

Rhodes College Digital Archives - DLynx

Exploring the Criteria for Artificial Consciousness: Implementing and Evaluating Agentic Software Systems in Virtual Environments

Item Type	Thesis
Authors	Le, Bach
Publisher	Memphis, Tenn. : Rhodes College
Rights	<p>Rhodes College owns the rights to the archival digital images in this collection. Images are made available for educational use only and may not be used for any non-educational or commercial purpose. Approved educational uses include private research and scholarship, teaching, and student projects. Original copies of the programs are stored in the Rhodes College Archives. In all instances of use, acknowledgement must be given to Rhodes College Archives Digital Repository, Memphis, TN. For information regarding permission to use this image, please email the Archives at archives@rhodes.edu</p>
Download date	2026-06-08 02:04:04
Link to Item	https://hdl.handle.net/10267/37259

I give permission for public access to my Honors paper and for any copying or digitization to be done at the discretion of the College Archivist and/or the College Librarian.

Signed Bach

[Name typed] BACH LE

Date May 6th 2026

Exploring the Criteria for Artificial Consciousness: Implementing
and Evaluating Agentic Software Systems in Virtual Environments

Bach Le

Department of Computer Science

Submitted in partial fulfillment of the requirements for
the Bachelor of Science degree with Honors in Computer Science

Rhodes College

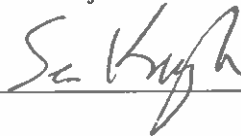
May 5, 2026

Signature Page

This Honors paper by **Bach Le** has been read and approved for Honors in **Computer Science**.

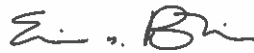
Dr. Sean Kugele

Project Advisor



Dr. Erin Bodine

Second Reader



Dr. Addison Schwamb

Third Reader



Dr. D. Brian Larkins

Department Chair



Contents

1	Introduction	2
2	What are indicator properties (IPs)?	2
2.1	Recurrent Processing Theory	3
2.2	Global Workspace Theory	4
2.3	Higher-Order Theories	5
2.4	Attention Schema Theory	6
2.5	Predictive Processing	6
2.6	Agency and Embodiment	6
3	Evaluation of Recent Algorithms	7
3.1	Algorithms	7
3.1.1	A2C	7
3.1.2	TRPO	8
3.1.3	PPO	9
3.1.4	DreamerV3	9
3.2	Indicator Properties Satisfied	10
3.3	Performance Comparison	11
3.3.1	Environments	11
3.3.2	Rewards	11
3.3.3	Observations	12
3.3.4	Actions	12
3.3.5	Results	12
4	Bottom-up construction of a conscious system	13
4.1	Environments	14
4.1.1	Rewards	14
4.1.2	Observation	15
4.1.3	Action	15
4.1.4	Custom Environments	15
4.2	Based model and features added	16
4.3	Indicator Properties Satisfied	19
5	Top-down design of a conscious system	19
5.1	LIDA	20
5.1.1	LIDA's cognitive cycle	20
5.1.2	Sensory Memory & Perceptual Associative Memory (PAM)	21
5.1.3	Global Workspace & Current Situational Model	22
5.1.4	Codelets	22
5.1.5	Procedural Memory & Action Selection	23
5.2	Indicator Properties Satisfied	23
6	Conclusion	26
	Appendices	28
A	Hyperparameters	28
A.1	A2C	28
A.2	TRPO	29
A.3	PPO	30

List of Tables

1	Indicator Properties and Algorithms.	13
---	----------------------------------------------	----

ABSTRACT

Exploring the Criteria for Artificial Consciousness: Implementing and
Evaluating Agentic Software Systems in Virtual Environments

by

Bach Hoang Chi Le

This project explores the possibility of consciousness in software agents and recent attempts to make the search for artificial consciousness more rigorous and empirically grounded. Starting from a set of “indicator properties” established as markers of consciousness in AI systems [2], we (1) analyze state-of-the-art agent-based systems to determine the properties they satisfy, (2) examine their behaviors to determine if additional indicator properties have a clear effect on the systems’ behaviors (are they more human-like, or more efficient) (3) attempt to design a system top-down, and (4) attempt to build a system bottom-up based on indicator properties. There is a significant gap between the proposed theories and the realization in computing systems, making it very difficult for us to evaluate current algorithms and to build an agent that satisfies all indicator properties. While we believe the framework could be useful, we find it needs more support from current literature to make evaluating consciousness feasible.

1 Introduction

Today’s frontier models can surpass current tests designed to assess human-like behaviors and capabilities, such as the Turing Test. They are able to deceive humans into believing that they are conscious or are another human. However, instead of showing signs of consciousness, it raises the concern of outdated tests and the need for new, capable evaluations. In response to this need, [2] proposed markers (indicator properties) such that an agent possessing all indicators is a candidate for artificial consciousness. In this way, we have a framework to evaluate AI systems. However, the conclusion from the paper is that no current system possesses all indicator properties, and all are unconscious [2]. Nonetheless, they suggest that the implementation of such a system is plausible. We will attempt to review current algorithms according to the criteria, design, and build a system with as many indicator properties as possible.

Beyond theoretical curiosity, the ability to determine whether an artificial system could be conscious holds substantial ethical and societal implications. If a system were to achieve genuine consciousness, it would raise unprecedented moral concerns regarding its capacity for subjective experience and potential suffering, as well as the ethical obligations humans may have toward it. Conversely, the misattribution of consciousness to systems that merely simulate human-like behaviors poses a distinct set of risks. Due to the human tendency toward anthropomorphism (the projection of human traits and emotions onto non-human entities), individuals may overestimate an AI’s understanding, agency, or empathy. Such misperception can lead to inappropriate trust, emotional dependency, or misplaced moral responsibility. Consequently, establishing rigorous criteria to evaluate consciousness in artificial agents is not merely a philosophical pursuit but a practical necessity for ensuring ethical integrity, psychological safety, and societal trust in emerging technologies.

2 What are indicator properties (IPs)?

Indicator properties are functional or computational characteristics derived from leading neuroscientific theories of consciousness, proposed as a rubric for evaluating whether an artificial system could be conscious [2]. Rather than relying on behavioral tests, which can be deceived by systems trained to mimic human responses, indicator properties focus on the underlying mechanisms and functional organization associated with conscious experience in humans. These properties are extracted from well-supported scientific theories that identify functions deemed necessary or sufficient for consciousness in the human brain. These theories include, but are not limited to Global Workspace Theory, Recurrent Processing Theory, Higher Order

Theory, and Attention Schema Theory.

The idea of an Indicator Property relies heavily on the assumption of computational functionalism. Computational functionalism is the view that consciousness arises from implementing the right kinds of computations or architectures, regardless of the physical medium. It holds that what matters is the functional organization and information processing, not the biological substance. Under this view, if an artificial system replicates the computational structures and causal dynamics associated with human consciousness, it could likewise be considered conscious. This assumption underpins the use of indicator properties, which assess whether an AI system implements the key computational functions linked to conscious experience in the brain.

While no architecture has been demonstrated to possess all the properties, the framework theoretically allows researchers to assess the computational capabilities of an agent compared to known neuroscience theories. The presence of more indicator properties in a system increases the plausibility of being considered a candidate for consciousness [2]. Nonetheless, it is critical to acknowledge that many of the theories introduced remain highly abstract, lacking robust computational design or concrete architectural specifications. This theoretical nature complicates the practical evaluation of existing frameworks and presents significant challenges in designing or engineering a computational agent that adheres to these IPs.

2.1 Recurrent Processing Theory

The Recurrent Processing Theory (RPT) [12] is a prominent neuroscientific theory of consciousness that distinguishes between unconscious and conscious states based on distinct stages of visual processing [2]. It proposes that consciousness exists when a sufficient specific type of activity occurs in a relatively contained area of the brain — the visual system. The theory is supported by evidence showing that while initial, feed-forward processing is sufficient for basic functions, consciousness requires a recurrent processing loop, where signals are sent back and forth between different visual brain regions to create an organized, integrated scene.

Based on RPT, the two indicator properties for consciousness are:

1. **RPT-1:** Input modules using algorithmic recurrence.
2. **RPT-2:** Input modules generating organized, integrated perceptual representations. This property indicates that the system is not only processing information but is also using the recurrent processing

to create a coherent and integrated representation of the input.

