AFRL-RY-WP-TR-2022-0103

ADAPTIVE LEARNING THROUGH ACTIVE NEUROMODULATION (ALAN)

Mario Aguilar-Simon, Andrew Brna, Ryan Brown, Larking Folsom, Jared Cook, Samuel Park, Alexandra Yanoschak, and Renee Shimizu

Teledyne Scientific & Imaging, LLC

JULY 2022
Final Report

STINFO COPY

AIR FORCE RESEARCH LABORATORY
SENSORS DIRECTORATE
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7320
AIR FORCE MATERIEL COMMAND
UNITED STATES AIR FORCE

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the Air Force Research Laboratory (AFRL) Public Affairs Office and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RY-WP-TR-2022-0103 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH THE ASSIGNED DISTRIBUTION STATEMENT.

//Signature//

ERIC D. BRANCH
Program Manager
Multi-Sensing Knowledge Branch
Multi-Domain Sensing Autonomy Division

//Signature//

DANIEL L. LAUBACK, Chief
Multi-Sensing Knowledge Branch
Multi-Domain Sensing Autonomy Division

//Signature//

ROY L. BALLARD
Division Chief
Multi-Domain Sensing Autonomy Division
Sensors Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

*Disseminated copies will show "//signature//" stamped or typed above the signature blocks.

REPORT DOCUMENTATION PAGE

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ORGANIZATION.

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED: START DATE	END DATE
July 2022	Final	23 March 2018	31 March 2022

4. TITLE AND SUBTITLE
ADAPTIVE LEARNING THROUGH ACTIVE NEUROMODULATION (ALAN)

5a. CONTRACT NUMBER	5b. GRANT NUMBER	5c. PROGRAM ELEMENT NUMBER
FA8650-18-C-7831	N/A	N/A
5d. PROJECT NUMBER	**5e. TASK NUMBER**	**5f. WORK UNIT NUMBER**
N/A	N/A	Y1SD

6. AUTHOR(S)
Mario Aguilar-Simon, Andrew Brna, Ryan Brown, Larking Folsom, Jared Cook, Samuel Park, Alexandra Yanoschak, and Renee Shimizu

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Teledyne Scientific & Imaging, LLC
1049 Camino Dos Rios
Thousand Oaks, CA 91360

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S)	11. SPONSOR/MONITOR'SREPORT NUMBER(S)
Air Force Research Laboratory, Sensors Directorate Wright-Patterson Air Force Base, OH 45433-7320 Air Force Materiel Command, United States Air Forces	Defense Advanced Research Projects Agency (DARPA/MTO) 675 North Randolph Street Arlington, VA 22203	AFRL/RYAP	AFRL-RY-WP-TR-2022-0103

12. DISTRIBUTION/AVAILABILITY STATEMENT
DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES
PAO case number AFRL-2022-3300, Clearance Date 13 July 2022. Report contains color.

14. ABSTRACT
Under the lifelong learning machines (L2M) program, Teledyne set out to research, implement and demonstrate algorithmic approaches to address two key problems. First, enabling intelligent agents to self-supervise in order to adapt and learn in complex environments without external intervention. To address this problem, Teledyne developed and validated the role of uncertainty tracking and modulation to enable agents to monitor their own performance and adapt with confidence when appropriate conditions exist. This is a significant breakthrough in that it demonstrated self-supervised learning and task performance by an embodied agent. The second problem was to enable robust knowledge representations that would remain accurate despite continual learning and adaptation and could accommodate the complexity of learning multiple tasks with likely differing requirements on knowledge granularity and composition. Teledyne developed and implemented a hierarchical learning system capable of decomposing task information across multiple layers to maximize robustness and re-use. This is a significant breakthrough as it enables a new class of learning systems that can maintain a consistent knowledge base and update it to accommodate multiple tasks without the requirements that they share a unified representation. The resulting algorithms were demonstrated to have a role in enhancing performance in state-of-the-art machine learning systems, and thus can be incorporated into many present-day AI solutions to equip them with lifelong capabilities. A key recommendation is to look for opportunities to transition these capabilities into existing AI systems, thus facilitating their transition to next-wave AI. Another recommendation is to see these accomplishments as a first step in elucidating lifelong learning mechanisms, and to engage in continued research to more fully understand how to achieve learning in highly complex environments and conditions. These will likely force us to look more closely at a more complete solution for building, maintaining and leveraging hierarchical knowledge representations.

15. SUBJECT TERMS
lifelong learning, continual learning, self-supervised learning, neural networks, machine learning, reinforcement learning, object recognition, drones

16. SECURITY CLASSIFICATION OF: a. REPORT	b. ABSTRACT	C. THIS PAGE	17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES
Unclassified	Unclassified	Unclassified	SAR	84

19a. NAME OF RESPONSIBLE PERSON	19b. PHONE NUMBER *(Include area code)*
Eric Branch	

PREVIOUS EDITION IS OBSOLETE.
STANDARD FORM 298 (REV. 5/2020)
Prescribed by ANSI Std. Z39.18

Table of Contents

Section **Page**

Section **Page**

List of Figures

Figure **Page**

List of Tables

Table **Page**

List of Equations

Equation **Page**

1.0 SUMMARY

1.1 Program Overview

Under the DARPA lifelong learning machine (L2M) program, Teledyne conducted a two-phase effort to develop machine learning systems capable of **selective plasticity.** Our effort addressed two critical challenges faced by a life-long learning system: the needs for (1) continuous but **stable learning** of its parameters, and (2) how to achieve **optimal capacity allocation** to obtain effective learning and performance as tasks and conditions change. Our core premise was that the brain solves both problems through **neuromodulation**: chemical signaling that continuously regulates neural activity and plasticity. Specifically, we investigated mechanisms by which the neuromodulator **acetylcholine (ACh)** regulates long-term synaptic plasticity and short-term synaptic activity, particularly in the visual pathway that performs object recognition and identification (ventral). We targeted ACh's role as a feedback signal encoding the level of **uncertainty** in both signal processing and inference; we explored how this signal regulates the computation and selection of low-level sensory features, while also driving learning of higher-level inferences.

These modulatory principles formed the core of our novel, plastic nodal network (PNN) architecture. Our PNN has a hierarchical structure that mirrors the two-stage organization of the brain's ventral pathway, and which is shared by other sensory pathways, such as the auditory and the visual localization (dorsal) pathways. Figure 1 provides a high-level overview of architectures for selective plasticity in hierarchical machine learning systems where heterogeneous layers are introduced to implement a continuum of dynamics to support optimal feature extraction and capacity allocation in early layers, while achieving stable and continuous learning in later layers. The following numbers in parentheses refer to the orange numbers in Figure 1. Modulation is driven by measures of uncertainty (1). Uncertainty derived by analyzing signals (bottom-up) and task requirements/rewards (top-down) are used to (2) influence feature extraction/selection in the early layers and inference in the later layers. The result of modulation in early layers is the rapid recruitment of specific portions of the capacity of the network (3), while in later layers, learning is more strongly modulated to ensure stability while maintaining appropriate plasticity for new or updated tasks (4): the network's early layers perform feature extraction (mirroring the occipital cortex), while the later layers compute inferences (matching prefrontal and temporal cortex processes). An ACh-like signal (measuring uncertainty) dynamically modulates computations and learning in the network. Our network is heterogeneous: different layers and types of nodes respond differently to the modulatory signal.

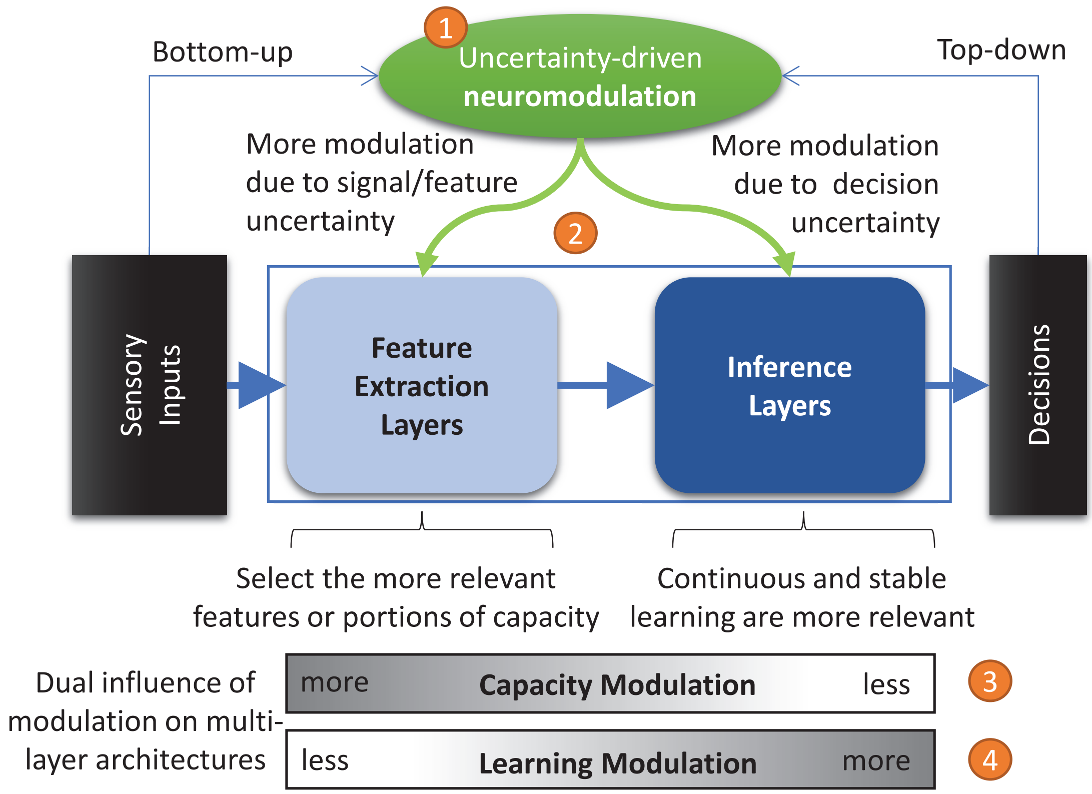


Figure 1. Architectures for Selective Plasticity in Hierarchical Machine Learning Systems

1.2 General Approach

Life-long learning requires constant adaptation; no amount of training can prepare a network, whether biological or artificial, for all the possible inputs that it might receive over its lifetime. In particular, ongoing learning requires the ability to change the network's parameters without forgetting prior information (i.e. **stable learning,** also known as the *stability-plasticity dilemma* [1]). Furthermore, a life-long learning system faces a second dilemma: the ability to continuously encode new information requires vast computational resources, but very large networks are intractable to optimize due to the huge number of free parameters. Figure 2 illustrates the scaling limits in the case of deep learning architectures. Ongoing research [2] suggests that deep learning networks cannot scale to arbitrary sizes, no matter how much data is used to train them. In particular, our own internal experiments under DARPA's TRACE program demonstrate that once a deep network exceeds an optimal size ([a] in Figure 2), its ability to learn decreases dramatically as it becomes larger ([b] in Figure 2). This implies that simply building larger deep networks and feeding them more data is insufficient to achieve human-level learning. In contrast, our modulated network recruits only a small subset of its nodes to optimize capacity (a), while carrying large overall capacity (b) allowing it to overcome this scaling limit. In contrast, a life-long learning system must manage its computational resources in a more intelligent manner to achieve **optimal capacity** allocation and mitigate performance degradation.

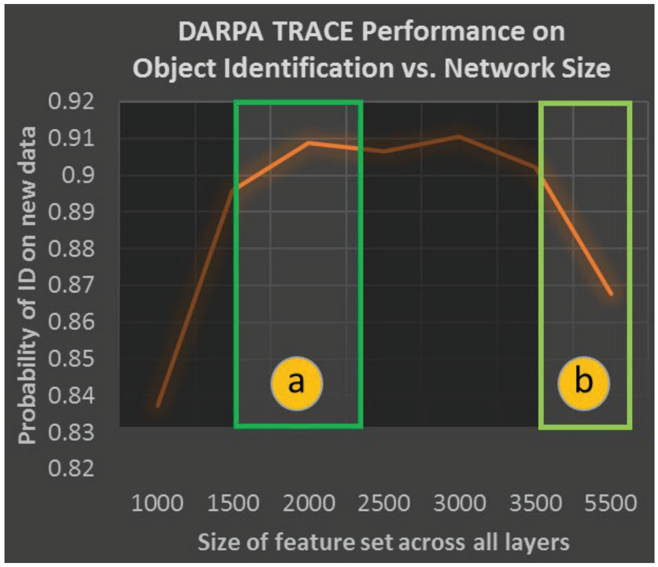


Figure 2. Deep Learning Scaling Limit

1.2.1 Theoretical Effort

Our foundational premise was that the brain achieves both capabilities through **neuromodulation**: the use of chemical signals that continuously regulate synaptic activity and plasticity. Among the many neuromodulators in the nervous system, ACh is one of the most extensively studied in the mammalian brain; it has been implicated in regulating several high-level cognitive functions, including attention, learning, and memory. More importantly, ACh regulates long-term synaptic plasticity and short-term neural activity levels, particularly in the ventral visual pathway (which performs object recognition and identification) [2]. ACh has been shown to encode **uncertainty**, specifically *expected uncertainty* [3] (as well as the related signal of *unexpected reward* [4]), which is a key feedback signal for triggering and regulating learning. In the ventral pathway in particular, ACh regulates the computation of low-level sensory features and drives learning of higher-level inferences.

