transparency	Evolving architectures complicate interpretability	Maintain modification logs, interpretable components	Dynamic explanation generation techniques
Accountability	Unclear responsibility for emergent decisions	Clear governance frameworks, audit trails	Legal frameworks for autonomous AI
Fairness	Potential for bias amplification	Active bias detection and mitigation	Fairness metrics for evolving systems
Privacy	Distributed knowledge may leak sensitive information	Differential privacy, secure computation	Privacy-preserving knowledge integration
Safety	Unpredictable emergent behaviors	Conservative modification bounds, rollback mechanisms	Safety verification for self-modifying systems
Control	Difficulty in stopping runaway evolution	Multiple kill switches, consensus requirements	Robust control mechanisms

10.4. Long-Term Research Trajectories

Future research directions span multiple disciplines and timescales. Immediate priorities include developing incremental implementation pathways, establishing safety protocols for self-modifying systems, and creating evaluation frameworks for adaptive AI. Medium-term goals involve exploring hybrid human-AI systems, investigating emergent collective intelligence, and developing interpretability methods for evolved architectures.

Long-term considerations include understanding potential intelligence explosion scenarios, exploring post-biological intelligence possibilities, and preparing for human-AI integration pathways. These trajectories point toward a future where boundaries between artificial, biological, and quantum intelligence become increasingly fluid, with Liquid AI providing a theoretical framework for their integration. Figure 10 provides a detailed analysis of sustained capability improvement, showing (a) the three-phase performance trajectory, (b) controlled architectural complexity evolution, and (c) the dynamic contribution of different feedback mechanisms over time.