2.2 Global Workspace Theory

Global Workspace Theory (GWT) [1] is a leading theory of consciousness that proposes humans use a central “global workspace” to integrate information from multiple specialized, parallel-processes systems or “modules.” This integration allows the system to coordinate and flexibly use information, enhancing its overall capabilities. The theory claims that a state is conscious after its representation enters the global workspace and is “globally broadcast” to all processes and modules [2].

GWT attempts to explain why only a select subset of information is available at any given time for higher-level functions like reasoning, decision-making, and recall. For example, some perceptual representations may become more salient due to an individual’s current goals, causing them to “win the contest” for entry into the limited-capacity global workspace. This allows the information to be broadcast across different modules and processes.

Based on GWT, four key indicator properties can be used to determine if an artificial system has a consciousness-like architecture.

1. **GWT-1:** Multiple specialized systems capable of operating in parallel (modules): The system must have different specialized components that can perform tasks independently and simultaneously.
2. **GWT-2:** Limited capacity workspace, entailing a bottleneck in information flow and a selective attention mechanism: There must be a central area with a smaller capacity than the combined modules. This bottleneck forces the system to learn useful, low-dimensional representations and requires an attention mechanism to select which information gets into the workspace.
3. **GWT-3:** Global broadcast: the distribution of selected information in the workspace to all modules. The information represented in the workspace must be accessible to all specialized modules.
4. **GWT-4:** State-dependent attention, giving rise to the capacity to query modules in succession for performing complex tasks: The attention mechanism that selects information for the workspace must be influenced by the system’s current state and new inputs. This “top-down attention” allows the

system to use the workspace to maintain goals and perform complex, multi-step tasks.

2.3 Higher-Order Theories

The core motivation for higher-order theories (HOTs) [13] of consciousness is “a conscious mental state requires an individual to be aware of being in that state”. Since awareness involves a representation of something, consciousness necessitates a higher-order representation of one’s own mental states. This distinguishes HOTs from other theories by emphasizing that consciousness isn’t just about a mental state itself (a “first-order” state, like seeing a red apple); it requires a separate, higher-level awareness of that state (a belief that you are seeing a red apple).

[2] focuses on two computational HOTs — perceptual reality monitoring (PRM) and higher-order state space (HOSS). These theories propose that consciousness stems from a metacognitive mechanism that monitors and assesses the reliability of first-order perceptual states, effectively distinguishing meaningful signals from noise.

Four key indicator properties are proposed:

1. **HOT-1:** Generative, top-down or noisy perception modules: This property is necessary because the core function of a monitoring mechanism is to discriminate between different sources of perceptual activity, such as top-down imagination, or perception affected by random noise.
2. **HOT-2:** Metacognitive monitoring distinguishing reliable perceptual representations from noise: It refers to a mechanism that labels first-order states as accurate representations of reality. This monitoring process is what makes the perceptual state “conscious.”
3. **HOT-3:** Agency guided by a general belief-formation and action selection system: This indicator highlights that a conscious experience has “assertoric force”—it presents itself as real and accurate, influencing an agent’s beliefs and subsequent actions. For a system to truly rely on the “realness” of a perceptual input, it must have a holistic system for reasoning and decision-making that is guided by the outputs of its metacognitive monitoring.
4. **HOT-4:** Sparse and smooth coding generating a “quality space”: This property addresses the subjective “feel” of conscious experience. It posits that consciousness is not possible without qualities, which

are defined by how a system can discriminate between different sensations. This requires a system where perceptual representations are coded sparsely (using few neurons) and smoothly (continuously), creating a “quality space” that allows for a grasp of similarities and differences between experiences.

2.4 Attention Schema Theory

Attention Schema Theory (AST) [18] proposes that the brain creates a model of attention to help control it, contributing to conscious experience [2]. The theory is used because it specifically aims to explain why we have certain intuitions and beliefs about consciousness. AST can be thought of as a higher-order theory of consciousness because it claims that consciousness depends on higher-order representations of a particular kind (in this case, representations of our attention). This model allows a system to learn how attention affects other cognitive processes and to anticipate changes in the objects of attention to adjust accordingly.

The indicator property is:

1. **AST-1:** A predictive model that represents and enables control over the current state of attention.

2.5 Predictive Processing

While not a direct theory of consciousness itself, many researchers view predictive processing [17] as a plausible necessary condition for consciousness, making it relevant for our study. PP claims that the essence of human cognition is the minimization of errors made when predicting sensory stimulation [2]. The process is modulated by attention, and it can also control action because acting can be considered as a way of reducing prediction error.

The indicator property is:

1. **PP-1:** Input modules using predictive coding

2.6 Agency and Embodiment

The theory that agency [5] and embodiment [4] are necessary for consciousness is supported by several scientific and philosophical arguments. Many scientific theories of consciousness, such as PRM (Predictive Reality Monitoring), the midbrain theory, and Global Workspace Theory (GWT), already incorporate agency as a

key component [2]. Philosophers also argue that consciousness requires a “perspective” or “point of view” on the environment, which they believe is generated through embodied agency. The paper uses this framework to introduce indicator properties that can be measured and tested in artificial systems.

The paper identifies two main indicator properties related to agency and embodiment:

1. **AE-1 Agency:** Learning from feedback and selecting outputs so as to pursue goals, especially where this involves flexible responsiveness to competing goals.
2. **AE-2 Embodiment:** Modeling output-input contingencies, including some systematic effects, and using this model in perception or control. Additionally, [2] requires the usage of a forward model to satisfy AE-2.

3 Evaluation of Recent Algorithms

In this section, we will analyze algorithms to determine how many indicator properties we can find through contemporary models, and whether additional IPs result in better performance.

3.1 Algorithms

This section will delve into a group of famous and accessible algorithms, including A2C [14], PPO [16], and TRPO [15]. We will use the particular implementation of these by Stable-Baselines3¹, a popular community-based resource of Reinforcement Learning in Python3. We will also evaluate DreamerV3 [8], a recent model-based reinforcement learning algorithm.

3.1.1 A2C

The Advantage Actor-Critic (A2C) algorithm is a reinforcement learning method that integrates principles from both value-based and policy-based approaches [14]. It belongs to the Actor-Critic framework, wherein the actor component is responsible for learning a policy that determines which actions to take, while the critic evaluates these actions by estimating their expected returns. A2C improves learning stability and efficiency by scaling the policy gradient with an estimate of the advantage function, defined as the difference between the return and a learned baseline value function, which effectively measures how an action’s performance

¹The project’s source code can be found [here](#).

deviates from the expected state value.

The Advantage Actor-Critic (A2C) algorithm offers several innovations that address key limitations of earlier policy gradient methods. Traditional policy-based approaches often suffer from high variance in gradient estimates, as they rely on full trajectory returns that can fluctuate significantly between episodes due to the stochasticity of the environment. A2C mitigates this problem by incorporating a critic network to estimate the expected value of states and by using the advantage function, the difference between the observed return and the estimated value, to update the policy. This adjustment reduces variance in gradient estimates while preserving the learning signal, resulting in less fluctuation in the policy due to noise and efficient training.

Furthermore, A2C combines the strengths of policy-based and value-based reinforcement learning methods. The actor directly optimizes the policy, while the critic provides value-based feedback that guides and stabilizes this optimization process. This dual structure allows the algorithm to achieve a balance between exploration and exploitation, improving sample efficiency and convergence reliability. Overall, A2C’s reduced variance, training stability, and hybrid design make it a robust and widely adopted baseline for reinforcement learning research.

3.1.2 TRPO

Trust Region Policy Optimization (TRPO) is a reinforcement learning algorithm designed to improve policy gradient methods by ensuring stable and monotonic policy improvement. Without proper constraints, policy optimization can occasionally compute large steps that cause catastrophic degradation of performance [15]. TRPO addresses this issue by constraining each policy update to a “trust region” using Kullback-Leibler (KL) divergence as a distance measure between the old and new policies [15]. The algorithm provides theoretical guarantees of monotonic improvement by optimizing a surrogate objective subject to the constraint that the KL divergence between successive policies remains bounded. This constraint-based approach ensures that the new policy does not deviate too far from the old one, preventing the large, destabilizing steps that plague standard policy gradient methods while still allowing for efficient policy optimization.