As part of our effort, we developed a hierarchical, heterogeneous, plastic nodal network (PNN) algorithm called Uncertainty-Modulated Learning (UML) where neuromodulation-based computational properties enable optimization of the network's capacity to permit adaptive and stable learning (Figure 3). UML was modeled after cortical mechanisms of hierarchical sensory signal decomposition and inference, feedback attention, and neuromodulation in response to mismatched expectations. In UML, an ACh-like signal (triggered by measured uncertainty) dynamically modulates computations and learning. UML achieved several groundbreaking capabilities in machine learning, specifically:

- **Stable learning** that permits maximal updating without disturbing existing, learned behaviors (i.e., addresses the stability-plasticity dilemma)
  - Which, in conjunction with top-down feedback, enables **continuous** and **few-shot learning** of inputs and tasks that differ radically from previously learned information
- **Optimal capacity** allocation that selects and enhances only those features that maximize information content and that are relevant to the current task

- Leading to the **co-existence of multiple computational motifs** when the network is configured for hierarchical learning (i.e., UML can multiplex between different tasks or behaviors),
- as well as to **selective recruitment** of different subsets of the network at a time, allowing it to scale to an arbitrarily number of nodes (i.e. virtually unlimited capacity to learn new information)

UML represents a compelling new computational model for the role of local heterogeneous architectures, feedback signals and neuromodulation.

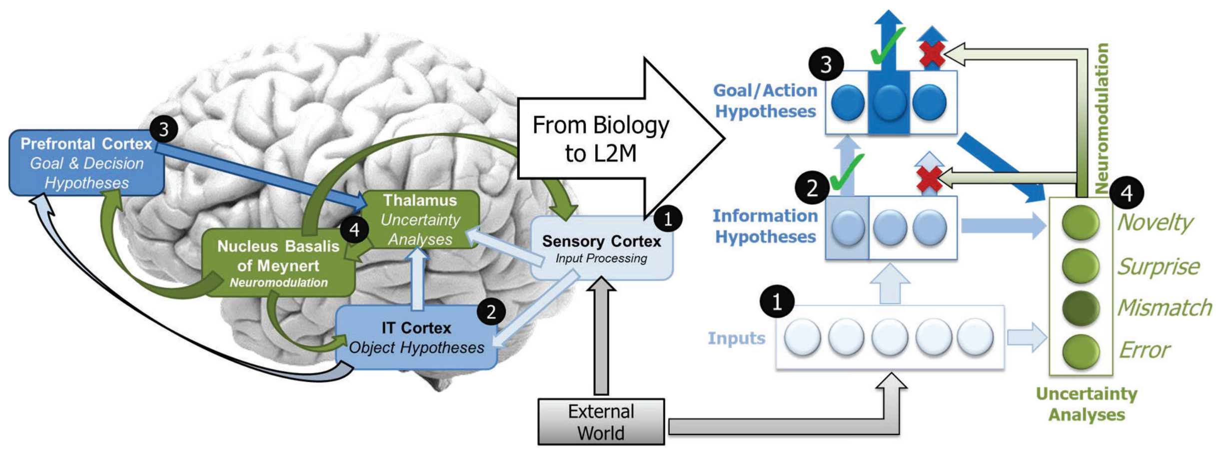


Figure 3. UML Algorithm Developed by Teledyne during L2M Phase 1

1.2.2 Experimental and Demonstration Efforts

Our work demonstrated algorithms and an integrated system with learning mechanisms capable of life-long learning in complex learning tasks. Additionally, we demonstrated that our UML algorithm is capable of imbuing other machine learning algorithms with the ability to adapt, learn without catastrophically forgetting and recover performance under out-of-nominal conditions. A summary of these results will be presented in Section 1.3.

During Phase 2 of the program, Teledyne led a systems group (SG) with the goal of integrating a full set of lifelong learning capabilities. To accomplish this, Teledyne defined a minimum set of capabilities relevant and aligned with our uncertainty-modulated continual learning paradigm (Figure 4, also see Section 2.2.1). Two L2M performers from Phase 1 of the program were invited to join our SG, University of California at Irvine who collaborated with researchers from the University of California at San Diego (UCI/UCSD) and Missouri S&T (S&T). Throughout Phase 1, Teledyne had developed and demonstrated algorithms for sensory signal processing that employ a bottom-up signal decomposition architecture to infer goal- and decision-relevant hypotheses (orange and blue blocks in Figure 4). Additionally, Teledyne began to demonstrate the use of attentional mechanisms to modulate learning and adaptation. S&T was recruited to bring their experience in this family of algorithms to jointly implement a system component inspired by brain mechanisms of top-down attention (green/yellow blocks in Figure 4). In collaboration with UCI/UCSD, we set out to investigate the role of sleep-inspired algorithms to optimize memory after task performance and consolidate memory (i.e., knowledge) across tasks (magenta and cyan blocks respectively in Figure 4).

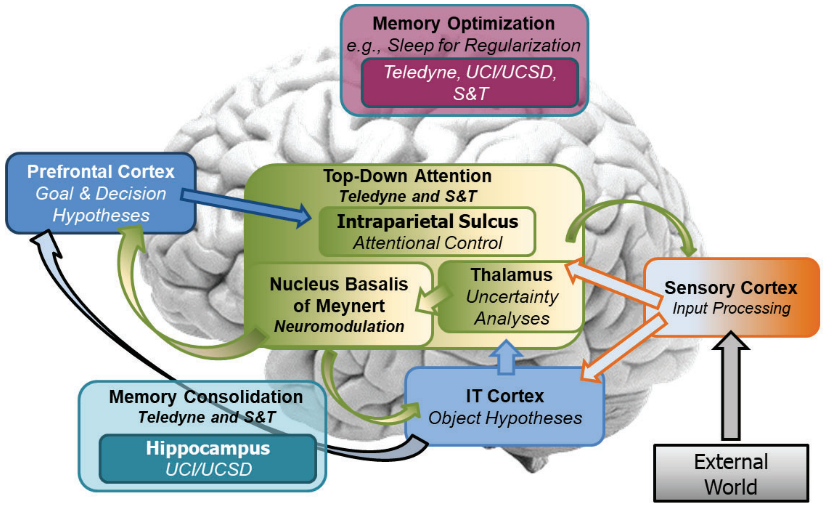


Figure 4. Key L2M Capabilities Integrated Based on SG Members-Developed Brain-Inspired Mechanisms

1.3 Overview of Results

The key premise of our proposed approach is that intelligent organisms measure and recognize critical changes in their environment, inputs, constraints or objectives, to enable them to adapt and learn without the need for external guidance (e.g., teachers, supervision, etc.). It is through such self-supervised monitoring and evaluation that a lifelong learning agent can be equipped to reliable function in complex and changing conditions.

Through our research and experimental work, we established that as in biological intelligent systems, measuring and tracking uncertainty serves as a key mechanism for triggering adaptation and learning. Our L2M agents were shown to either adapt their learned skills or incorporate new skills into their repertoire without catastrophic forgetting. We also demonstrated the agents' ability to leverage previous skills to improve learning efficacy (Forward and Backward transfer), quickly recover performance in the presence of interfering tasks or changes in conditions, leverage samples to adapt or acquire skills equally as efficiently or better than a single-task expert (see Sections 4.1-4.4).

Finally, Teledyne demonstrated the effectiveness of its integrated system through a series of milestone experiments conducted throughout Phase 2 of the program. These results are presented in Section 4.5 and highlight performance on L2M metrics in program-defined scenarios. These experiments served to create a steady tempo and coordination of results among all L2M SG teams, and document progress on performance. Additionally, we used them to identify successes and shortcomings of our system and/or algorithms. An analysis of the latter was leveraged to optimize our effort and focus system and algorithm development appropriately. The result was a consistent improvement of our system over the course of four milestone events from only meeting one of the metrics during the first event, to meeting all five by the fourth one. These results are also summarized in Section 5.0.

1.4 Major Conclusions and Recommendations

Our work throughout the program accomplished its main goals:

- deriving an effective algorithm inspired by biological mechanisms of neuromodulation
- implementing an algorithm with broad applicability to existing machine learning systems
- enabling intelligent agents that can self-supervise to adapt and learn continually
- integrating a system that exhibits mechanisms of attention, uncertainty-based regulation, hierarchical learning, and sleep-inspired memory optimization to demonstrate lifelong learning capabilities

A significant accomplishment of our work was the development of UML, a novel lifelong learning algorithm capable of self-supervising to adapt to new conditions, learn from few samples, and derive robust hierarchical knowledge representation. An exciting recent realization is the fact that the critical capabilities we set out to study and presented in our original proposal (see Table 1) were not only fully implemented but also thoroughly demonstrated throughout all the experiments and demonstrations in the program.

Table 1. Features and Benefits of Teledyne's Approach

Feature	Benefit(s)/Impact
Uncertainty Modulates Learning: We posited that neuromodulation upregulates learning for neurons that are critical to resolving distinctions between two or more classes.	Demonstrated that learned representations for new tasks do not overwrite previously learned tasks.
Uncertainty Modulates Capacity Allocation: We proposed to investigate the role of neuromodulation in upregulating activation and learning in portions of a network that can optimally solve a specific task and suppress those that do not contribute to reducing uncertainty.	Built networks with very large capacity to support life-long learning while not suffering from accuracy degradation by only activating portions of the network that optimally support task performance.
Uncertainty Triggers New Learning: By tracking expectations, new algorithms can adjust and improve their performance over time, especially when new tasks or conditions are introduced.	Demonstrated how learning is triggered when response certainty drops below desired thresholds leading to a system able to autonomously detect new tasks or conditions that require learning.
Uncertainty Modulates Feature Extraction: Measures of signal uncertainty across feature layers drive modulation of transfer function in early layers (feature extractors).	Implemented algorithms able to adapt feature extraction processing to compensate to changes in task, conditions, or signal properties.

In the Month 18 (M18) evaluation, the Teledyne SG showed results indicating that our lifelong learner met or exceeded the lifelong learning threshold in the five program metrics and exceeded targets in two of the five. This is shown in Section 4.5.1, Table 11, with light green indicating that a metric exceeds the lifelong learning threshold, and darker green indicating that a metric exceeds the DARPA program target.

One of the key insights we derived from our efforts is that uncertainty has proven to be an effective measure that supports online learning and the creation of robust knowledge representations without supervision or reinforcement signals. We also established that the L2 components we developed can be effectively integrated into existing ML system to support improved performance (e.g., robustness, adaptation, etc.). Hence, a significant number of transition opportunities exist (examples are discussed in Section 2.4). Teledyne will continue to pursue such opportunities through Government-funded efforts, commercial endeavors, and internally-funded research activities. Teledyne also welcomes any Government agency or individual to request discussions that could facilitate a deeper understanding or identification of transition opportunities.

Our UML algorithm proved to be an effective component (Section 2.3), not only for an integrated L2 system, but also as a plug-in to existing machine learning systems. These include end-to-end systems designed for decision support, where UML can monitor out-of-nominal conditions or flag conditions requiring additional samples or learning. UML was also demonstrated to support performance recovery under novel conditions for systems as complex as a reinforcement learning-based agent. Due to its lightweight processing requirements, UML can execute at 2000Hz on a commodity processor (CPU) and is thus amenable to deployment across many platforms.

2.0 INTRODUCTION

2.1 Program Objectives

A key objective of our work was to address two crucial problems for lifelong learning machines:

1. Enabling deployable L2Ms that manage resources, maintain performance, acquire new skills, and derive optimal strategies in the absence of external supervision signals. This with the goal of enabling machine learning systems that can operate on their own, not just based on prior knowledge, but constantly updating knowledge without human supervision.

2. Supporting flexible, robust, and task-effective knowledge representations in L2Ms that allow them to leverage past information and incorporate new experiences? This with the goal of addressing the challenge of learning skills requiring multiple levels of abstraction and that capture semantics of the task. Relevant examples of tasks using different levels of abstraction is learning to find a target of interest (e.g., a car or a treasure in a game) vs. learning to avoid objects that may represent threats (e.g., obstacles). Each task requires a different set of bottom-up signals (i.e., features) to be recognized and a different set of actions (e.g., report the car vs. avoid colliding with a wall, in the case of a drone conducting traffic monitoring missions).

Our work focused on conducting a systematic investigation on the computational properties that arise from mechanisms of neuromodulation, such as expressed in ACh modulation (see Figure 5 for an illustration). Among them are the influences that neuromodulation has in signal filtering, controlling learning (i.e., learning rate), and reconsidering information in the presence of significant uncertainty. Through experiments and demonstrations, we sought to empirically demonstrate the impact of these mechanism in learning and adaptive capacity utilization, most notably their ability to support **selective plasticity and self-supervision**.