 doi:10.20944/preprints202507.2283.v1

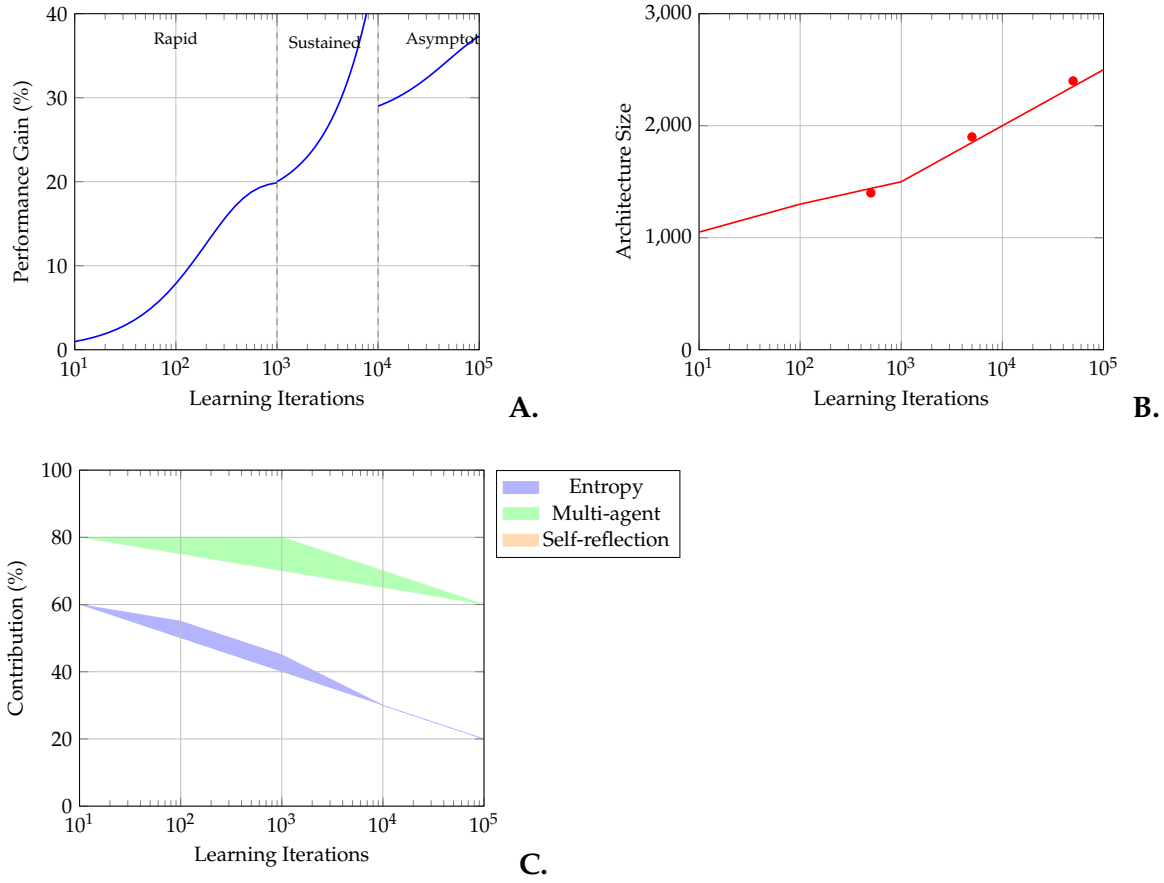

Figure 10. Theoretical analysis of sustained capability improvement in Liquid AI. (a) Performance improvement trajectory showing three distinct phases: rapid initial learning (0-10^3 iterations), sustained improvement (10^3-10^4), and asymptotic convergence (>10^4). (b) Architectural complexity evolution demonstrating controlled growth with periodic efficiency optimizations. (c) Relative contributions of three primary feedback mechanisms, showing the shift from entropy-driven exploration to balanced multi-mechanism optimization.

Figure 9 presents a comprehensive roadmap showing six key research areas arranged along temporal progression, from immediate theoretical foundations to long-term applications and implications.

11. Conclusions

This paper has presented Liquid AI as a theoretical framework and mathematical thought experiment for continuously self-improving artificial intelligence systems. By exploring what becomes possible when we remove traditional architectural constraints, we have established foundations for a potential new paradigm in AI research.

Our work makes several contributions to the theoretical foundations of artificial intelligence. We introduced a comprehensive mathematical framework for AI systems theoretically capable of runtime topological modification, information-theoretic autonomous restructuring, and emergent multi-agent specialization. We established convergence bounds and stability conditions for self-modifying systems, providing theoretical guarantees under which architectural evolution could preserve functional coherence while enabling capability expansion.

The evaluation methodologies developed address the unique challenges of assessing continuously evolving systems, moving beyond static benchmarks to capture temporal dynamics and emergent behaviors. We explored potential applications across healthcare, scientific discovery, and industrial systems, demonstrating how adaptive AI could theoretically transform these fields while acknowledging the significant implementation challenges.

Liquid AI represents a long-term research direction rather than an immediate implementation target. The computational requirements are substantial, likely requiring infrastructure on the scale

 doi:10.20944/preprints202507.2283.v1

of nuclear-powered data centers as industry trends suggest. The stability challenges inherent in self-modifying systems require careful theoretical development and extensive safety research before practical deployment becomes feasible.

Nevertheless, this theoretical exploration opens important new directions for AI research. By establishing mathematical foundations for continuously adaptive AI, we aim to inspire research toward systems that could eventually match the flexibility of biological intelligence while leveraging computational advantages of artificial systems. The comprehensive supplemental materials provide additional technical depth, implementation considerations, and a decade-spanning development roadmap for researchers interested in pursuing these concepts.

The journey from static to liquid intelligence represents a fundamental transition in how we conceive of artificial intelligence. While immediate implementation faces significant challenges, the theoretical framework presented here provides a foundation for future research toward truly adaptive, self-improving AI systems. We invite the research community to join in advancing this vision, whether through theoretical development, incremental implementation strategies, or critical analysis of the concepts presented.

Supplementary Materials: The following supporting information can be downloaded at the website of this paper posted on Preprints.org.

Author Contributions: Conceptualization, T.R.C.; methodology, T.R.C.; formal analysis, T.R.C.; investigation, T.R.C.; writing—original draft preparation, T.R.C.; writing—review and editing, T.R.C., N.N.I., R.C.; visualization, T.R.C. and N.N.I.; supervision, T.R.C.; project administration, T.R.C. All authors have read and agreed to the published version of the manuscript.