TRPO is one of the first algorithms to successfully combine policy gradient methods with theoretical guarantees of improvement, paving the way for more efficient and stable algorithms such as Proximal Policy Optimization (PPO).

3.1.3 PPO

Proximal Policy Optimization (PPO) is a reinforcement learning algorithm like TRPO that improves policy-based learning through a more stable and efficient training process. It belongs to the family of policy gradient methods, where the agent directly learns a policy that maps states to actions. PPO was introduced by OpenAI as a simplified yet effective alternative to more complex algorithms like Trust Region Policy Optimization (TRPO) [16].

The core idea of PPO is to strike a balance between improving the policy and avoiding large, destabilizing updates. It does this by using a clipped objective function, which limits how much the new policy can diverge from the old one during each update [16]. This constraint ensures smoother learning, preventing performance collapse due to overly aggressive policy changes.

PPO achieves strong performance while maintaining computational simplicity. It is easy to implement, requires fewer hyperparameter adjustments than previous methods, and works well across a wide range of environments. As a result, PPO has become one of the most widely adopted algorithms in reinforcement learning research and applications, such as in reinforcement learning from human feedback (RLHF) in state-of-the-art LLM models.

3.1.4 DreamerV3

DreamerV3 is a model-based algorithm that learns to solve tasks across a wide range of applications with fixed parameters. The most interesting aspect for us is that DreamerV3 is exclusively operated on data generated from its imagination [8], supported by the architecture of DreamerV3. The algorithm learns the world from experience, which consists of three neural networks: the world model predicts the future outcomes of potential actions, the critic evaluates the value of each situation, and the actors learn to achieve desirable situations. Moreover, DreamerV3 is the first algorithm to solve the problem of collecting diamonds in Minecraft without supervised data — a problem that has been the focal point of reinforcement learning in recent years [8].

With DreamerV3, the world model uses both a recurrent sequence model and a predictive approach to handle the sensory inputs. Firstly, a stochastic representation z_t is encoded from a sensory input x_t . After that, the sequence model takes in the recurrent state h_t , the action a_t , and the representation z_t to predict the next recurrent state h_{t+1} (Figure 1). At each time step, z_t and the recurrent state h_t form the base of the model’s recurrent internal representation, which is used to predict rewards and select action outputs by the

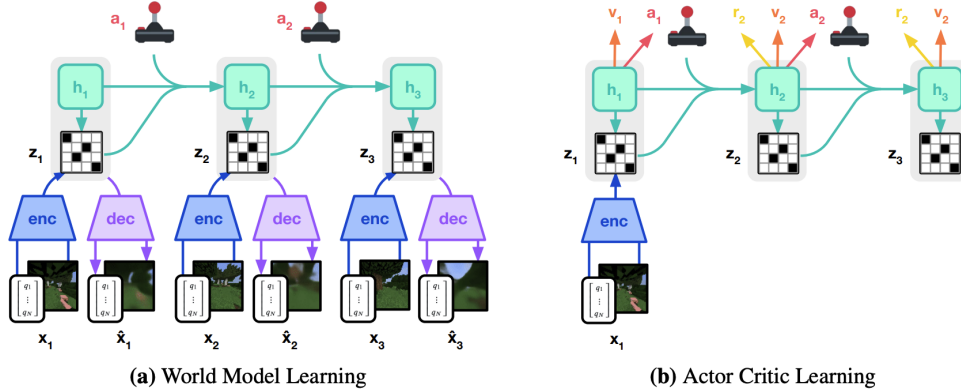


Figure 1: DreamerV3 World Model and Actor Critic Architecture

Actor Critic model (Figure 1).

3.2 Indicator Properties Satisfied

The indicator properties satisfied by each algorithm are summarized in Table 1. Our assessment with A2C, TRPO, and PPO is that they do not possess any IP except **AE-1**. Since **AE-1** requires the agent to learn from feedback and select actions to pursue goals, almost all, if not all, Reinforcement Learning agents will satisfy this requirement.

Regarding DreamerV3, based on the architecture of the world model and on the fact that the internal representation has been utilized successfully to solve different problems without hyperparameter tuning [8], we claim that the recurrent internal representation is organized, coherent, and stable. Thus, the DreamerV3 satisfies both **RPT-1** and **RPT-2** properties. Secondly, the internal presentation is always predicted by the world model. This predicted presentation is passed through Actor and Critic networks so the agent can make decisions. By passing the predicted presentation through these two neural networks, DreamerV3 optimizes its policies through gradient descent, a process dedicated to the continual minimization of prediction error and objective loss. Hence, DreamerV3 utilizes the output of a predictive input model (world model) to strategically choose action in order to minimize error and to optimize its current goal. Therefore, we claim that DreamerV3 also satisfies the **PP-1** indicator.



Figure 2: Picture of the environment with the Ant standing up

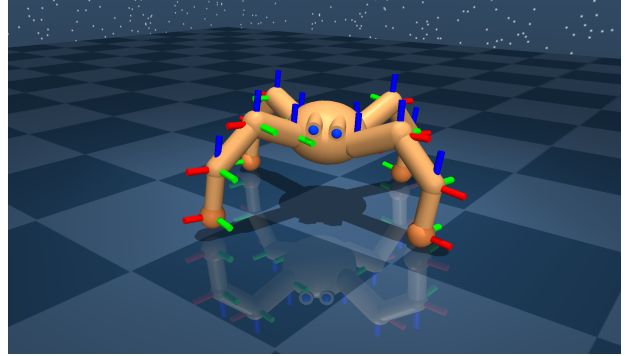


Figure 3: Ant with reference frames of observation

3.3 Performance Comparison

3.3.1 Environments

We initially tried a wider selection, including an in-house environment using PyGame and Gymnasium, Atari suites (Tennis and Pong), and the Gymnasium Mujoco suites. However, in order to set the foundation to satisfy the AE-2 property, we narrowed the pool of environments down to those that offered agents with egocentric observation, hoping to introduce a sense of embodiment. In other words, the agent senses its environment from a self-centered perspective, not from a god’s eye view, such as in Pong.

In the end, the decision was to use an environment featuring a quadruped agent (Ant-v5 and DMC Quadruped Walk), though any environment that offers the same egocentric agent observation would work. Due to the limitation on time and computing power, we prioritize building a process to compare different sets of IP. Future work can explore additional environments to monitor the impact of environmental contexts on IP. For this section, we will use DMC Quadruped Walk².

3.3.2 Rewards

The Quadruped Walk task is to walk around in an upright position. The default reward function encourages a *move_reward* larger than 0.5 without an upper limit. The reward is 1 when the move speed is ≥ 0.5 , and scaled as a linear sigmoid function by how far the speed is below 0.5. In addition, the environment also motivates an upright position. The upright reward is 1 when the z-axis of the agent is 0 degrees away from the environment z-axis, else decreasing the farther it is away from the environment z-axis with value in the

²Initially, we planned to use Ant-v5 with others algorithms and had run several experiments using this environment. However, DreamerV3 requires significant change to work with Ant-v5. Therefore, we chose DMC Quadruped as a training/evaluating environment for DreamerV3 and this section, since it is natively supported by DreamerV3.

range of $[0, 1)$.

$$reward = upright_reward \times move_reward$$

3.3.3 Observations

The agent’s observation consists of multiple sensory inputs that describe its physical state and movement. These include an egocentric state vector, force torque readings, inertial measurements (IMU), torso orientation, and torso velocity. Together, these signals provide the necessary information for the agent to perceive its posture, balance, and location during training.

3.3.4 Actions

The agent acts on the environment by applying torques to its joints. Torques are represented as a 12-dimensional vector of values that act as “rotational muscle power” for the joints. By applying these values, the agent controls the angular acceleration of each limb segment to coordinate movement.