ACh is perhaps the best studied neuromodulator in the mammalian brain. ACh is released from the terminals of neurons that originate in the brain stem and more commonly from the nucleus basalis of Meynert (NBM) in the basal forebrain. The NBM diffusely innervates and releases ACh in the cerebral cortex and the hippocampus where it exerts its action through two types of receptors: the nicotinic receptor that drives fast changes in ionic conductance and the muscarinic receptors that drive slower and longer lasting changes through second messenger systems (chained intracellular protein interactions). The actions at both receptors contribute to the modulation of plasticity and network tuning and are discussed below.

ACh has been demonstrated to participate in the regulation of several cognitive functions including attention, learning and memory. Pharmacologic block at muscarinic ACh receptors degrades the encoding (learning) of new memories, while drugs which activate nicotinic ACh receptors lead to enhanced memory formation [5-8]. Furthermore, increasing cholinergic modulation through stimulation of cholinergic neurons during perceptual learning has been shown to directly alter the receptive fields of sensory neurons and boost long term learning [9]. Also, Minces et al. [10] showed that ACh plays a crucial role in enabling rats to learn to discriminate fine-spatial features.

While cholinergic signaling is often associated with attention, recent research [4], has shown that ACh releasing neurons are most strongly activated in response to reward surprise. This finding highlights the potential importance of cholinergic modulation in reinforcement learning. Chubykin et al. [11] presented similar findings, in which they showed that ACh encodes a reward timing signal that drives plasticity in the visual cortex.

Figure 5 shows how ACh can induce a wide range of short-term changes to the behavior of neural circuits. The following numbers in parentheses refer to the orange numbers in within Figure 5. In the top row of Figure 5, (1) we zoom into a specific layer of our proposed architecture to reveal how (2) the input, which is processed by (3) excitatory nodes is regulated by (4) inhibitory internodes that affect (5) the output of the circuit. In the absence of ACh, the circuit computes a summation of its input. In the middle and bottom rows, different levels of ACh partially (6) or completely inhibit (7) the internodes, changing the output into contrast enhancement (middle row) or winner-take-all (bottom row), respectively.

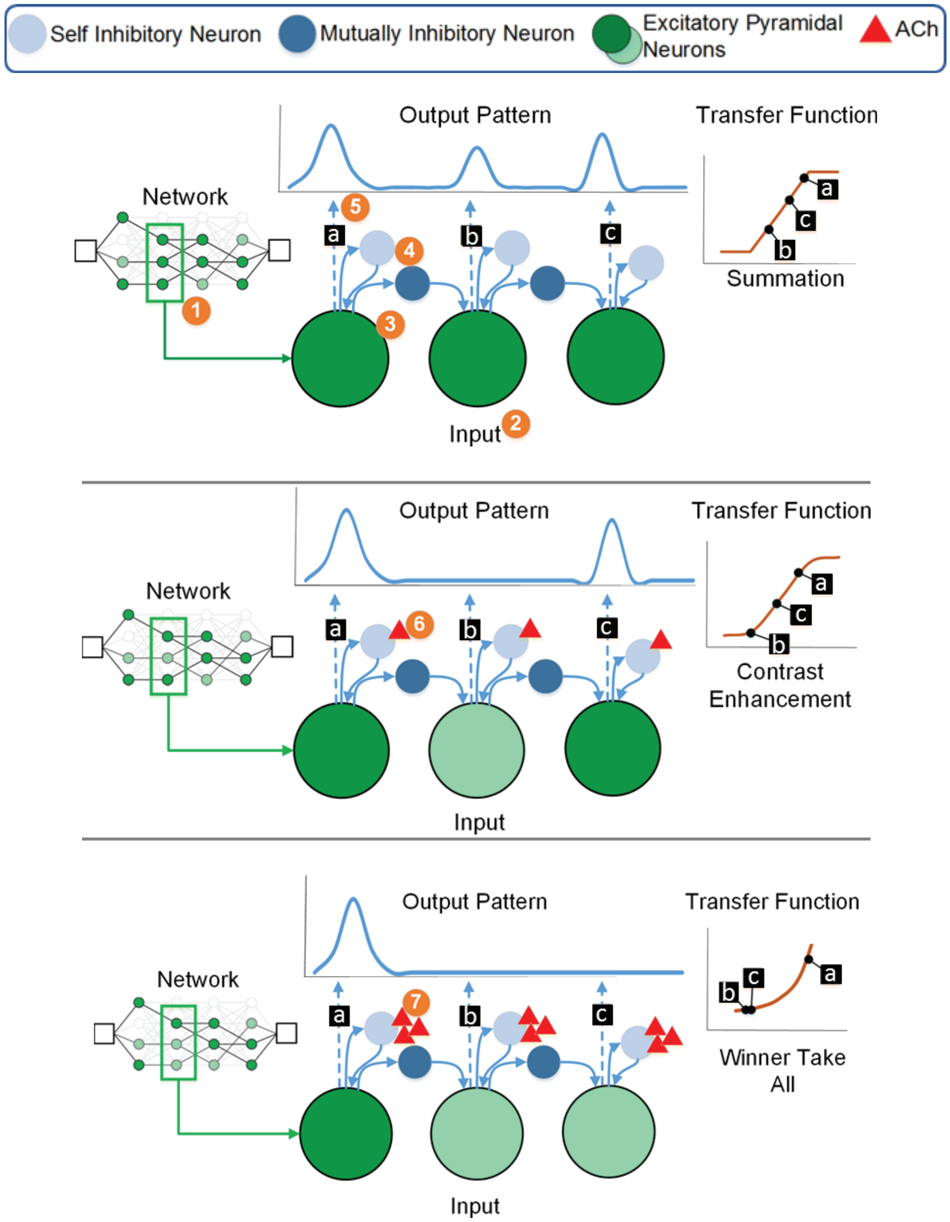


Figure 5. ACh Modulation of Local Circuit Computations

How can ACh, which is diffusely released across the cerebral cortex, precisely modulate learning? Within the mammalian brain the distribution of the two classes of receptors differs considerably from region to region and likely contributes to the modulation of learning in a spatiotemporally specific rather than global level (see [12] for review). In addition to the known differences in the expression of cholinergic receptors on different cell-types, recent research [13] has identified differences in the distribution of cholinergic receptors across cortex. Figure 6 (adapted from [13]) shows a gradient of receptor densities for the ACh muscarinic receptor (M2) across the hierarchy of visual areas (V1-V4). This gradient (i.e. high in V1 and low in V3) illustrates one example of a biological mechanism of cholinergic modulation. The gradient of receptor density enables varied and specific effects of system dynamics at different levels of processing. This supports the idea that differences in dynamics and computational outcomes depend on the level of modulation [13-15]. For example, the nicotinic receptors are most commonly found on inhibitory interneurons where their activation leads to increased lateral inhibition, but also enhancement of thalamic input (perhaps through disinhibition). This may result in a spatial sharpening and response gain of the incoming sensory input at short timescales. Alternatively, in the primate visual system, muscarninic (type M1) receptors are located primarily postsynaptically on inhibitory interneurons [15] and cortico-cortical connections [16] where they reduce top down and cross cortical excitation. Other muscarinic receptor subtypes (M2 and M4) are mostly found presynaptically where they produce an inhibitory effect on their target cells. Depending on the cell type this can result in net inhibition in pyramidal cells or net excitation by inhibiting inhibitory interneurons.

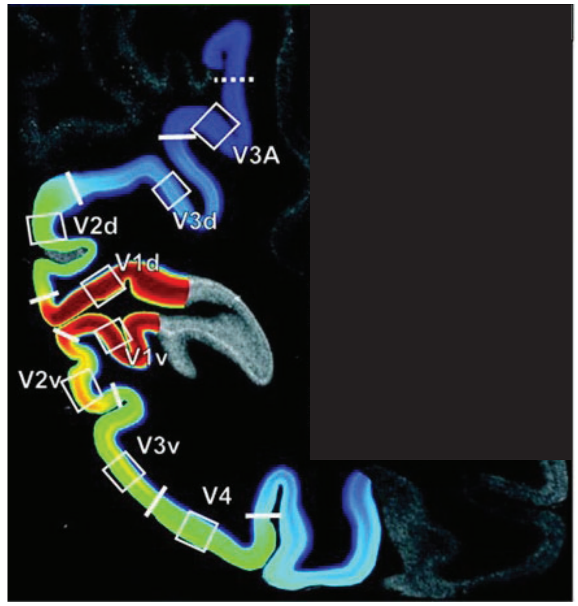


Figure 6. Gradient of Muscarinic Receptor Distribution in Visual Brain Areas

Taken together, the multiplicity of mechanisms for cholinergic modulation enable a wide range of behavior in concert with the non-homogenous network of the brain. As described above, these mechanisms can account for a significant amount of the varied behavior of neural circuits. A key contribution by Teledyne during this effort was to study and implement these heterogeneous mechanisms across multiple algorithmic scales (e.g., within layer “cell types” and cross-layer changes in modulatory “receptor expression”).

2.2 Technical Approach

2.2.1 Teledyne-Specific Algorithm Development

During both Phases 1 and 2 of the effort, we systematically studied and demonstrated the role of uncertainty in supporting robust continual learning in the presence of new conditions and tasks. In particular, we validated the use of biologically-inspired mechanisms to implement and demonstrate a hierarchical learning architecture capable of optimally self-managing its capacity while ensuring stable continuous learning. This included capturing the multiple roles of neuromodulatory processes mediated by acetylcholine on information processing and decision making by cortical circuits. Figure 7 illustrates the neural underpinnings of our architecture as applied to visual processing tasks and its role in processing input signals to generate action or decision-relevant signals.

Figure 7 shows a two-stage model of sensory feature extraction and category processing in the "What" pathway of the visual system. A first stage, mainly in Occipital Cortex, implements a hierarchical decomposition of signals for adaptive feature extraction, while a second stage, mainly in Temporal Cortex, performs category analysis and classification. Green and white nodes indicate neurons on a given layer. Signals from both stages are analyzed against expectations in non-specific thalamic areas (Thal) [6] and prefrontal cortex (PFC) [7] respectively. The presence of a mismatch triggers ACh modulation by projections from the NBM. The red arrows indicate possible targets for the influence of neuromodulation.

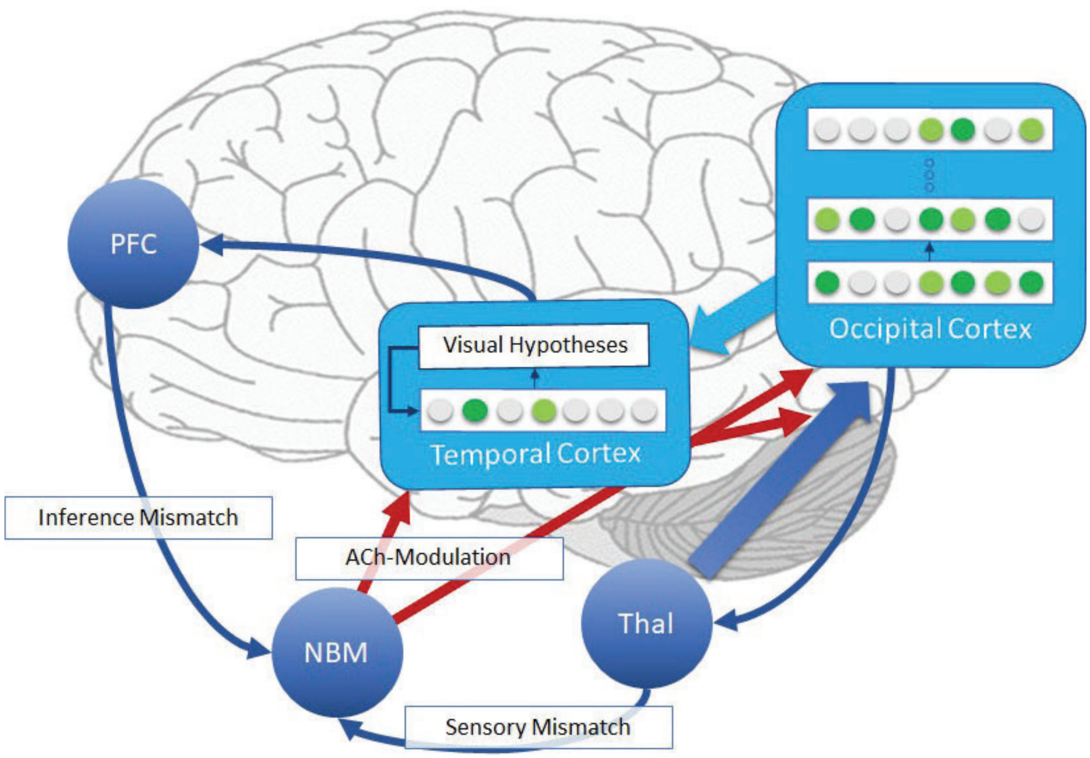


Figure 7. Model of Sensory Feature Extraction and Category Processing in the "What" Pathway of the Visual System

We successfully demonstrate the following key computational properties of our algorithms:

1. **Sensory processing is organized to achieve hierarchical aggregation** where raw sensory inputs are systematically processed and analyzed at increasing levels of abstraction [17]. Such processing is the most fundamental basis for modern backpropagation-based computational networks such as deep learning networks. However, sensory processing is also significantly modulated by complex dynamics that ensue as a result of both local and top-down feedback signals. At the local level, interneurons and lateral connectivity can be modulated to influence the transfer function and computation performed at each layer. These mechanisms can give rise to a number of properties that can be exploited in learning algorithms as will be described in the next section.