References

1. LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436-444.
2. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT Press.
3. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. Advances in Neural Information Processing Systems, 30, 5998-6008.
4. Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT Press.
5. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. Advances in Neural Information Processing Systems, 33, 1877-1901.
6. Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., ... & Fiedel, N. (2023). PaLM: Scaling language modeling with pathways. Journal of Machine Learning Research, 24(240), 1-113.
7. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. IEEE Conference on Computer Vision and Pattern Recognition, 770-778.
8. Howard, J., & Ruder, S. (2018). Universal language model fine-tuning for text classification. Annual Meeting of the Association for Computational Linguistics, 328-339.
9. Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., ... & Chen, W. (2022). LoRA: Low-rank adaptation of large language models. International Conference on Learning Representations.
10. Holtmaat, A., & Svoboda, K. (2009). Experience-dependent structural synaptic plasticity in the mammalian brain. Nature Reviews Neuroscience, 10(9), 647-658.
11. Ruder, S. (2017). An overview of multi-task learning in deep neural networks. arXiv preprint arXiv:1706.05098.
12. Crawshaw, M. (2020). Multi-task learning with deep neural networks: A survey. arXiv preprint arXiv:2009.09796.
13. Hutter, F., Kotthoff, L., & Vanschoren, J. (2019). Automated machine learning: Methods, systems, challenges. Springer.
14. Elsken, T., Metzen, J. H., & Hutter, F. (2019). Neural architecture search: A survey. Journal of Machine Learning Research, 20(1), 1997-2017.
15. Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., ... & Hadsell, R. (2017). Overcoming catastrophic forgetting in neural networks. Proceedings of the National Academy of Sciences, 114(13), 3521-3526.

 doi:10.20944/preprints202507.2283.v1

16. Parisi, G. I., Kemker, R., Part, J. L., Kanan, C., & Wermter, S. (2019). Continual lifelong learning with neural networks: A review. Neural Networks, 113, 54-71.
17. Finn, C., Abbeel, P., & Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. International Conference on Machine Learning, 1126-1135.
18. Nichol, A., Achiam, J., & Schulman, J. (2018). On first-order meta-learning algorithms. arXiv preprint arXiv:1803.02999.
19. Draganski, B., Gaser, C., Busch, V., Schuierer, G., Bogdahn, U., & May, A. (2004). Neuroplasticity: Changes in grey matter induced by training. Nature, 427(6972), 311-312.
20. Bonabeau, E., Dorigo, M., & Theraulaz, G. (1999). Swarm intelligence: From natural to artificial systems. Oxford University Press.
21. Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., ... & Hadsell, R. (2016). Progressive neural networks. arXiv preprint arXiv:1606.04671.
22. Kazemi, S. M., Goel, R., Jain, K., Kobyzev, I., Sethi, A., Forsyth, P., & Poupart, P. (2020). Representation learning for dynamic graphs: A survey. Journal of Machine Learning Research, 21(1), 2648-2720.
23. Andreas, J., Rohrbach, M., Darrell, T., & Klein, D. (2016). Neural module networks. IEEE Conference on Computer Vision and Pattern Recognition, 39-48.
24. Ji, S., Pan, S., Cambria, E., Marttinen, P., & Philip, S. Y. (2021). A survey on knowledge graphs: Representation, acquisition, and applications. IEEE Transactions on Neural Networks and Learning Systems, 33(2), 494-514.
25. White, C., Nolen, S., & Savani, Y. (2021). Exploring the loss landscape in neural architecture search. International Conference on Machine Learning, 10962-10973.
26. Ru, B., Wan, X., Dong, X., & Osborne, M. (2021). Interpretable neural architecture search via Bayesian optimisation with Weisfeiler-Lehman kernels. International Conference on Learning Representations.
27. Zhang, K., Yang, Z., & Başar, T. (2021). Multi-agent reinforcement learning: A selective overview of theories and algorithms. Handbook of Reinforcement Learning and Control, 321-384.
28. Gronauer, S., & Diepold, K. (2022). Multi-agent deep reinforcement learning: A survey. Artificial Intelligence Review, 55(2), 895-943.
29. Laird, J. E., Lebiere, C., & Rosenbloom, P. S. (2019). A standard model of the mind: Toward a common computational framework across artificial intelligence, cognitive science, neuroscience, and robotics. AI Magazine, 40(4), 13-26.
30. Kotseruba, I., & Tsotsos, J. K. (2020). 40 years of cognitive architectures: Core cognitive abilities and practical applications. Artificial Intelligence Review, 53(1), 17-94.
31. Schmidhuber, J. (2007). Gödel machines: Fully self-referential optimal universal self-improvers. Artificial General Intelligence, 199-226.
32. Jia, X., De Brabandere, B., Tuytelaars, T., & Gool, L. V. (2020). Dynamic filter networks. Advances in Neural Information Processing Systems, 29, 667-675.
33. Friston, K. (2017). Active inference and artificial curiosity. arXiv preprint arXiv:1709.07470.
34. Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., & Mané, D. (2016). Concrete problems in AI safety. arXiv preprint arXiv:1606.06565.
35. Hendrycks, D., Carlini, N., Schulman, J., & Steinhardt, J. (2021). Unsolved problems in ML safety. arXiv preprint arXiv:2109.13916.
36. He, X., Zhao, K., & Chu, X. (2021). AutoML: A survey of the state-of-the-art. Knowledge-Based Systems, 212, 106622.
37. Karmaker, S. K., Hassan, M. M., Smith, M. J., Xu, L., Zhai, C., & Veeramachaneni, K. (2021). AutoML to date and beyond: Challenges and opportunities. ACM Computing Surveys, 54(8), 1-36.
38. Lake, B. M., Ullman, T. D., Tenenbaum, J. B., & Gershman, S. J. (2017). Building machines that learn and think like people. Behavioral and Brain Sciences, 40, E253.
39. Pateria, S., Subagdja, B., Tan, A. H., & Quek, C. (2021). Hierarchical reinforcement learning: A comprehensive survey. ACM Computing Surveys, 54(5), 1-35.
40. Franceschi, L., Frasconi, P., Salzo, S., Grazzi, R., & Pontil, M. (2018). Bilevel programming for hyperparameter optimization and meta-learning. International Conference on Machine Learning, 1568-1577.
41. Lorraine, J., Vicol, P., & Duvenaud, D. (2020). Optimizing millions of hyperparameters by implicit differentiation. International Conference on Artificial Intelligence and Statistics, 1540-1552.
42. Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P., & Mordatch, I. (2017). Multi-agent actor-critic for mixed cooperative-competitive environments. Advances in Neural Information Processing Systems, 30, 6379-6390.