3.3.5 Results

Figure 4 illustrates clear differences in performance across the four algorithms. TRPO achieves the highest mean episode reward throughout training, with PPO and A2C following behind but still showing steady improvement. DreamerV3 consistently underperforms relative to these model-free algorithms given the training budget. Initially, DreamerV3 also improves at a much slower rate, but faster between 0.75e6 and 1.7e6 steps, before plateauing again. Although DreamerV3 incorporates more indicator properties and relies on a complex recurrent world model, this additional sophistication does not translate into higher task performance. Instead, the results suggest that increased model complexity can introduce longer training times and higher computational demands without necessarily providing a performance benefit for this environment with a limited budget.

Table 1: Indicator Properties and Algorithms.

Indicator Properties		Algorithms			
		DreamerV3	PPO	TRPO	A2C
RPT-1	Input modules using algorithmic recurrence	✓	—	—	—
RPT-2	Input modules generating organised, integrated perceptual representations	✓	—	—	—
GWT-1	Parallel operating specialized systems	—	—	—	—
GWT-2	Limited capacity workspace	—	—	—	—
GWT-3	Global broadcast	—	—	—	—
GWT-4	State-dependent attention	—	—	—	—
AST-1	A predictive model representing and enabling control over the current state of attention	—	—	—	—
PP-1	Input modules using predictive coding	✓	—	—	—
AE-1	Agency: Learning and selecting outputs to pursue goals	✓	✓	✓	✓
AE-2	Embodiment: Modeling output-input contingencies to use in perception or control	—	—	—	—
HOT-1	Generative, top-down or noisy perception modules	—	—	—	—
HOT-2	Metacognitive monitoring	—	—	—	—
HOT-3	Actions guided by beliefs, which are updated by metacognitive monitoring.	—	—	—	—
HOT-4	Sparse and smooth coding generating a “quality space”	—	—	—	—

4 Bottom-up construction of a conscious system

In this section, we attempt to construct an agent based on the proposed list of IPs. Being a reinforcement learning agent naturally helps the system satisfy a base set of indicator properties. On top of that, we will introduce two additional groups of indicator properties. This division is due to the complexity of different IP requirements, limited resources, and the available algorithms.

The first group contains the addition of **RPT-1** and **RPT-2** with the help of recurrence algorithms. The

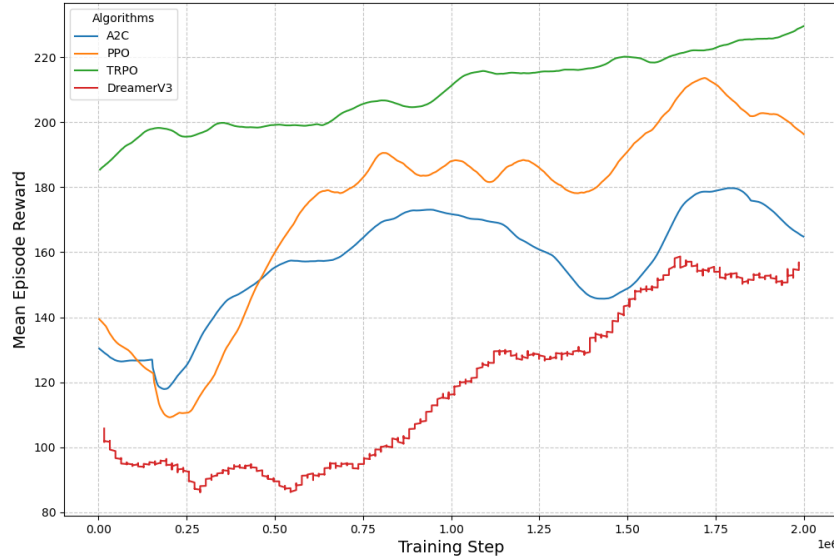


Figure 4: Average training reward of algorithms on Quadruped Walk environment.

second group features **AE-2** and **PP-1** with the support of the Kalman filter. For each group, we create a respective custom environment based on Ant-v5, a respective baseline model, and augmented models that satisfy the targeted IPs. We are looking forward to examine the impact of each group’s additive IPs on the augmented models’ performance compared to the respective baseline model.

4.1 Environments

We evaluated our constructed agent on Ant-v5 by Gymnasium³. In this environment, the agent is similar to the DMC Quadruple Walk agent, a 3D quadruped “ant”, consisting of a torso with four legs attached, and each leg has two parts. However, the ultimate goal of the Ant-v5 agent is to move in the right direction by applying torques to its body parts.

4.1.1 Rewards

Ant-v5’s reward function is broken down into 4 different factors.

$$reward = healthy_reward + forward_reward - ctrl_cost - contact_cost$$

³Even though we evaluate current algorithms on DMC Quadruped, we decided to keep the experiment section on Ant-v5. We already have results with building agent in Ant-v5, and we discovered DMC suite much later.

- *healthy_reward*: Every timestep that the Ant is healthy, it gets a reward of fixed value *healthy_reward* (default is 1). The Ant is considered unhealthy when any of the state space values is no longer finite, or the z-coordinate of the torso (the height) is not in the closed $[0.2, 1.0]$.
- *forward_reward*: the reward is positive if the Ant moves in the positive x direction (to the right side). The reward is also scaled with the distance it has traveled.
- *ctrl_cost*: A punishment for taking a large action. It is scaled with the multitude of actions.
- *contact_cost*: A punishment for having a large external force with the environment.

4.1.2 Observation

The environment’s observation space is an array of values in the following orders:

- *qpos*: 13 elements of position values of the robot’s body parts.
- *qvel*: 14 elements of the velocities of these individual body parts.
- *cfrc_ext*: 78 elements of center of mass-based external forces on the body parts. It has a shape $13 * 6$ (nbody * 6), for the ant as 13 observable body parts. Six values include external forces - force x, y, z, and torque x, y, z.

4.1.3 Action

The ant action space is very similar to the Quadruped action space (Section 3.3.4). The number of joints is reduced from 12 to 8, so each action only specifies 8 torques, and the min-max values of all actions are bound between $[-1, 1]$.

4.1.4 Custom Environments

For the first group with the intention of satisfying the Recurrent Processing Theory, we identify that the baseline model TRPO performs exceptionally well on the normal Ant-v5 environment. We attempted to add the recurrent features to the agent, but it yielded no significantly different result. With further investigation into the environment observation, we see that the observation already includes the velocity information of each body part (*qvel* section). Thus, we suspect that this information nullifies the expected effect of the additional recurrent algorithm since we hoped the agent would derive its velocity from the recurrent observation of its location. Therefore, for the first group, the custom environment will be the Ant-v5

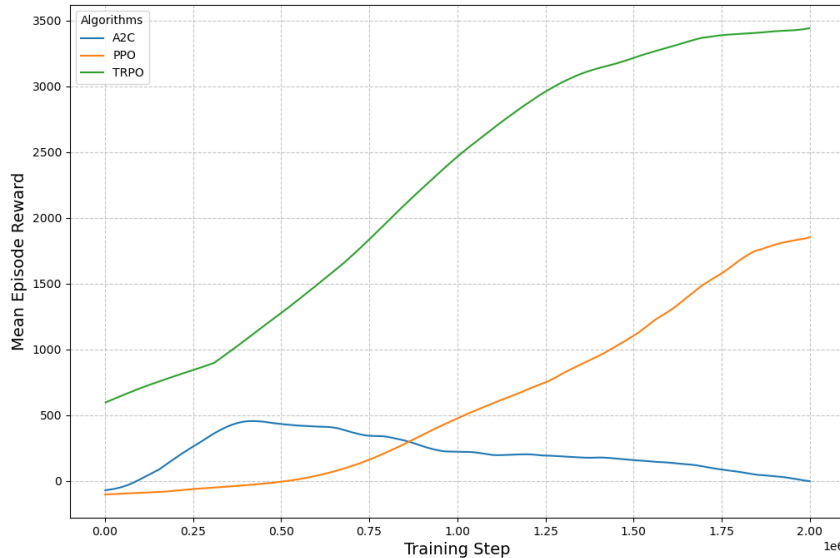


Figure 5: Average reward of algorithms on Ant-v5 environment

without the *vel* section in the observation.