2. **Visual decision processing leverage bidirectional cortical circuitry** to achieve tasks such as object learning, categorization and prediction. Here, local and global interactions serve to stably learn and adapt categorization responses to visual stimuli. The details and richness of such mechanisms are not well modeled in modern machine learning algorithms such as deep learning. In the latter, this stage is typically modeled as a support vector machine (SVM) or multi-layer perceptron (MLP) responding to the activations of the learned features. Such algorithm choices have left a significant vacuum in understanding how to achieve both discriminative and generative classification, and achieve stable learning in the presence of continuously changing sensory information and task requirements.
3. **Generated sensory hypotheses in occipital cortex (e.g., V1-V4 activations) can be continuously reweighted, enhanced or suppressed** based on the match between the processed inputs and the expected or primed responses (e.g., the expectation of a car must be matched with an appropriate subset of activated feature responses as compared to those for a face). This allows a hierarchical system to adaptively bias processing to the appropriate abstraction level and/or allocate the subset of the network's capacity to maximally disambiguate signals and change weights.
4. **Generated category hypotheses (e.g., object ID/type/etc.) are analyzed in prefrontal cortex (PFC) to measure the uncertainty** of the inferred information/decisions and compare it against expectations. Such mechanisms can enable tracking task performance without supervision and triggering a recalibration that is weighted by the characteristics of the inference process, thus demoting portions of the signals that contribute to the error and promoting those that support the expectations.

2.2.2 System Group-specific algorithm and system development

As described in Section 1.2.2, Teledyne assembled a team of collaborators to develop and implement an integrated L2M system with five critical capabilities. Our system development and integration approach focused on realizing the critical functional roles of each of the L2M capabilities under development by the SG members. As Figure 8 illustrates, our L2M system integrated five functional blocks.

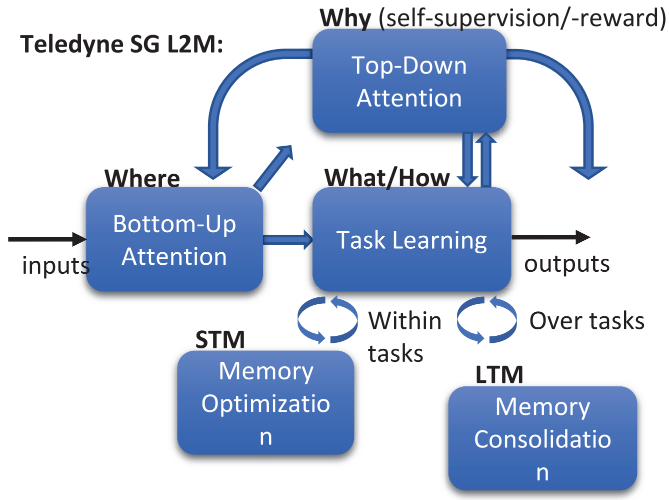


Figure 8. Functional Blocks of the Teledyne SG Integrated System

2.3 Key Accomplishments

We demonstrated that a heterogeneous architecture with modulatory inputs can: (1) selectively modify the subset of knowledge that is relevant for adapting or learning new information (*stable learning*), and (2) continuously modulate its constituent nodes to instantaneously modify their computational properties to adapt to unexpected signal or context properties. Figure 9 illustrates how modulation can reconfigure a network's behavior. In this simplified example, circles denote neurons, triangles represent a neuromodulator, arrows and square are excitatory and inhibitory connections, respectively, and dashes indicate that a connection is being suppressed. The neuromodulator suppresses the circuit's lateral inhibition, leading to a summation behavior; otherwise, the circuit behaves in a winner-take-all fashion. The result is that a single circuit (with the same learned weights) can quickly shift between different modes of operation by dynamic modulation in its connections.

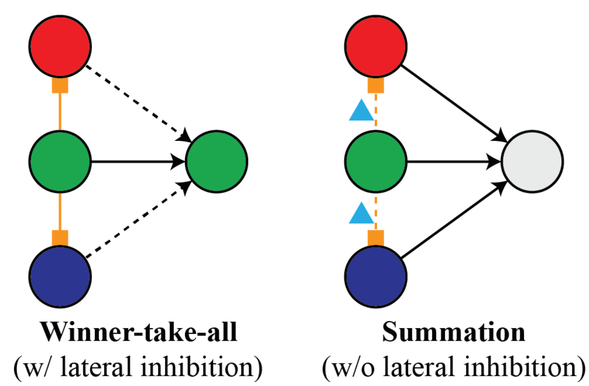


Figure 9. Neuromodulation Enables Neural Circuits to Quickly and Temporarily Change Their Dynamics

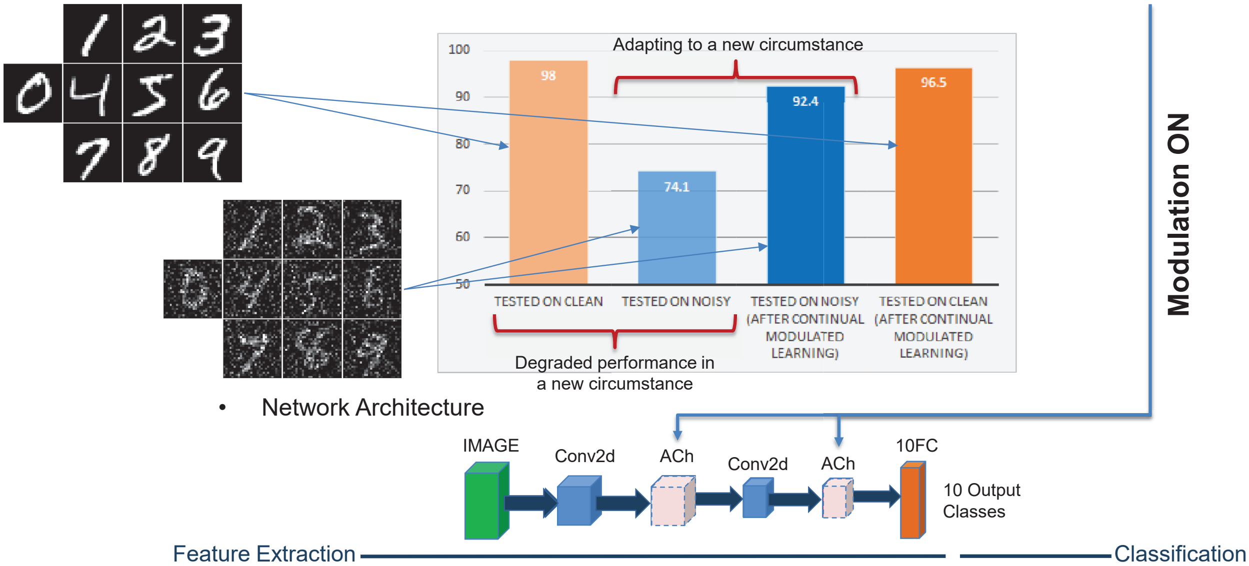
A key focus of our work was to develop algorithms and computational principles that would be broadly applicable to many machine learning problems and systems. Therefore, we demonstrated how an algorithm that integrates the proposed mechanisms of uncertainty, attention and memory optimization (UML, Figure 3) can support improved performance and lifelong learning capabilities across multiple machine learning domains (summarized in Figure 10). In particular, we were interested in demonstrating that UML could also be repurposed as a plug-in component to improve or expand lifelong learning capabilities to state-of-the-art algorithms. Figure 10 *a* and *b* present results of applying UML to learn and monitor out-of-nominal conditions to trigger adaptation in the underlying machine learning system (deep neural network trained to recognize digits and reinforcement learning agent trained to play Pong, respectively).

Additionally, the full set of mechanisms developed by our SG (Figure 5 and Figure 8), integrated in an embedded agent performing multiple tasks was shown to meet all L2M program metrics across multiple milestone experiments (described in Section 4). Figure 10c presents preliminary results of the integrated system we implemented in Phase 1, where the agent (a simulated drone) was able to self-supervise to adapt to changes in conditions (recognizing objects from the air after only pretraining on ground-based imagery), and discovering and learning new tasks (detecting and learning new object types despite not being in its existing repertoire). An important highlight is the fact that our SG was able to demonstrate consistent progress throughout the Phase 2 effort. While our system only met one program metric at the Month 12 (M12) milestone, it successfully met all five metrics by the M18 milestone.

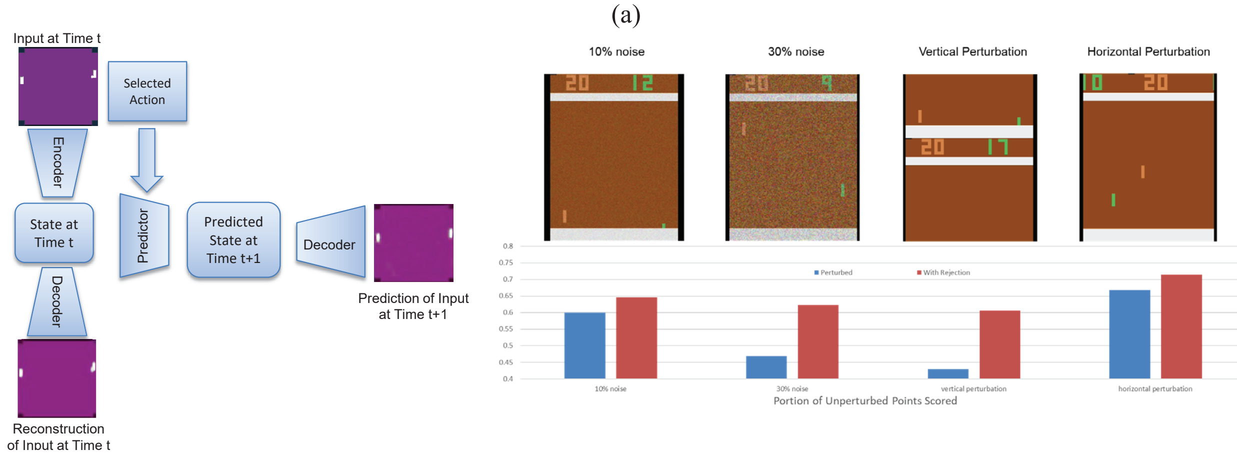
Figure 10 shows that UML can be applied to different machine learning problems. Panel (a) shows the role of UML in performance recovery for deep neural networks performing in out-of-nominal conditions (noisy MNIST samples). Panel (b) shows recovery of performance in RL algorithms operating in out-of-nominal conditions. Panel (c) shows self-supervised adaptation to new conditions, learning new tasks, and discovering new ones from a few samples.

Our work proposed algorithms and system architectures integrating the above mechanisms to address the problems presented in the beginning of Section 2.1: 1) robust self-supervised learning and 2) robust knowledge representations. The approach to address them was demonstrated throughout the program and consisted of:

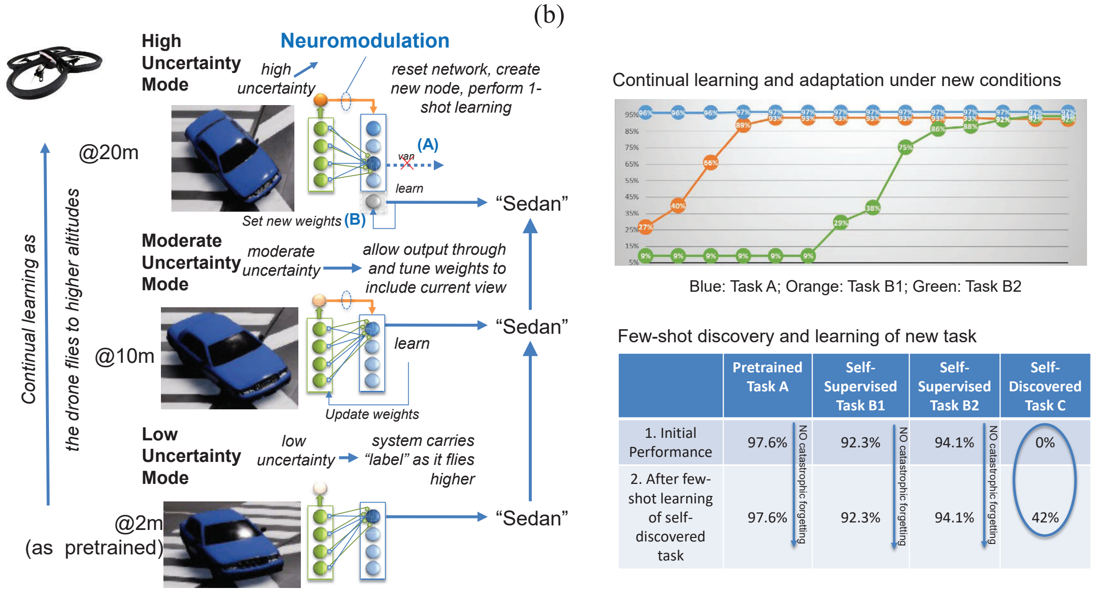
1. Tracking uncertainty to allow L2Ms to measure performance against learned expectation and trigger appropriate adaptation. The impact of this was a demonstration of self-supervision that frees machine learning systems from requirements of supervision or reinforcement signals external to the agent.
2. Developing a distributed and hierarchical learning system. Distributed learning was implemented by employing different components of the system to learn elements of the task separately but in coordination. Hierarchical learning was implemented by architecting UML into a multi-layer learning system to decompose tasks into multiple levels of abstraction. The impact of this work is the demonstration of knowledge representations that are more interpretable, capable of accommodating tasks with different levels of abstraction, and improved capabilities for transfer and maintenance (the latter demonstrated during our M18 experiments presented in Section 4).