 doi:10.20944/preprints202507.2283.v1

43. Shoham, Y., & Leyton-Brown, K. (2008). Multiagent systems: Algorithmic, game-theoretic, and logical foundations. Cambridge University Press.
44. Boyd, S., Parikh, N., Chu, E., Peleato, B., & Eckstein, J. (2011). Distributed optimization and statistical learning via the alternating direction method of multipliers. Foundations and Trends in Machine Learning, 3(1), 1-122.
45. Kennedy, J., & Eberhart, R. (2001). Swarm intelligence. Morgan Kaufmann.
46. Chalkiadakis, G., Elkind, E., & Wooldridge, M. (2022). Computational aspects of cooperative game theory. Synthesis Lectures on Artificial Intelligence and Machine Learning, 16(2), 1-219.
47. Olfati-Saber, R., Fax, J. A., & Murray, R. M. (2007). Consensus and cooperation in networked multi-agent systems. Proceedings of the IEEE, 95(1), 215-233.
48. Hogan, A., Blomqvist, E., Cochez, M., d'Amato, C., Melo, G. D., Gutierrez, C., ... & Zimmermann, A. (2021). Knowledge graphs. ACM Computing Surveys, 54(4), 1-37.
49. Wang, Q., Mao, Z., Wang, B., & Guo, L. (2017). Knowledge graph embedding: A survey of approaches and applications. IEEE Transactions on Knowledge and Data Engineering, 29(12), 2724-2743.
50. Castro, M., & Liskov, B. (1999). Practical Byzantine fault tolerance. Proceedings of the Third Symposium on Operating Systems Design and Implementation, 173-186.
51. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Philip, S. Y. (2021). A comprehensive survey on graph neural networks. IEEE Transactions on Neural Networks and Learning Systems, 32(1), 4-24.
52. Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., ... & Sun, M. (2020). Graph neural networks: A review of methods and applications. AI Open, 1, 57-81.
53. Gal, Y., & Ghahramani, Z. (2016). Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. International Conference on Machine Learning, 1050-1059.
54. Strubell, E., Ganesh, A., & McCallum, A. (2019). Energy and policy considerations for deep learning in NLP. Annual Meeting of the Association for Computational Linguistics, 3645-3650.
55. Xu, K., Hu, W., Leskovec, J., & Jegelka, S. (2019). How powerful are graph neural networks? International Conference on Learning Representations.
56. Hernández-Orallo, J., Martínez-Plumed, F., Schmid, U., Siebers, M., & Dowe, D. L. (2016). Computer models solving intelligence test problems: Progress and implications. Artificial Intelligence, 230, 74-107.
57. Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., & Catanzaro, B. (2019). Megatron-LM: Training multi-billion parameter language models using model parallelism. arXiv preprint arXiv:1909.08053.
58. Rajbhandari, S., Rasley, J., Ruwase, O., & He, Y. (2020). ZeRO: Memory optimizations toward training trillion parameter models. International Conference for High Performance Computing, Networking, Storage and Analysis, 1-16.
59. Chen, C., Wang, K., & Wu, Q. (2015). Efficient checkpointing and recovery for distributed systems. IEEE Transactions on Parallel and Distributed Systems, 26(12), 3301-3313.
60. Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., ... & Yoon, D. H. (2017). In-datacenter performance analysis of a tensor processing unit. ACM/IEEE International Symposium on Computer Architecture, 1-12.
61. NVIDIA. (2020). NVIDIA A100 tensor core GPU architecture. NVIDIA Technical Report.
62. Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., ... & Wu, H. (2018). Mixed precision training. International Conference on Learning Representations.
63. Kotthoff, L. (2016). Algorithm selection for combinatorial search problems: A survey. Data Mining and Constraint Programming, 149-190.
64. Dwork, C., & Roth, A. (2014). The algorithmic foundations of differential privacy. Foundations and Trends in Theoretical Computer Science, 9(3-4), 211-407.
65. Evans, D., Kolesnikov, V., & Rosulek, M. (2018). A pragmatic introduction to secure multi-party computation. Foundations and Trends in Privacy and Security, 2(2-3), 70-246.
66. Cohen, J., Rosenfeld, E., & Kolter, Z. (2019). Certified adversarial robustness via randomized smoothing. International Conference on Machine Learning, 1310-1320.
67. Carlini, N., & Wagner, D. (2017). Towards evaluating the robustness of neural networks. IEEE Symposium on Security and Privacy, 39-57.
68. Hernández-Orallo, J. (2017). Evaluation in artificial intelligence: From task-oriented to ability-oriented measurement. Artificial Intelligence Review, 48(3), 397-447.
69. Chollet, F. (2019). On the measure of intelligence. arXiv preprint arXiv:1911.01547.