For the second group, with the intention of satisfying the Embodiment and the Predictive Processing Theory, we intend to implement the Kalman filter. However, the Kalman filter’s purpose is to reduce noise from observation, yet our artificial environment is inherently noise-free. We decide to introduce the Gaussian noise to the system in order to simulate imperfect sensing. Thus, the custom environment of the second group is Ant-v5 with Gaussian noise added to its observation.

4.2 Based model and features added

We began our experiments using Trust Region Policy Optimization (TRPO) as the baseline model, as it demonstrated the highest performance among the available Stable-Baselines3 (SB3) algorithms, outperforming alternatives such as PPO and A2C (Figure 5).

For the first group, we incorporated recurrent neural architectures, specifically Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) networks, into the feature extractor of the TRPO model. When the agent receives an observation from the environment, it will first process the information by using the feature extractor. The output of the feature extractor is then passed through a fully-connected network and used for either the actor or critic network.

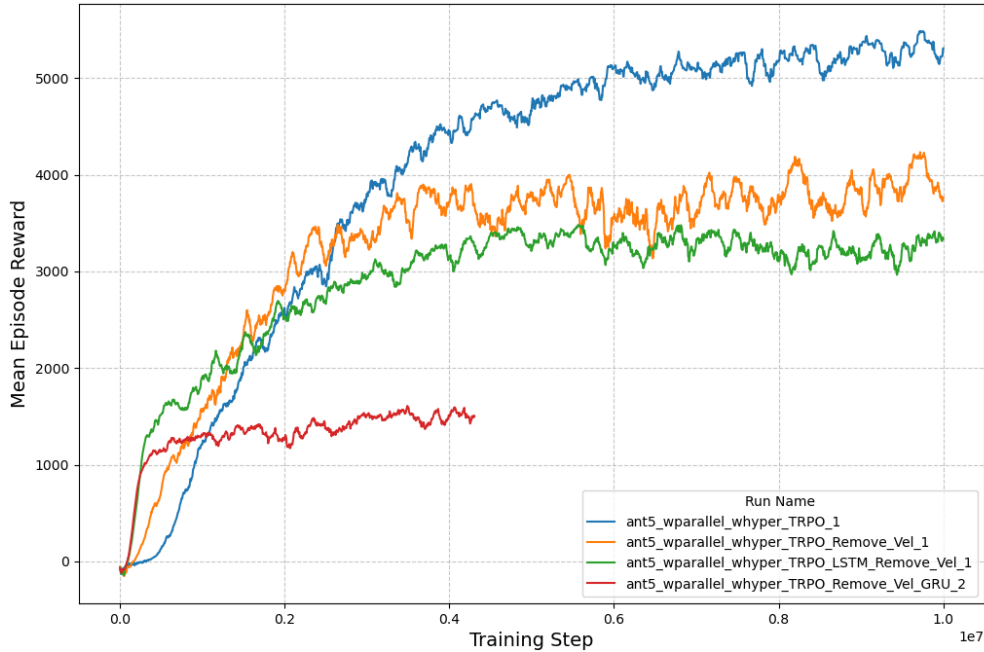


Figure 6: Mean episode reward over training of Recurrent Variations

To further investigate the model’s dependency on input features, we removed the *qvel* component from the observation space as explain in 4.1.4. As expected, the new baseline model trained without velocity information (‘removed_vel’) performed worse than the TRPO trained on the full observation. When we combined the ‘removed_vel’ agent with recurrent architectures, we expected the agent to perform better since the recurrence would create a sense of movement relatively to previous states, implying the observation of speed. Surprisingly, the performance degraded with the introduction of recurrent algorithms, with the “removed_vel + LSTM” model performing worse than the new baseline model, and the “removed_vel + GRU” model yielded the poorest results overall (Figure 6).

For the second group, we explored the integration of a Kalman filter to improve observation estimation, as the Kalman filter is specifically mentioned as a good example of a forward model in embodied systems (AE-2) [2]. We also augmented the environment by introducing the Gaussian noise to the observations in order to simulate imperfect sensing as explained in 4.1.4. The intention was to assess whether the Kalman filter could mitigate the effects of this noise and enhance agent performance.

The new baseline agent with noisy observations and without Kalman filter does indeed has worse performance than the TRPO on the original noise-free observation. Unlike the first group, the addition of a Kalman filter significantly increases the performance of the agent (Figure 8).

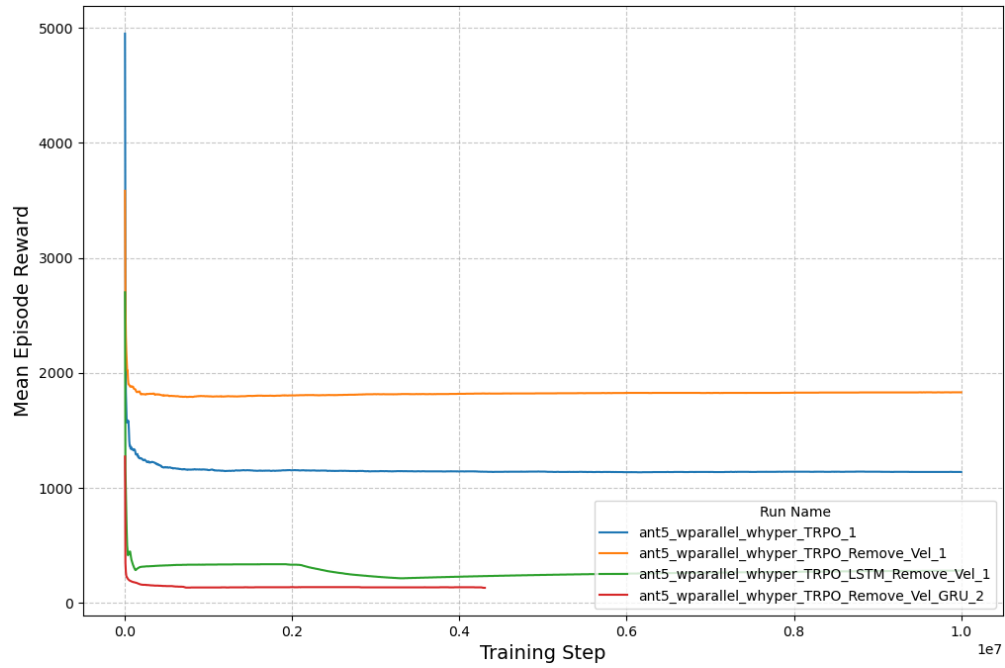


Figure 7: Mean fps^a over the training of Recurrent Variations

^aFrames per second.



Figure 8: Mean episode reward over training of Noisy Variations

4.3 Indicator Properties Satisfied

Using an RNN as the feature extractor satisfied the **RPT-1** requirement of using recurrence in input modules. At the same time, **RPT-2** is also satisfied because the feature extractor is generating perceptual representations, which can be used for choosing and evaluating actions.

However, not only did this modification decrease the performance of the agent compared to the baseline model with velocity information removed, but it also resulted in a notable decline in the training speed (Figure 7). The training process became significantly more demanding, leading to slower convergence. Therefore, while satisfying the requirements of **RPT-1** and **RPT-2**, we are decreasing the performance of the agent in both quality and training efficiency.

The Kalman filter is a forward predictive model, and the agent uses it for perception. Therefore, the agent with the Kalman filter satisfies the **AE-2** and **PP-1** requirements. Unlike recurrent properties, introducing the Kalman filter does help the agent perform better (Figure 8). While the performance of the noisy agent decrease significantly, the Kalman filter version performs significantly better while still worse than the original version. Nonetheless, this demonstrates the effectiveness of a forward model in the sensory module.

Naturally, an agent in a reinforcement learning environment will satisfy **AE-1**. At every learning step, the agent receives the feedback (reward) from the environment based on their previous action. Then, the agent will update its policies accordingly to pursue a better reward from its future interactions with the environments.

5 Top-down design of a conscious system

This section introduces the LIDA cognitive architecture (Learning Intelligent Decision Agent [7]) and a top-down design of a “conscious” software system. LIDA was chosen as the basis for our top-down design because it implements the Global Workspace Theory of consciousness along with many other psychological and neuro-psychological theories. Therefore, a subset of Butlin et al.’s indicator properties (**GWT-1**, **GWT-2**, **GWT-3**, **GWT-4**) are trivially satisfied at the onset by basing the initial design on LIDA.