(a)



(b)



(c)

Figure 10. Application of UML to a Variety of Machine Learning Problems

Finally, we prepared, submitted and/or published multiple publications listed below:

- Brna, A. P., Brown, R. C., Connolly, P. M., Simons, S. B., Shimizu, R. E., & Aguilar-Simon, M. (2019). Uncertainty-based modulation for lifelong learning. *Neural Networks*, *120*, 129-142.

- Kudithipudi, D., Aguilar-Simon, M., ... & Siegelmann, H. (2022). Biological underpinnings for lifelong learning machines. *Nature Machine Intelligence*, *4*(3), 196-210.

- Brown, R., Brna, A., Cook, J., Park, S., Aguilar-Simon, M. (Submitted) Uncertainty-Driven Control for a Self-Supervised Lifelong Learning Drone. *2022 IEEE International Geoscience and Remote Sensing Symposium.*

- Stephens, T., Corley, I., … & Aguilar-Simon, M. (Submitted) Self-Supervised Representation Learning Enhances Broad Area Search in Multi-Temporal Satellite Imagery. *2022 IEEE International Geoscience and Remote Sensing Symposium.*

- Petrenko, S., Brna, A., Wunsch, D., & Aguilar-Simon, M. (Submitted) Lifelong Context Recognition via Online Deep Clustering. *2022 International Conference on Machine Learning.* [In collaboration with Missouri S&T]

- Delanois, E., Brown, R., … & Aguilar-Simon, M. (In progress) Sleep-Inspired Replay for Lifelong Learning in Multi-Task Object Detection. [In collaboration with UCSD]

2.4 Transition Opportunities

We successfully evaluated and/or validated the use of our L2M algorithms across several non-program activities. UML was employed in two DARPA programs also under contracts managed by AFRL, Competency Aware Machine Learning (CAML) and Seeker Cost-Transformation Closed-Loop (SECTR-CL) evaluations. Additionally, UML was evaluated for insertion into several Government-funded projects that are successfully transitioning ATR algorithms for SAR imagery developed under DARPA's Target Recognition and Adaptation in Contested Environments (TRACE) program, also originally managed by AFRL.

Figure 11 presents results obtained in our experiments with UML replacing the target classification stage of a state-of-the-art synthetic aperture radar (SAR) automatic target recognition (ATR) system (first developed under the TRACE program). The plot on the left shows that UML has a significant impact on the area-under-the-ROC (receiver operating characteristic) curve for test sets that included novel targets and conditions. Condition changes included time of year, geographical regions for the target sites, and background clutter. Novel targets included pre-trained targets in new configurations or completely new targets (i.e., confusers). The plot on the right demonstrates how in addition to improving ATR performance, UML can report the uncertainty registered for every target under each of the conditions, thus supporting more informed decisions for self-supervision or for operator awareness. We suggest that both of these capabilities are excellent targets for the use of UML in any existing ATR system that is susceptible to day-1 failures in the field due to novel conditions not anticipated during training.

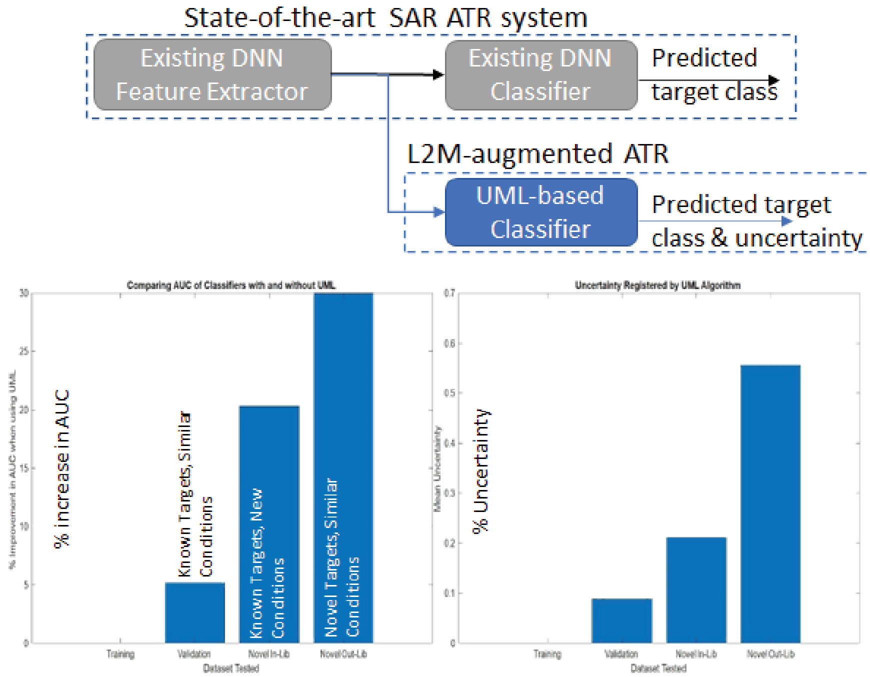


Figure 11. UML Improves Performance of a State-of-the-Art SAR-Based ATR Algorithm in Novel (Out-of-Nominal) Conditions

Figure 12 presents results obtained in our experiments with UML applied to predicting the performance of an existing machine learning algorithm under the DARPA CAML program. Here, UML was used to learn the patterns of activation across different layers in the machine learning system in response to presentation of images in the training set post-training (top diagram in the figure). In our main experiment, UML was trained on a concatenated vector composed of the activations in layers 3 through 5 of the deep neural network. Through this process, UML learned to characterize the nominal responses for the training set, which we coined *strategies* in the CAML program. Then, a novel dataset was used to generate both activations and target classification outputs. UML used the activations to predict the probability of correct classification by the network on this novel dataset. As the plot on the bottom of the figure suggests, UML could predict network performance about 80% accuracy. These experiments were terminated early so there was not an opportunity to investigate additional means of increasing accuracy. However, the results suggest the promise that UML, without access to ground truth, can predict a pre-trained network's performance in the field, a feat not possible today unless ground truth is available.

As these results demonstrate, our UML algorithm can be used as a drop-in replacement in any system/program where either supervised, unsupervised or semi-supervised algorithms are being used, which will equip the underlying system with a continual and self-supervised learning capability. We assess the TRL for this algorithm at 4-5. Our stand-alone uncertainty tracking algorithm can be used to inform the performance of any machine learning algorithm or system that relies on machine learning algorithms. Furthermore, such systems can be modified to use information about uncertainties to adapt their execution. We assess the TRL for this algorithm at 3-4. Finally, both of the algorithms above, when integrated in a system for object recognition, can support capabilities for rapid adaptation to new environments or the addition of new object

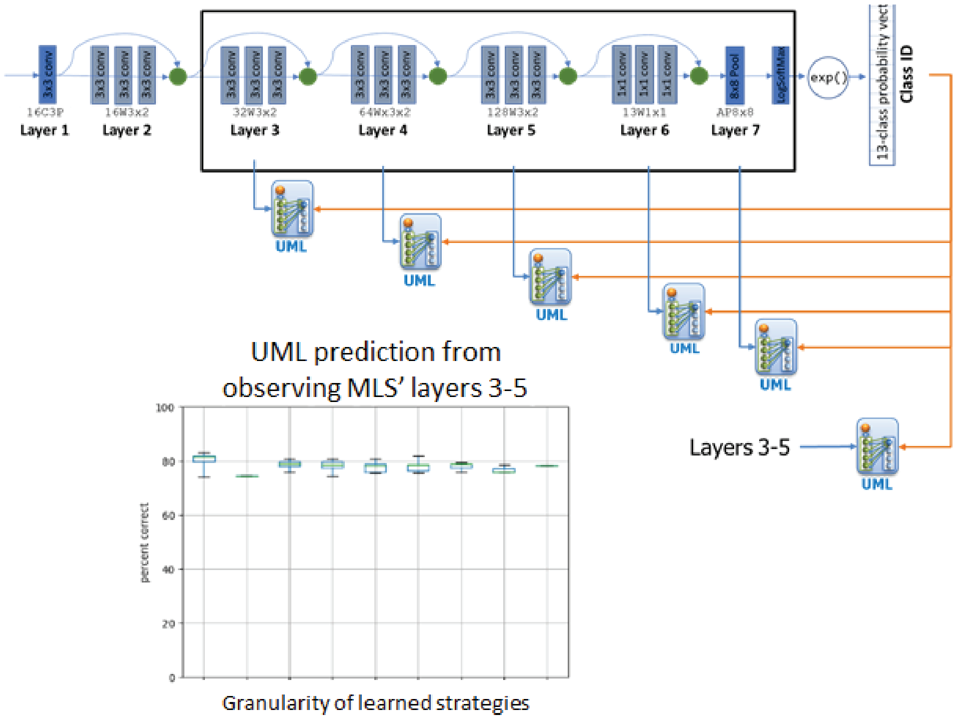


Figure 12. UML Applied to Predict Performance of an Existing Machine Learning Algorithm

types or variations. The example in Figure 13 illustrates a successful experiment we conducted to investigate the insertion of UML into a commercial product under development at Teledyne. The system uses a camera to detect and classify recyclable material on a conveyor belt. The initial object classification algorithm performs very well (~90%) on a small set of object classes. However, when the number of classes was increased to 27, performance degraded substantially. UML was used to replace the classifier resulting in an accuracy of ~88%. In addition to the increased performance, UML's lifelong learning capabilities allow users to add new object

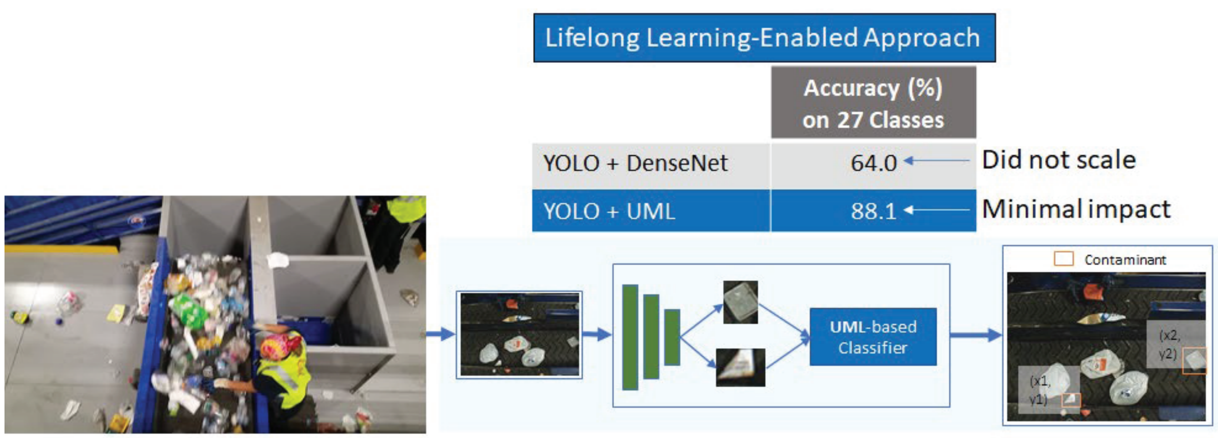


Figure 13. UML Applied to Classify Objects in Complex Environments Outperforms State-of-the-Art Object Classifiers

classes (i.e., new tasks) on-demand. We assess the TRL of this system at 6-7 as it was tested under operationally relevant conditions.

As the previous example demonstrates, our L2M algorithms can be plugged into new or existing machine learning solutions or systems. They can be integrated as part of individual components of a solution or integrated into the various components of a complete solution. Table 2 lists several of the transitions facilitated through the L2M program.

Table 2. List of Transitions and Future Opportunities

Element	TRL	Link to publication or repository	Details
Components: UncertaintyPropagation.jl ObjectDetector.jl	4	https://github.com/TDYbrownrc/UncertaintyPropagation.jl https://github.com/ldfolsom2/ObjectDetector.jl	Julia modules for uncertainty propagation and object detection
UML learns context and nominal conditions to trigger alerts and promote adaptation	4	https://www.sciencedirect.com/science/article/pii/S0893608019302722	Current DARPA programs candidates for transition ACE, SESU
UML to Astrocyte	7	https://www.teledynedalsa.com/en/products/imaging/vision-software/astrocyte/	Transition of algorithm into Teledyne product
UML to Optical Sorting	6-7	Commercial application	Demonstrated robust performance in the presence of increased tasks relative to the state-of-the-art
UML in other Government programs	4-6	CAML, SECTR, TRACE-related transition programs across multiple agencies	"Plugged" UML to predict performance of pre-trained models on novel data, without labels → 80% accuracy; Increased baseline accuracy by up-to 30% in novel conditions.