 doi:10.20944/preprints202507.2283.v1

70. Kerschke, P., Hoos, H. H., Neumann, F., & Trautmann, H. (2019). Automated algorithm selection: Survey and perspectives. Evolutionary Computation, 27(1), 3-45.
71. Vamathevan, J., Clark, D., Czodrowski, P., Dunham, I., Ferran, E., Lee, G., ... & Zhao, S. (2019). Applications of machine learning in drug discovery and development. Nature Reviews Drug Discovery, 18(6), 463-477.
72. Johnson, K. B., Wei, W. Q., Weeraratne, D., Frisse, M. E., Misulis, K., Rhee, K., ... & Snowdon, J. L. (2021). Precision medicine, AI, and the future of personalized health care. Clinical and Translational Science, 14(1), 86-93.
73. Alamo, T., Reina, D. G., Mammarella, M., & Abella, A. (2020). Covid-19: Open-data resources for monitoring, modeling, and forecasting the epidemic. Electronics, 9(5), 827.
74. Himanen, L., Geurts, A., Foster, A. S., & Rinke, P. (2019). Data-driven materials science: Status, challenges, and perspectives. Advanced Science, 6(21), 1900808.
75. Reichstein, M., Camps-Valls, G., Stevens, B., Jung, M., Denzler, J., & Carvalhais, N. (2019). Deep learning and process understanding for data-driven Earth system science. Nature, 566(7743), 195-204.
76. Zhong, R. Y., Xu, X., Klotz, E., & Newman, S. T. (2017). Intelligent manufacturing in the context of Industry 4.0: A review. Engineering, 3(5), 616-630.
77. Zhang, Q., Li, H., & Liao, Y. (2018). Smart grid: A review of recent developments and future challenges. International Journal of Electrical Power & Energy Systems, 103, 481-490.
78. Dreyfus, H. L. (1992). What computers still can't do: A critique of artificial reason. MIT Press.
79. Chalmers, D. (2010). The singularity: A philosophical analysis. Journal of Consciousness Studies, 17(9-10), 7-65.
80. Clark, A. (2008). Supersizing the mind: Embodiment, action, and cognitive extension. Oxford University Press.
81. Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., ... & Hadsell, R. (2017). Overcoming catastrophic forgetting in neural networks. Proceedings of the National Academy of Sciences, 114(13), 3521-3526.
82. Liu, H., Simonyan, K., & Yang, Y. (2019). DARTS: Differentiable architecture search. International Conference on Learning Representations.
83. Rashid, T., Samvelyan, M., Schroeder, C., Farquhar, G., Foerster, J., & Whiteson, S. (2018). QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning. International Conference on Machine Learning, 4295-4304.