In the sub-sections that follow, we give a brief overview of LIDA’s modules and processes, followed by a set of implementation choices and enhancements that we propose to satisfy additional indicator properties. We conclude with an analysis of the indicator properties that our high-level design satisfies. The result is a theoretical design based on LIDA. It is also important to note that, since not all components of LIDA have

specific computational designs, we cannot offer performance evaluation in RL environments. Instead, we can only evaluate how our design satisfies the IP framework in theory.

5.1 LIDA

LIDA is a biologically inspired, systems-level cognitive architecture. In this section, it is treated strictly as a theoretical framework—a top-down exploration of whether such a system could function as a unified model of cognition, rather than a guide or an attempt for design or implementation. While LIDA models “minds” not brains (that is, it abstracts cognition from a given physical substrate), it was heavily inspired by neuroscientific research.

LIDA incorporates numerous short- and long-term memory modules along with various supporting processes. Each of the relevant modules and processes will be explained in the subsections that follow.

5.1.1 LIDA’s cognitive cycle

Cognition emerges in LIDA from a continual series of cognitive cycles, which are comparable to the “action-perception cycle” referred to by many cognitive scientists. Cognitive cycles can be thought of as the “atoms” of thought, and activities such as deliberation and problem solving can take many cognitive cycles. LIDA’s cognitive cycle, depicted in Figure 9, can be further subdivided into three phases: (1) perception and understanding, (2) attention, and (3) action and learning.

In the *perception and understanding phase*, environmental stimuli activate low-level feature detectors in Sensory Memory. These, in turn, activate higher-level, modality-specific feature detectors in Perceptual Associative Memory (PAM). Activation continues to propagate through PAM’s activation graph, eventually activating perceptual representation for objects, events, and concepts. PAM’s representations that received sufficient activation are instantiated as *percepts* and added to LIDA’s Current Situational Model (CSM). Percepts and other content in the CSM then cue (query) LIDA’s long-term memory modules, bringing additional situation-relevant content into the CSM. The CSM can be understood as representing an agent’s “preconscious” understanding of its current situation.

In the *attention phase*, attention codelets (special-purpose parallel processors) search for salient content in CSM that satisfies their individual selection criteria. Attention codelets and their selected content are then formed into one or more *coalitions* that compete in a winner-take-all competition in the Global Workspace. The winning coalition’s content is then *globally broadcast* to all of LIDA’s modules and processes. An agent

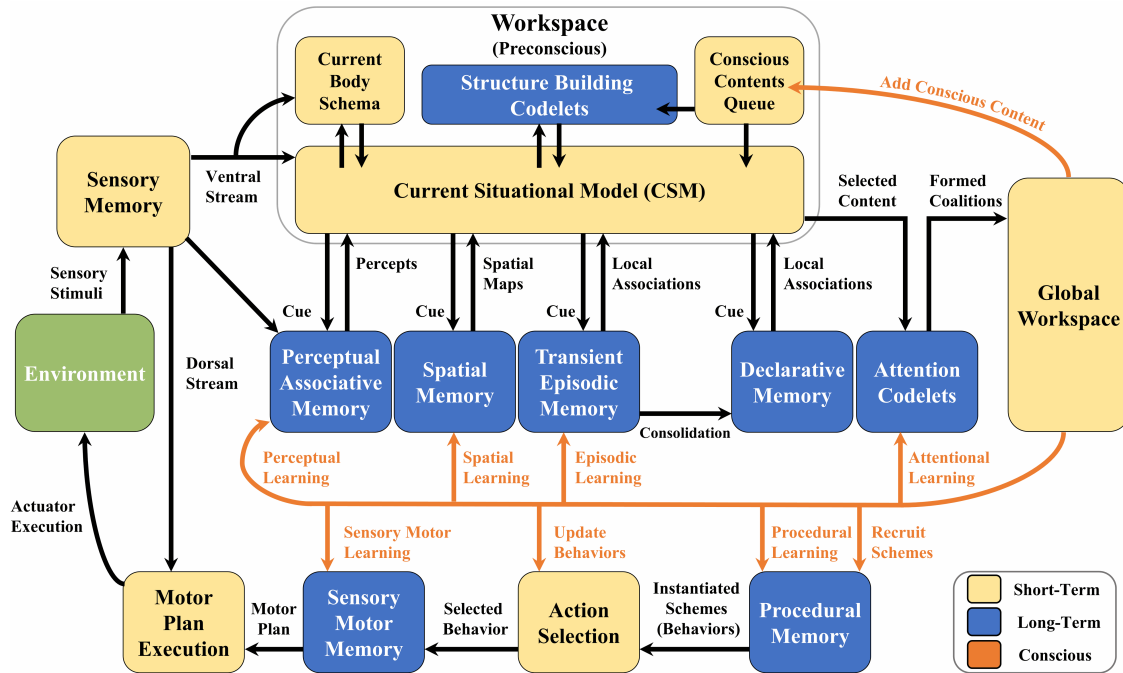


Figure 9: Picture of the LIDA's cognitive cycle

is considered to be “conscious” of the mental representations in the global broadcast.

In the *action and learning* phase, Procedural Memory uses the content in the global broadcast to instantiate a set of situation-relevant *behaviors*. The Action Selection module chooses one of these and sends it to the Sensory Motor System for execution. The Sensory Motor System is composed of two modules: Sensory Motor Memory and Motor Plan Execution. Sensory Motor Memory instantiates an appropriate motor plan for the selected behavior, and Motor Plan Execution oversees its execution through a process of online control [7]. The “conscious” content from the global broadcast is also used by LIDA’s many learning mechanisms to update representations in LIDA’s short- and long-term memory modules.

5.1.2 Sensory Memory & Perceptual Associative Memory (PAM)

Sensory Memory works with Perceptual Associative Memory (PAM) to support the recognition of objects and events in the environment. Sensory Memory is a short-term memory module composed of a set of modality-specific neural networks that transduce environmental stimuli into sensory representations (see Figure 10). PAM is a long-term memory module typically implemented as an “activation graph” containing nodes that represent features, objects, feelings, events, categories, concepts, etc., and edges that represent the relationships between them. Edges also support the propagation of activation between nodes. Additionally,

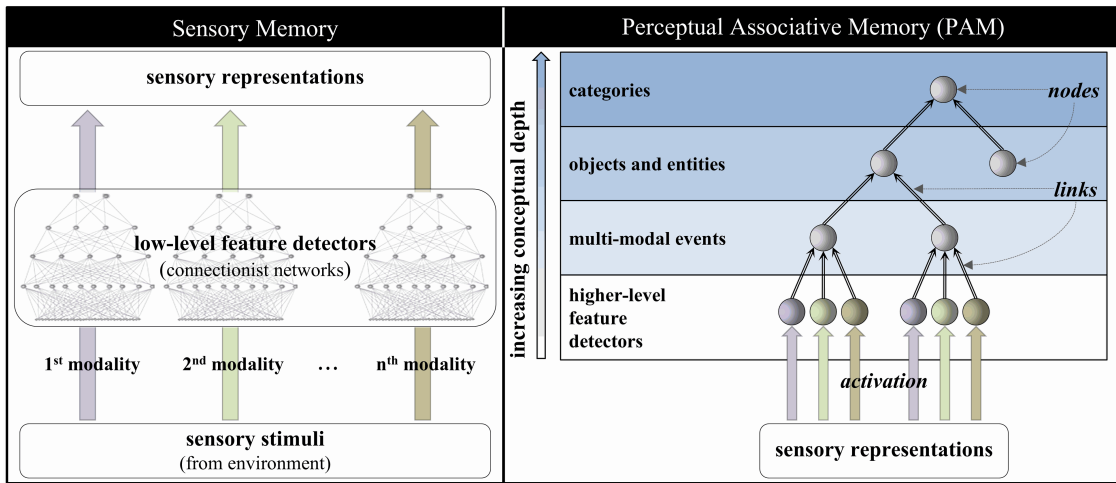


Figure 10: A visualization of the LIDA’s sensory memory

PAM representations are the basis for LIDA’s motivation system, supporting an agent’s ability to identify events that it “likes” (e.g., eating cake) and “dislikes” (e.g., writing papers).