3.0 METHODS, ASSUMPTIONS, AND PROCEDURES

3.1 Methods

3.1.1 L2M Architecture

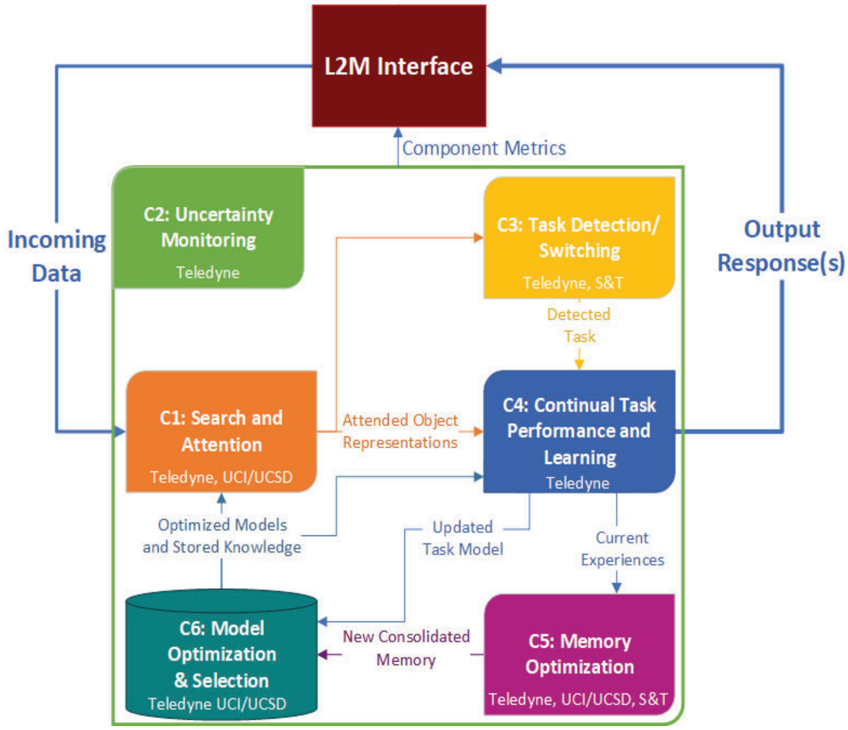


Figure 14. Teledyne's L2M Architecture

Our chief pursuit over the course of this project was to develop and ultimately evaluate algorithms capable of lifelong learning following principles of neuromodulation. Our approach in Phase 2 utilized a **sophisticated, multi-component architecture**, which is shown in Figure 14. Our L2M architecture allowed the sequestration of key operations deemed necessary for lifelong learning, which were brought together by the end of the program to produce an embodied learning agent capable of **self-supervision, uncertainty management, resource optimization, and on-the-job learning within an open environment**.

Our L2M architecture consists of six named components (denoted by C#) and an interface layer. The named components and their roles in the system are as follows:

- C1: Search and Attention – C1 accepts raw inputs provided by a given application through the interface layer and extracts meaningful features for subsequent evaluation in other system components. Additionally, C1 provides information on potential regions of interest within a given input.

- C2: Uncertainty Monitoring – C2 tracks uncertainties generated within system components and communicates them across the system. This component enables neuromodulatory activities across components, promoting different modes of operation in response to uncertainty.

- C3: Task Detection/Switching – C3 provides top-down feedback within the system by identifying the active task during system operation and deriving contextual information. Its outputs subsequently drive task-specific attentional mechanisms in other components to enable context-specific operations. This component was developed by the SG team member Missouri University of Science and Technology (S&T), and it is driven by distributed dual-vigilance fuzzy (DDVFA) Adaptive Resonance Theory Mapping

Algorithm (ARTMAP) [18] and is itself capable of online continual learning without catastrophic forgetting [publication under review].

- C4: Continual Task Performance and Learning – C4 utilizes bottom-up extracted features, top-down contextual information and learning signals, and calculated uncertainties to learn multiple tasks in a self-supervised manner. This component houses the neuromodulation-based uncertainty-modulated learning (UML) algorithm [19] developed by SG team leader Teledyne during this program, and it serves as the learning centerpiece of the architecture (see Section 3.1.2 for more details). Inputs are evaluated in a hierarchical and online manner to identify appropriate system actions under changing circumstances, and behaviors and learning are modulated according to available uncertainties, including denoting high-uncertainty samples as "unknown".

- C5: Memory Optimization – C5 evaluates the knowledge bases (e.g. learned weights) accumulated by other system components during online operation and optimizes their organization and storage. Such consolidation both reduces the amount of memory required by each component, promoting long-term operations, and improves task performance through better arrangement of information.

- C6: Model Optimization & Selection – C6 coordinates the use of separate knowledge bases corresponding with the active context. Additionally, it handles offline modification of knowledge bases, including batch processing of samples accumulated over some duration. Algorithms to power this component were developed and tested by SG team members University of California, Irvine (UCI) and University of California, San Diego (UCSD), though they were not integrated into the architecture in time for program evaluations.

- Interface: While explicitly a component of the learning architecture, the interface component handles communications into and out of the architecture. Its inclusion allows the L2M architecture to be application-agnostic, promoting its use as a plug-and-play algorithm in any system.

3.1.2 Uncertainty-Modulated Learning (UML)

The learning algorithm used in this work expands upon principles of uncertainty pioneered by Grossberg's Adaptive Resonance Theory (ART) algorithm [20]. In ART, uncertainty is best embodied in the vigilance parameter, which controls how similar a sample's bottom-up input needs to be to an existing top-down expectation to incorporate the sample into it. If the similarity metric does not pass the vigilance criteria, then the system is considered uncertain, and it engages a different learning mechanism to introduce the sample into the knowledge base. Our main contribution is to introduce specific mechanisms for measuring different types of uncertainty that an embodied agent needs to be able to track to perform robustly under a variety of conditions. Through an extensive review of analysis of evidence, we have identified a broad number of additional forms of uncertainty that we incorporate into a new uncertainty-modulated learning (UML) algorithm. Furthermore, we expand the mechanisms that trigger changes in the learning system so that new information is incorporated in a fully self-supervised manner.

Figure 15 illustrates the generalized form of the UML algorithm. The algorithm accepts high-dimensional representations of data, and it makes an initial decision/hypothesis using that data based on its own existing knowledge base. The hypothesis, the end point of traditional machine learning techniques, could take different forms based on the nature of the inputs; in this work it represents a classification, but it could also represent a translation, a prediction, a diagnosis, or even a motor action. The uncertainty algorithm follows the initial hypothesis by calculating a set of uncertainty metrics, which represent sources of noise or confusion in the signals or decisions/hypotheses that may have an impact on the algorithm's output. Uncertainty can come from a variety of sources, including but not limited to the hypothesis, the internal representations of the existing knowledge base, the inputs themselves, and the conditions under which the inputs were received. Figure 15 presents three such uncertainties.

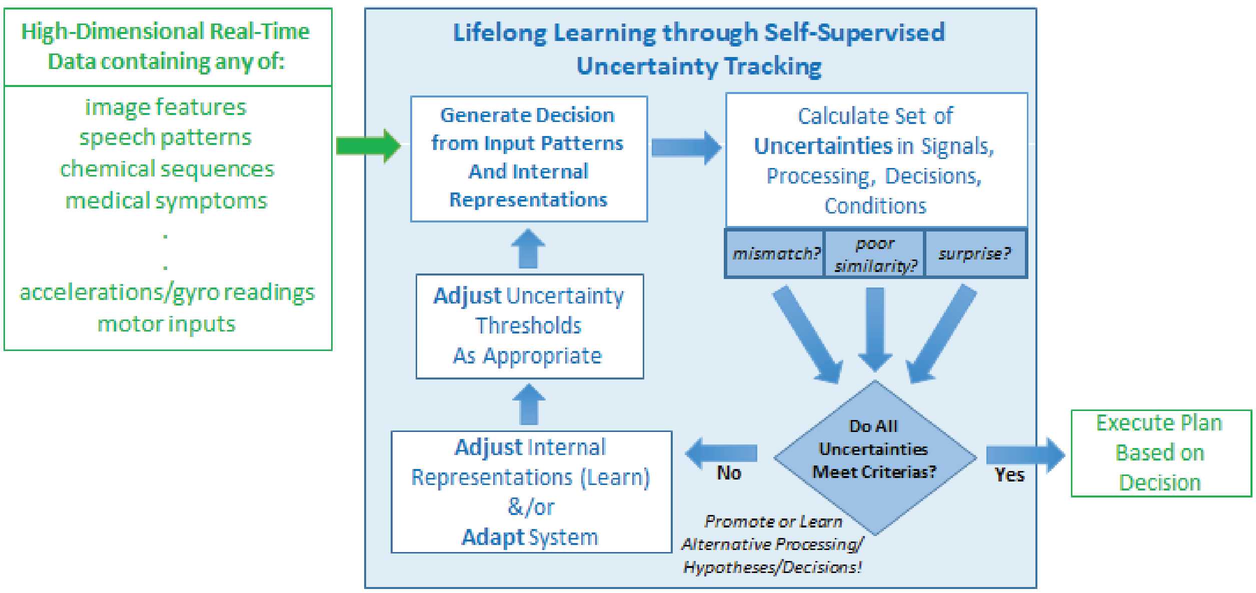


Figure 15. Generalized Algorithm for the Uncertainty-Modulated Architecture

Next, the uncertainty metrics are compared against internal criteria representing the algorithm's tolerance for each uncertainty type. If all uncertainty criteria are met, representing a confident hypothesis, the architecture finalizes its decision and passes it on to downstream components of the system. However, if any of the uncertainties do not pass criteria, then the algorithm changes, or modulates, its operations based on the specific uncertainty failure. Changes can be temporary, lasting only for one input, or permanent, changing the knowledge base for all subsequent inputs. Finally, the algorithm adjusts its criteria as appropriate for the failed uncertainties, and a new hypothesis is generated. This process repeats until the algorithm finds or develops a hypothesis that satisfies all uncertainty criteria.

Changes resulting from uncertainty are mediated by modulation mechanisms inspired by the acetylcholine neuromodulatory system [21, 22] and norepinephrine [23]. These neuromodulators can be released in response to expected uncertainty and surprise, triggering multiple effects on signal processing and neuroplasticity in multiple brain regions [4, 24-26]. Some of those effects induce temporary adaptations in cortical networks [27], and others induce extended changes in hippocampal memory [28]. These neuromodulatory mechanisms form the basis by which the UML algorithm can incorporate new information into its knowledge base without destroying its prior knowledge.

3.1.2.1 Types of Uncertainty

The UML algorithm measures uncertainty with the principal goal of confidently adapting to the environment or task changes. Specifically, uncertainty allows the algorithm to monitor its performance against expectations and respond with the appropriate form of adaptation or response. UML currently evaluates five types of uncertainty: detection, category fit, similarity, relevance, and persistence. The exact sources of each type will vary among algorithm implementations, but as a whole, they represent critical questions that a continual learning system must ask to determine when, what, and how it should learn. Detection relates to the uncertainty that a system has appropriately detected an object of interest. Fit reflects uncertainty in how closely the inputs match internal representations of the knowledge a system has already learned; ART examines this type of uncertainty to an extent through its vigilance parameter ρ and its "reset and search" and "match tracking" mechanisms. Similarity asks how well a sample relates to everything the system has already experienced, i.e., does this sample share any common ground with what has already been seen, or is it something entirely new? Relevance is uncertainty in the relationship between the learned information (e.g., objects it knows) and the current input. For instance, whether the input is related or similar enough to the classes of inputs it has previously learned. Lastly, persistence relates to temporal or observational uncertainty, as it examines consistency in both the knowledge to be learned and the system's understanding of that knowledge.

Since a principal mechanism for measuring uncertainty in our algorithm is to compare the inputs or feature activations against learned expectations, it can readily incorporate any machine learning method that can learn priors (e.g., Bayesian networks). Algorithm 1 gives the variant modeled after default ARTMAP 2 for illustration purposes, which can be leveraged for classification tasks. Parameters related to Algorithm 1 are listed in Table 3, and rationale for ARTMAP's equations and parameters are described in the associated article [29]. This variant examines five types of uncertainty, compares them to criteria Ψ_i, and triggers modulation to control processing flow accordingly. Uncertainty criteria Ψ_i are given values at the start of operation but can be modulated during operation to promote adaptation under different processes.

Throughout the L2M program, we demonstrated the use of UML across multiple domains and applications. For each application, values for parameters Ψ_i can be derived through a linear programming procedure to optimize task performance. In order to optimize for continual learning, optimization is performed on every sample on the training set (in contrast to collecting statistics across the entire dataset). We found that for optimization it is critical that sample order is preserved, as this reflects temporal dependencies in natural sensor streams.