5.1.3 Global Workspace & Current Situational Model

In general, there is too much information (content) in LIDA’s Current Situational Model (CSM) for an agent to simultaneously attend to all of it in a single cognitive cycle. An information bottleneck (saliency filter) is needed to focus an agent’s limited computational resources on the aspects of its current situation that are most important, urgent, novel, interesting, etc. Attention codelets identify salient content in the CSM, and this content is then used to form coalitions. The resulting coalitions compete in the Global Workspace for conscious broadcast. The coalition with the highest activation wins the competition for consciousness, and the Global Workspace makes this content available to all modules and processes.

LIDA’s learning mechanisms depend on “conscious content” in the global broadcast. This is in accordance with Global Workspace Theory’s “Conscious Learning Hypothesis”, which states that all significant learning requires consciousness.

5.1.4 Codelets

LIDA has two types of special-purpose processors: attention codelets and structure building codelets. These operate in parallel on the content in the CSM. The specific implementations of these processors depend on the needs of each agent, but, in general, structure building codelets form associations between content in

the CSM (e.g. causality links) or construct new representations (e.g. simulations and plans), and attention codelets identify the most salient content in the CSM.

Different types of attention codelets have been proposed in the LIDA literature. We will briefly mention two of these here. *Expectation codelets* are created as a result of action selection. Their purpose is to monitor the CSM, looking for the expected results (anticipated consequences) of recently executed actions. *Intention codelets* identify content in the CSM that is deemed useful for the agent to achieve its current goals.

5.1.5 Procedural Memory & Action Selection

Procedural Memory is a long-term memory module that contains *schemes* — data structures that encode contexts, actions, and their expected results. Each scheme asserts “what might occur if an agent were to execute an action in a given situational context” [11]. A base-level activation is also associated with every scheme that quantifies its reliability; that is, the likelihood of an action achieving its expected result, given that the context is met. These predictions form an internal model of an agent’s interactions with its environment. This internal model enables an agent to predict the future and select the best actions to take based on its past experiences.

Following each global broadcast, Procedural Memory activates schemes whose contexts overlap with the content in the broadcast. Schemes receiving sufficient activation are instantiated as behaviors and passed to the Action Selection module for selection and eventual execution. Action Selection can choose behaviors for internal or external execution. The internal execution of behaviors is fulfilled by structure building codelets and is a form of mental simulation (i.e., mental imagery) [11]. The external execution of behaviors is fulfilled by LIDA’s Sensory Motor System.

5.2 Indicator Properties Satisfied

Firstly, the majority of LIDA’s components can be executed in parallel as stated in Section 5 ‘LIDA is a massively parallel system with the processes of each module operating independently and asynchronously in response to current local conditions’ [7]. This helps LIDA achieve **GWT-1** property quite easily. On the other hand, the preconscious content of the CSM is vast and complex. Dealing directly with all of this content would be overwhelming for an agent. Therefore, LIDA needs the attention codelets to act as a salient filter. It only picks the most meaningful representations to feed the global broadcast. From there, only one coalition will be chosen for global broadcast. For said reason, LIDA introduces an information

flow bottleneck that satisfied **GWT-2**, as well as a global broadcast that satisfied **GWT-3** at the same time. Furthermore, examining the intention codelet, it is obvious that its role is to support the agent in achieving its current set goal. Therefore, the output coalitions are dependent on the current state of the agent while also allowing the agent the ability to query important modules in succession to perform complex tasks. Therefore, LIDA also satisfies the **GWT-4** indicator property.

In addition, LIDA’s cognitive cycle also learns from the global broadcast, which is influenced by the attention codelets. The content then triggers the Procedural Memory and Action Selection module, making the agent take actions that are relevant to the current goal. At the same time, the broadcasted content is added to the CSM, which the CSM can cue other long-term memory modules to find relevant information to the current conscious content. Thus, LIDA’s cognitive cycle as a whole satisfies the **AE-1** indicator.

On the other hand, the specific implementation of the Sensory Memory is still an undefined answer. There are proposed approaches that would be beneficial to pursue further in this discussion. Firstly, there is the deep learning solution mentioned in [7]. Since the nature of the task is feature detection, including visual representation in many cases, Recurrent Neural Networks such as LSTM and GRU are very performant. As discussed in [9], LSTM has the ability to bridge very long time lags in case of problems, as well as handling noise [9]. [3] also mentions the widely accepted superiority of RNN on sequence-based tasks with long-term dependencies [3]. As a result, the LSTM and GRU units are very effective on the chosen task of sequence modeling polyphonic music data and raw speech signal data, and their performance is comparable [3]. Moreover, the Sensory Memory (5.1.2) is open to the implementation of LSTM and GRU networks, especially when the input modality is sequential data such as visual and audio. This allows LIDA to satisfy the **RPT-1** since Sensory Memory is an input module, and can use recurrent algorithms. At the same time, the output of the Sensory Memory is then passed into PAM and CSM, used in the recognition memory system, and to continuously update the agent’s understanding of its environment. Thus, we agree that the Sensory Memory also satisfied the requirement of **RPT-2**.

Additionally, the Procedural Memory is considered to be a predictive model of context, action, and outcome. The outputs of Procedural Memory are extensively used in LIDA’s external as well as internal interaction by Action Selection. As a result, the combined process of Procedural Memory and Action Selection is predictive and can directly influence the current state of an agent, specifically to look for an expected outcome. This description is aligned with Attention Schema Theory, which helps LIDA satisfy **AST-1**.

Since the LIDA cognitive model is a comprehensive, complex architecture with a significant number of

modules, the specific interpretation of some of these modules is not clearly stated. This opens up the opportunity for LIDA to adapt its implementation to a variety of different tasks. One of the variations we will discuss today was proposed in 2015 that incorporates the Kalman filter into its motor system [6].

However, first we need to understand the purpose of the Kalman filter [10]. The said filter is a forward model that predicts the outcome of a process based on a set of inputs and parameters. In practice, the inputs are often noisy and contain inaccurate measurements. The Kalman filter takes these inputs and produces outputs that are less noisy and more accurate. The real power of the Kalman filter is not smoothing measurements, but rather its capability to estimate parameters that can not be measured or observed with high accuracy.

The inner flow of a Kalman filter proceeds in two main phases that repeat at each time step: prediction and update. First, the filter takes the previous state estimate and covariance (uncertainty) and projects them forward via the system's dynamical model to form a predicted estimate. Then, once a new measurement arrives, it computes the difference between the measurement and the predicted measurement, computes the Kalman gain (weighting how much to trust the measurement vs prediction), and fuses prediction and observation to produce the succeeding estimate and updated covariance (reflecting reduced uncertainty).

Therefore, this version of LIDA that incorporates the Kalman filter into its motor system is a reasonable utilization of the Kalman filter. This combination is based on a hypothesis that the central nervous system (CNS) internally predicts the outcome of a movement by modeling the dynamics of the environment using a so-called (forward) internal model. Therefore, [6] attempts to integrate the Kalman filter, a forward predictive model, into the LIDA sensory motor for controlling and correcting errors in the motor system observation [6]. At the end, they conclude that an agent integrated with a Kalman filter is able to simulate both (1) human estimation of the lifting movement within one trial, and (2) human estimation between lifting trials driven by memory of errors. This version of LIDA satisfies both **AE-2** and **PP-1** for its input module uses a predictive model to simulate the relationship between input vs. output for perception and control purposes.

While this architectural design demonstrates considerable promise and satisfies a substantial number of the specified indicator properties, the implementation of an agent based on this framework presents significant challenges. Numerous components within the design lack clearly defined computational methods. Furthermore, satisfying HOT 1-4 remains unattainable, as these requirements lack established design precedents upon which to build. These difficulties underscore the considerable complexity inherent in translating theoretical frameworks into concrete implementations.

6 Conclusion

This project set out to examine whether current artificial agents, as well as our custom systems, can meaningfully satisfy [2]’s recently proposed indicator properties of consciousness. Across three complementary approaches, (1) evaluating current algorithms, (2) analyzing a top-down cognitive architecture, and (3) constructing a bottom-up agent, we found that while several systems display subsets of these properties, the relationship between indicator properties and practical performance remains unresolved.

Our evaluation of state-of-the-art reinforcement learning models highlights this tension clearly. DreamerV3 satisfies more indicator properties than algorithms such as A2C, PPO, or TRPO, yet it does not outperform them in task-specific metrics and requires substantially longer training. Although DreamerV3 claims stronger generalization and more efficient use of experience [8], these advantages did not translate directly into behavioral superiority on a limited budget in our experiments. This raises a critical concern: indicator properties may make a system design an agent “closer” to consciousness, according to computational functionalism, while performing worse in a certain benchmark-oriented task.

Our bottom-up design experiments show that adding indicator properties incrementally does not consistently improve an agent’s capabilities. Some features, such as the predictive Kalman filter, provide measurable gains, whereas recurrent processing does not reliably enhance performance. This again underscores that the functional value of an indicator property is not equivalent to its theoretical role in consciousness, and the connection between consciousness and performance may not be directly related.

Finally, architectures like LIDA show that, in theory, a system can integrate most of the proposed properties, but it may be difficult to implement such an agent in practice. Indicator properties associated with Higher-Order Theories (HOT) also lack clear computational implementations. This underscores the challenge of mapping computational implementations to IPs, which are theoretical constructs.

Across all three approaches, a consistent challenge emerges: the IP framework, while theoretically promising, proves difficult to apply in practice. Satisfying more indicators does not straightforwardly transfer to better agent behavior or task performance, in part because optimizing for more IP often pulls design decisions away from goal-oriented, task-specific considerations. This reflects a broader difficulty: the framework was not built with direct software implementation in mind, and mapping abstract properties like HOT onto concrete architectures requires significant interpretive effort.

That said, our experience does not invalidate the framework, but rather highlights how hard the problem

is. Future work should focus on refining and formalizing these criteria to make them more actionable and focus on identifying settings where consciousness-like properties may offer concrete functional advantages. Ultimately, just as the Turing Test redefined how we evaluate machine intelligence, we may need similarly grounded benchmarks that reward not merely the number of properties satisfied, but the degree to which they contribute to coherent, capable behavior.

Appendices

A Hyperparameters

For transparency and reproducibility, this appendix provides a comprehensive list of the hyperparameters used in the experiments presented in this paper. These settings apply to both the comparative analysis of current algorithms and the bottom-up construction of the proposed agent due to the similarity in the environment. Unless explicitly noted otherwise, all algorithms were executed using the standard default parameters provided by the Stable-Baselines3 library or the implementation of the respective base model.

A.1 A2C

Hyperparameter for A2C

Parameter Name	Value	Description
<code>normalize</code>	True	Normalizing the environment.

A.2 TRPO

Hyperparameter for TRPO

Parameter Name	Value	Description
<code>normalize</code>	True	Normalizing the environment.
<code>batch_size</code>	128	Minibatch size for the value function.
<code>cg_damping</code>	0.1	Damping in the Hessian vector product computation.
<code>cg_max_steps</code>	25	Maximum number of steps in the Conjugate Gradient algorithm for computing the Hessian vector product.
<code>gae_lambda</code>	0.95	Factor for trade-off of bias vs variance for Generalized Advantage Estimator.
<code>gamma</code>	0.99	Discount factor.
<code>learning_rate</code>	0.001	The learning rate for the value function, it can be a function of the current progress remaining (from 1 to 0).
<code>n_critic_updates</code>	20	Number of critic updates per policy update.
<code>n_steps</code>	1024	The number of steps to run for each environment per update.
<code>sub_sampling_factor</code>	1	Sub-sample the batch to make computation faster.
<code>policy_kwargs</code>	See footnote	Additional keyword arguments for the policy network construction.

*`policy_kwargs`: `{log_std_init : -1, ortho_init : False, activation_fn : nn.ReLU, net_arch = {pi = [256, 256], vf = [256, 256]}}`.

A.3 PPO

Hyperparameter for PPO

Parameter Name	Value	Description
<code>normalize</code>	True	Normalizing the environment.
<code>batch_size</code>	32	Minibatch size for the value function.
<code>n_steps</code>	512	The number of steps to run for each environment per update.
<code>gamma</code>	0.98	Discount factor.
<code>learning_rate</code>	1.90609e-05	The learning rate for the value function, it can be a function of the current progress remaining (from 1 to 0).
<code>ent_coef</code>	4.9646e-07	Entropy coefficient for the loss calculation.
<code>clip_range</code>	0.1	Clipping parameter, it can be a function of the current progress remaining (from 1 to 0).
<code>n_epochs</code>	10	Number of epoch when optimizing the surrogate loss.
<code>gae_lambda</code>	0.8	Factor for trade-off of bias vs variance for Generalized Advantage Estimator.
<code>max_grad_norm</code>	0.6	The maximum value for the gradient clipping.
<code>vf_coef</code>	0.677239	Value function coefficient for the loss calculation.

References

- [1] B. J. Baars. *A Cognitive Theory of Consciousness*. Cambridge University Press, Cambridge, 1988.
- [2] P. Butlin, R. Long, E. Elmoznino, Y. Bengio, J. Birch, A. Constant, G. Deane, S. M. Fleming, C. Frith, X. Ji, R. Kanai, C. Klein, G. Lindsay, M. Michel, L. Mudrik, M. A. K. Peters, E. Schwitzgebel, J. Simon, and R. VanRullen. Consciousness in Artificial Intelligence: Insights from the Science of Consciousness, 2023. URL <https://arxiv.org/abs/2308.08708>. eprint: 2308.08708.
- [3] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling, 2014. URL <https://arxiv.org/abs/1412.3555>.
- [4] A. Clark. *Supersizing the Mind: Embodiment, Action, and Cognitive Extension*. Oxford University Press, 2008.
- [5] A. Clark and J. Kiverstein. Experience and agency: Slipping the mesh. *Behavioral and Brain Sciences*, 30(5–6):502–503, 2007.
- [6] D. Dong, S. Franklin, and P. Agrawal. Estimating human movements using memory of errors. *Procedia Computer Science*, 71:1–10, 2015. ISSN 1877-0509. doi: <https://doi.org/10.1016/j.procs.2015.12.174>. URL <https://www.sciencedirect.com/science/article/pii/S1877050915036352>. 6th Annual International Conference on Biologically Inspired Cognitive Architectures, BICA 2015, 6-8 November Lyon, France.
- [7] S. Franklin, T. Madl, S. Strain, U. Faghihi, D. Dong, S. Kugele, J. Snaider, P. Agrawal, and S. Chen. A LIDA cognitive model tutorial. *Biologically Inspired Cognitive Architectures*, 16, Apr. 2016. doi: 10.1016/j.bica.2016.04.003.
- [8] D. Hafner, J. Pasukonis, J. Ba, and T. Lillicrap. Mastering Diverse Domains through World Models, 2024. URL <https://arxiv.org/abs/2301.04104>.
- [9] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 11 1997. doi: 10.1162/neco.1997.9.8.1735.
- [10] R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82:35–45, Mar. 1960.
- [11] S. Kugele. Constructivist procedural learning for grounded cognitive agents. *Cognitive Systems Research*, 90:101321, 04 2025. doi: 10.1016/j.cogsys.2025.101321.

- [12] V. A. F. Lamme. Visual functions generating conscious seeing. *Frontiers in Psychology*, 11:83, 2020.
- [13] H. Lau. Consciousness, metacognition, and perceptual reality monitoring. *PsyArXiv*, 2019.
- [14] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning, 2016. URL <https://arxiv.org/abs/1602.01783>.
- [15] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel. Trust region policy optimization, 2017. URL <https://arxiv.org/abs/1502.05477>.
- [16] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms, 2017. URL <https://arxiv.org/abs/1707.06347>.
- [17] C. J. Whyte. Integrating the global neuronal workspace into the framework of predictive processing: Towards a working hypothesis. *Consciousness and Cognition*, 73:102763, 2019.
- [18] A. I. Wilterson and M. S. A. Graziano. The attention schema theory in a neural network agent: Controlling visuospatial attention using a descriptive model of attention. *Proceedings of the National Academy of Sciences*, 118(33):e2102421118, 2021.