ALGORITHM 1

ARTMAP 2 variant of UML

1. Initialize model and uncertainty criteria $\Psi_{1\text{-}5}$
2. For each frame, generate *d* detections with features *F* and objectness through object detector
3. For detections $i = 1\ldots d$, perform the following:
    3.1. Evaluate detection metrics against detection criteria Ψ_1
        3.1.1. If detection does not pass criteria, continue to next detection
    3.2. Evaluate complement-coded features $A=(F,F^c)$ against category fit criteria Ψ_2 using the following process
        3.2.1. for each network node $j = 1....C$, calculate node activation levels $T_j = |A \wedge w_j| + (1 - \alpha)(M - | w_j |)$
        3.2.2. collect activated node subset $\Lambda = \{\lambda = 1\ldots C : \; T_\lambda > \alpha M\}$
        3.2.3. set winning node $J = \text{argmax}(T_\Lambda)$
        3.2.4. evaluate node activations T_J against category fit criteria Ψ_2
            3.2.4.1. set vigilance ρ = category fit criteria Ψ_2
            3.2.4.2. if T_J does not pass vigilance ρ, remove *J* from Λ and return to step 3.2.3 [reset]
            3.2.4.3. if T_J passes vigilance ρ, assign an initial label from node *J* and attempt to learn
                3.2.4.3.1. if supervisory label is provided & node J's label does not match,
                    3.2.4.3.1.1. modulate criteria Ψ_2 to reject label mismatch [match tracking]
                    3.2.4.3.1.2. remove *J* from Λ and return to step 3.2.3
                3.2.4.3.2. otherwise, update node weights w_J to incorporate detection *i*
                    3.2.4.3.2.1. node weights $w_J = \beta(A \wedge w_J) + (1 - \beta)w_J$
                    3.2.4.3.2.2. proceed to step 3
        3.2.5. if no nodes pass criteria Ψ_2, examine similarity of features *A* to existing nodes
            3.2.5.1. for top nodes in original node subset Λ, calculate overlap(features *A*, node weights w_j)
            3.2.5.2. for first node *J* that passes similarity criteria Ψ_3,
                3.2.5.2.1. otherwise, update node weights w_J to incorporate detection *i*
                    3.2.5.2.1.1. node weights $w_J = \beta(A \wedge w_J) + (1 - \beta)w_J$
                3.2.5.2.2. proceed to step 3
        3.2.6. if no nodes pass criteria Ψ_3, self-generate new category to hold novel sample
            3.2.6.1. create new label $N = n + 1$ [new class]
            3.2.6.2. create new node with label = *N*, $w = A$
    3.3. store current category in memory
    3.4. if number of detections for label N or physical object remains below relevance criteria Ψ_4 for too long,
        3.4.1. remove label N or physical object from consideration and learning
    3.5. if number of detections for label N or physical object passes persistence criteria Ψ_5 is insufficient,
        3.5.1. remove label N or physical object from consideration and learning
    3.6. finalize category as hypothesis and request human-relevant label if a new label was self-generated
4. continue to next detection (*i++*)

Figure 16. Algorithm 1

Table 3. Table Describing Parameters for the UML Algorithm

Notation	Parameter	Possible Values
Ψ_1	uncertainty criteria for detection	(0, 1)
Ψ_2	uncertainty criteria for category fit	(0, 1)
Ψ_3	uncertainty criteria for similarity	(0, 1)
Ψ_4	uncertainty criteria for relevance	(0, 1)
Ψ_5	uncertainty criteria for persistence	(0, 1)
C	number of nodes in system	(0, ∞)
n	number of labels known to the system	(0, ∞)
a	feature vector	(0, 1)
A	complement-coded feature vector	(0, 1)
M	number of complement-coded features	--
T	node activation level	(0, 1)
Λ	subset of activated nodes	(1, C)
J	winning node	(1, C)
w	node weights	(0, 1)
α	signal rule parameter	(1, C)
β	learning fraction	(0, 1)
ε	match tracking	(-1, 1)
N	self-supervised label (class)	(1, n)
H	final hypothesis	(-1, n)

Each criteria Ψ_i relates to a specific type of uncertainty which arises in a classification learning system, and their values reflect the type of uncertainty they are designed to address. Criteria Ψ_1 relate to detection confidence, and controls whether specific objects are considered further ("objectness", [30]). Criteria Ψ_2 and Ψ_3 relate to uncertainties in the analysis of the inputs, specifically confidence of a classification or hypothesis given pre-existing knowledge. Finally, criteria Ψ_4 and Ψ_5 relate to uncertainty in the observations. The former establishing the relatedness of the object to previously learned objects and the latter establishing the permanence or consistency in the observation of any new objects.

The algorithm begins by creating or loading an ARTMAP-based network and setting the uncertainty criteria. A separate detector (C1) finds objects in a scene and returns multi-dimensional features and an initial hypothesis, which are then fed to the network one-at-a-time. The first stage of modulation subsequently occurs in step 3.1, as the detected object is examined against two uncertainty criteria and potentially rejected as poor quality or undesirable.

In Steps 3.2-3.6, object instances generated by the detector of Step 2 are processed to infer category label and engage continual learning and adaptation. Steps under 3.2.4 embody the core of ARTMAP, which is modified to evaluate multiple uncertainty criteria. Features are first complement-coded, and if no nodes are available in the network, the one is created using those features and a corresponding supervised/unsupervised label. ART then implements a "forward pass" on the network and generates a level of activation T in each available node. ARTMAP uses the node activations to find the closest approximation "winning" node J, and the fit T_J of the input features to node J's calculated weights is compared against the vigilance parameter ρ. The vigilance parameter ρ is modulated by three of the uncertainty measures (Ψ_2, Ψ_3, and Ψ_4) to enable self-supervised learning under appropriate levels of uncertainty (Step 3.3-3.4). Ψ_5 is used to determine if sufficient evidence (observations) have been collected for a newly acquired object (Step 3.5). If this criterion is met, then the system preserves the new object class and can make it available for analysis by a human operator who can assign a human-readable label (Step 3.6).

During lifelong learning, if node J satisfies criterion $\Psi_{2\text{-}3}$, and the provided label matches node J's label if supervised, then the network incorporates the input features into node J's weights using the following the equation in Step 3.2.4.3.2.1. A node's weights can be considered the node's "internal representation" or template of the part of feature space it has learned to recognize. However, if the node J does not pass criterion Ψ_2, UML engages a "reset and search" to select a new node for evaluation. This allows the algorithm to modify the network's knowledge base intelligently by focusing attention on the information relevant to the most likely hypothesis (i.e., category). If the Ψ_2 criterion is passed, but node J's label does not match the assigned label, then the algorithm performs match tracking, increasing the value of Ψ_2 before starting reset and search. In this way, the algorithm is self-supervised, adjusting its uncertainty criteria automatically. Both match tracking and reset and search constitute modulatory mechanisms.

Once the algorithm has determined an appropriate category for a sample (including "I don't know"), the algorithm places that category into a list containing candidate-values for a specific object over time in Step 3.3. Categories in this list are replaced with non-detections (zeros) over time, so if the list does not contain enough recognitions of an object to pass criteria $\Psi_{4\text{-}5}$, then there is uncertainty of the object's permanence, and the current detection is rejected (Step 3.4). If criterion Ψ_4 is passed, but the frequency of the most common hypothesis in the list is below criterion Ψ_5, then there is uncertainty in the algorithm's ability to form a hypothesis, and the detection is also rejected (Step 3.5). Both rejection instances further exemplify modulation. The algorithm only continues to Step 4 and outputs the final hypothesis for a detected object if all uncertainty criteria have been passed.

3.2 Novelty

The UML algorithm also includes the option of examining an input's novelty to determine if it constitutes a not-yet-seen category. Algorithm Step 6.3.2 shows that during learning, if an input does not match any existing nodes, the algorithm will create a new category or class for that input. The new class is immediately available for further refinement and classification. In this way, the algorithm can add entirely new classes in an unsupervised manner. This function, which is inherent in ARTMAP, represents a form of modulation, but more importantly, it provides the algorithm with a form of one-shot learning.

3.2.1 Embodied Learning Agent

During this program, we developed the L2M architecture and its underlying algorithms using the application of a **flying drone performing asset recognition tasks**. This application featured an **embodied learning agent** maneuvering within an open simulated environment seeking out and identifying objects related to discrete asset categories (see Section 3.6.2 for asset details). The challenge in this application lay primarily in the agent's learning of how to recognize novel objects and views, and as additional asset categories became relevant, incorporating knowledge necessary to recognize such assets without reducing performance on prior ones. Such an application required that the agent **learn during live operation to improve its performance while on-the-job**, simultaneously evaluating and learning upon streaming information in a continuous manner.

We promoted learning of asset appearances in the drone agent through **uncertainty-driven self-supervised learning**. Under this paradigm the agent uses generated uncertainties to determine when and how to learn on a given object. Figure 17 illustrates one version of self-supervised learning utilized by the agent. In this scenario, a drone agent pretrained on vehicle appearances from the ground only encounters a sedan from the air, where it generates high uncertainty. Recognizing a need to learn, the drone flies to the ground where it has low uncertainty and acquires the learning label ("sedan") for the object. This label is then retained as the agent flies back up to its operational altitude, and uncertainty in the views encountered is used to trigger additional learning using the UML algorithm within C4. At altitudes where the agent was originally trained, UML algorithm operates without modulation or learning, as low uncertainty in its results reflect an appropriate level of understanding. At altitudes near to those where the agent was originally trained, views generate moderate uncertainty, and the UML algorithm both returns its hypotheses on given views and engages neuromodulatory mechanisms to learn by changing existing weights to tune its understanding of the object. At high altitudes where the agent has no prior training, views generate high uncertainty, and the UML algorithm rejects its hypotheses on given views ("van") and engages novelty-related neuromodulatory mechanisms to learn by generating additional weights and associating them with the label acquired from low-uncertainty samples ("sedan"). This provides **one-shot learning** capabilities to the agent. In this way, the agent learns to recognize assets from new views, improving its performance on an asset recognition task during live operation.

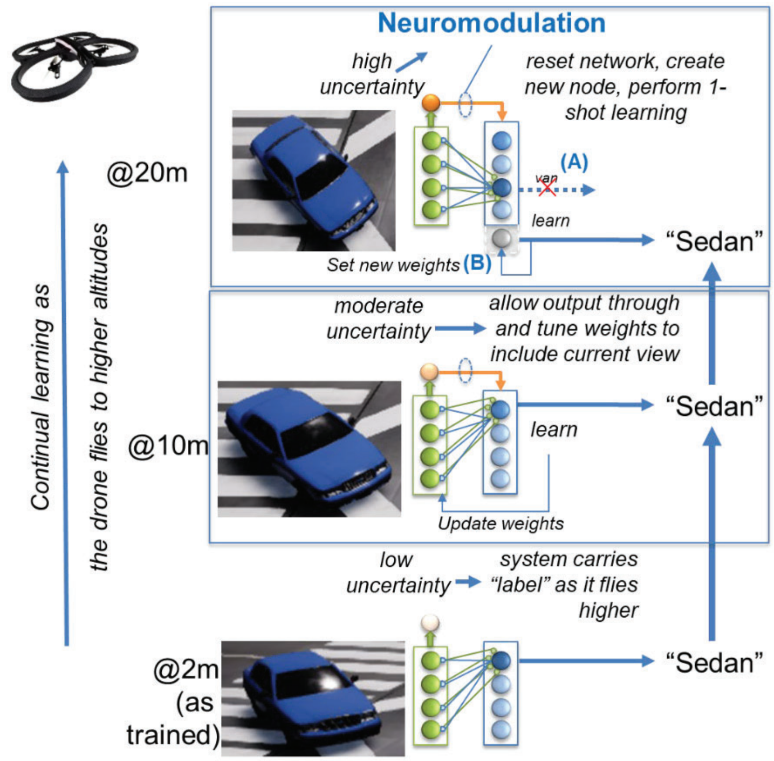


Figure 17. Self-Supervised Learning of a New Object

Following development, embodied learning agents using the L2M architecture were subsequently put through a series of demonstrations (Phase 1) and learning scenarios (Phase 2) to demonstrate L2M capabilities. Demonstrations consisted of learning scenarios showcasing the ability of learning agents to improve asset recognition task performance over time and acquire new skills. Learning scenarios were defined by the L2M Program and consisted of the agent learning multiple asset recognition tasks in sequence with evaluation periods following each learning period. Program metrics generated during these scenarios were used to demonstrate overall success during the program.

3.3 Metrics

3.3.1 Overview

Metrics consisted of two types: application-level metrics and lifelong learning metrics. Application-level metrics are defined as those generated within the L2M application itself; such metrics carry application-specific units and characterize the ability of the learning agent to perform a given task. Lifelong learning metrics are those that characterize the overall learning abilities of a learning system; such metrics are unitless, are generated using application metrics for a given system, and are agnostic to a given application or system configuration. For the latter reason, lifelong learning metrics were used as the L2M program metrics and will be referred to as such in subsequent sections.

3.3.2 Application-Level Metrics

Application-level metrics calculated in this project were generated by performing a series of asset recognition tasks during learning scenarios. Generally, a drone agent with or without some level of pretraining was flown through an open simulated environment, simultaneously performing object recognition tasks and learning at key locations. Learning was disabled at certain locations to allow evaluation without concurrent learning.

Firstly, **object recognition accuracy** and later **precision** were monitored during operation, quantifying the ability of the system to return an appropriate classification for given objects. Accuracy was initially measured to quantify the benefit of UML's ability to report "unknown" instead of committing to a poor decision. The metric thus gives partial credit for potential True Negatives as follows:

$$P(ID) = \frac{\#\ Correct\ ID + {}^{\#\ Uknown}/_{2}}{Total\ Objects}$$

Equation 1. Object Recognition Accuracy

UML is unique among most machine learning algorithms in being able to confidently declare that it does not have enough information to produce an output. This should be highlighted as an important contribution to the field. However, as the program progressed, and more emphasis was placed on comparing performance across performers, we opted to adopt the standard metric for precision as follows:

$$Prec = \frac{TP}{Total\ Objects}$$

Equation 2. Precision

These metrics were adjusted to incorporate the identification of context, active task, or task-relevancy in those configurations of the agent able to generate them; in such cases, partial credit was awarded for accuracy metrics for correct recognitions of either object or context labels. Additionally, accuracy metrics awarded partial credit when samples were labeled as "unknown", indicating successful identification of when the agent could not confidently classify a sample based on prior knowledge. Precision metrics were calculated using only those samples given a confident identification, with the agent treating "unknown" samples as too high uncertainty to use for task execution.

Additionally, **levels of uncertainty** were measured as application-level metrics. These metrics included uncertainty of similarity between a sample and the agent's knowledge base ("fit"), uncertainty in an object's classification over time ("persistence"), uncertainty in a classification given conflicting alternatives ("contested"), and uncertainty in knowledge base organization (“fragmentation”), among several others (see [19] for additional details). These uncertainties were used to both monitor and drive the decision-making ability of the learning agent:

fit: $$U_f = dist_{L1}(w_{J,}A) \quad (1)$$

persistence: $$U_p = P(J|N\ samples) \quad (2)$$

contested: $$U_c = \frac{(w_J \cap W)}{(w_J \cup W)} \quad (3)$$

fragmentation: $$U_{fr} = P(J|W^T) \quad (4)$$

where,

distL1 is the L1 distance, and

W^T are all the candidate categories. Thus, fragmentation measures the percentage of those candidates that match the most likely category.

Equation 3. Uncertainty Measures

Lastly, **memory requirements** (measured as total number of bytes allocated in RAM) for the learned knowledge base were monitored during operation as an application-level metric. Lifelong learning methods such as ours that allow the creation of additional weights will practically be limited by the platforms on which they operate, making memory optimization an important part of long-term lifelong learning. Metrics calculated here were used to promote such optimization operations.

3.3.3 Lifelong Learning / Program Metrics

The program metrics calculated for this project served to evaluate the lifelong learning agent in a system and application-agnostic way, allowing for comparison against other agents. Methods and logic for each metric are described briefly below, with a more in-depth description and calculation being available in [31].

- **Performance Maintenance (PM)** – the ability of a lifelong learning system to retain performance on a set of prior tasks following learning on a different task. A value ≥ 0.0 reflects a system that does not catastrophically forget following new learning.

  Computed by:
  - Select an application-specific metric to monitor for the given environment (accuracy or precision)
  - Set up a learning scenario with a sequence of Learning Blocks alternating with Evaluation Blocks. Each Evaluation Block exercises all the previously learned tasks.
  - For a given task and Evaluation Block:

        - Calculate the Maintenance Value, defined as the difference between each the Task's most recent Evaluation Performance and second-most-recent Evaluation Performance
    - Performance Maintenance for a lifetime = mean Maintenance Value across the lifetime
- **Forward Transfer (FT)** – the ability of a lifelong learning system to utilize information learned on one task to improve the learning on a set of different, subsequent tasks. A value ≥ 1.0 reflects a system that learns information in such a way that it can be utilized by subsequent tasks to improve their own learning.

    Computed by:
    - Select an application-specific metric to monitor for the given environment
    - Set up a learning scenario beginning with initial Evaluation Blocks for all tasks, followed by a sequence of Learning Blocks (for different tasks) alternating with Evaluation Blocks.
    - Assuming a block sequence like: *Eval Block 1, Learning Block 1 (Task-1), Eval Block 2*, then Task-2's Forward Transfer (from Task-1) is computed as the contrast of the Evaluation Performances of Task-2 in Eval Block 2 to Eval Block 1.
    - A task's Forward Transfer is this FT calculation, the first time it appears in the learning scenario.
    - The Forward Transfer for a lifetime is the mean of each task pair's Forward Transfers.
- **Backward Transfer (BT)** – the ability of a lifelong learning system to utilize information learned on a new task to improve the performance on a set of different, previous tasks. A value ≥ 1.0 reflects a system that improves performance of prior tasks using information acquired during a new task, which suggests shaping the underlying knowledge base in a way that is helpful to multiple tasks.

    Computed by:
    - Select an application-specific metric to monitor for the given environment
    - Set up a learning scenario with a sequence of Learning Blocks. Between each Learning Block, there are Evaluation Blocks for each of the other tasks.
    - For each task:
        - Compute the Backward Transfer Contrast, defined as the contrast of the average performance within the most recent Evaluation Block to the second-most recent Evaluation Block
        - Backward Transfer for task T = the average of the Backward Transfer Contrasts
    - The Backward Transfer for a scenario is the mean of each task pair's first calculated Backward Transfer value.

- **Relative Performancee (RP)** – the ability of a lifelong learning system to utilize the knowledge shaped from multiple tasks to improve its performance on a given task. This metric compares the lifelong learner against a learner trained only on the given task, and a value ≥ 1.0 reflects a system that can leverage its cumulative knowledge to improve its performance overall.

  Computed by:
  - Select an application-specific metric to monitor for the given environment that has also been logged for a Single Task Expert (STE; e.g. a non-lifelong learner)
  - Set up a learning scenario with some sequence of Learning Blocks for some number of tasks
  - For a given task T:
    - Consider only the Learning Blocks for Task T, in order of appearance
    - Intuitively, compare the "area under the curve" for the lifelong learner experiencing Task T with the area under the curve for the equivalent Single Task Expert.
    - Formally, Compute the Single Task Expert Ratio, defined as the ratio of the sum of the application-specific metric over all of the Learning Experiences in the lifetime to the sum of the same application-specific metric over the same amount of learning experiences for the Single Task Expert
    - Relative Performance for Task T = the Single Task Expert Ratio
  - The RP for a lifetime is the mean of each task's RP score.
- **Sample Efficiency** – the ability of a lifelong learning system to utilize the knowledge shaped from multiple tasks to achieve competence on new tasks more rapidly. This metric also compares the lifelong learner against a learner trained only on the new task, and it measures the relative number of learning experiences required to achieve the same level of performance. A value ≥ 1.0 reflects a system that can leverage its cumulative knowledge to acquire new tasks over time.

  Computed by:
  - Select an application-specific metric to monitor for the given environment that has also been logged for a Single Task Expert
  - Set up a learning scenario with some sequence of Learning Blocks for some number of tasks
  - Intuitively, compare the saturation value of the Single Task Expert with that of the lifelong learner.
  - For each task T:
    - Consider only the Learning Blocks for Task T, in order of appearance
    - Compute Saturation Value (the max of the rolling average of the application-specific metric) and the Experience to Saturation (the number of Learning

  - Experiences it takes to achieve the Saturation Value) for both the L2 agent and the Single Task Expert system
  - Compute the ratio of the Saturation Values of the L2 agent and the STE
  - Compute the ratio of the Experience to Saturation (ETS) for the STE and the L2 Agent
  - Sample Efficiency = Saturation Value Ratio * Experience to Saturation Ratio
- **Performance Recovery (PR)** – the ability of a lifelong learning system to progressively reduce the time required to return to prior performance on a given task following learning on other tasks. A value ≥ 0.0 reflects a system that can rapidly adapt to perturbations in learning. (This metric was removed from program metrics late in Phase 2 due to erratic results across multiple performers.)

  Computation:
  - For each Task:
    - Select an application-specific metric to monitor for the given environment (e.g. precision)
    - Set up a learning scenario with a sequence of LX blocks. Each LX block introduces a parametric change to an already-learned task.
    - From the second Learning Block onwards, calculate the Recovery Time relative to the most recent Terminal Learning Performance. The "Recovery Time" is the number of LXs for performance to return to the previous Terminal Learning Performance.
    - Task-Specific Performance Recovery = negative slope of the line of (Learning Block index, recovery time) values.
  - Report Lifetime PR as the mean of all Task-Specific PRs

Our system calculated program metrics using the accuracy, then later precision, metrics described above. While our system is capable of concurrently evaluating and learning upon streaming information, following the learning scenarios outlined in [31] and described in Section 3.3.3, the first three metrics were calculated using evaluation periods without learning engaged; a similar format was used with sequestered validation data for the last two metrics later in the program. Metrics requiring single task learning algorithms measured the performance of learning agents using the L2M architecture with equivalent settings as a lifelong learning algorithm but only trained using samples from singular tasks.

3.4 Testing and Evaluation

We evaluated our L2M architecture and associated learning agents through algorithm demonstrations and learning scenarios with associated program metrics, as well as through separate experiments exploring other applications of the system.

3.4.1 Phase 1a Experiments and Demonstration

During Phase 1a, we began by executing a consecutive series of three proof-of-concept (POC) learning experiments examining fundamentals of uncertainty-based algorithms for lifelong learning. The results of these and other related experiments informed the design of a demonstration at the end of Phase 1a, which verified the applicability of our lifelong learning algorithm, dubbed UML [19], within an embodied learning agent performing asset recognition tasks.

POC 1 examined the ability of uncertainty-based algorithms to recover task performance under novel contexts. Under this experiment, a learning agent was pre-trained using randomized views of sedans, vans, and city clutter from the ground only, which imitated an agent restricted to ground-based movement. Then, this agent was exposed to randomized views of sedans from the air, representing a significant change in the sedans' appearance and leading to poor recognition performance. The agent was then allowed to learn on the new views in periods of high uncertainty, which **recovered performance on the task for the novel view** without affecting its performance on ground-based views.

POC 2 pushed this capability to recover performance on a separate set of resources in a new context without catastrophically forgetting prior learning. Continuing from the same model as POC 1, this experiment exposed the learning agent to randomized views of vans from the air, which it initially performed poorly on. After subsequently learning on views of vans from the air according to sample uncertainties, the agent **recovered performance on that asset as well without impacting prior performance** on ground-based imagery nor air-based sedan imagery.

POC 3 investigated the algorithm's ability to identify and learn upon entirely novel information. In this experiment, the model from the end of POC 2 was exposed to random samples of a new class of never-before-seen objects, fire trucks, which it could not correctly identify based on prior knowledge. Using different uncertainties focused on novelty, the agent then identified select views of the object to learn upon, adding the new object class to its knowledge base in a self-supervised manner. Finally, labels were applied manually to the novel learned information, ultimately **learning to recognize a new object class** in a **one-shot** or **few-shot** manner.

Following POC 1-3 and two smaller experiments confirming the suitability of self-supervised learning with continuous imagery, Phase 1a efforts culminated in a demonstration of an **embodied learning agent** operating within an open environment. In this demo, a drone agent pre-trained with ground-based imagery of vans, sedans, and city clutter encounters sedans at an intersection from the air and executes a series of maneuvers to learn to recognize them from its new operating altitude using continuous imagery. Following this, the agent flies to another intersection containing both sedans and vans, where it repeats similar maneuvers to acquire recognition skills with vans from the air. Finally, the agent encounters fire trucks at a third intersection, which it learns as new classes following self-supervision before then adding a unifying label, thus acquiring recognition skills with an entirely new class.

3.4.2 Phase 1b Experiments and Demonstration

During Phase 1b, we examined the uses of additional uncertainties and contextual signals to enable **discovery** and **on-the-job learning**. Following two smaller experiments examining focused, uncertainty-driven, self-supervised learning of asset objects of interest, we ran a fourth POC experiment. POC 4 utilized a hierarchical version of UML including drone altitude as a contextual signal, which provided a top-down attentional mechanism to promote differential treatment of assets based on context. In this experiment, a learning agent was trained using randomized imagery to perform 1 of 3 actions (COUNT, AVOID, IGNORE) based on a combination of an object's class and the altitude from which it was viewed, which required appropriate recognition of both class and an active task directly based on context.

The Phase 1b demonstration utilized an embodied learning agent with the ability to execute different actions in response to recognized objects based on contextual signals. This demo continues from that of Phase 1a, using a drone-based agent trained on ground and aerial views vehicles locating vehicles it needs to count. The agent in this demo additionally utilizes a distance sensor to discover objects within its flight path, which prompt it to learn to avoid such objects at operating altitudes and ignore them at higher altitudes. Finally, the agent uses its new knowledge to autonomously maneuver in the environment while continuing its counting task.

3.4.3 Phase 2 Learning Scenarios

During Phase 2, we ran a series of Learning Scenarios defined during the program to evaluate the capabilities of UML and the L2M Architecture based around it. Learning scenarios consisted of having a learning agent attempt to learn on multiple tasks and variants in sequence, which tested its ability to acquire new skills without adversely affecting prior ones. Tasks were defined as asset recognition/management tasks using discrete object categories, as described in Section 3.1; task variants were defined as different versions of a given task, such as from different altitudes or using different times-of-day. Tasks, variants, and object categories are defined in Section 3.6.7.

Learning scenarios were run at discrete months into the Phase 2 period of performance (denoted M#) following Table 4. Scenarios were progressive in difficulty and were routinely repeated to evaluate development progress. A given scenario included a series of learning blocks where the agent learned to perform a single task variant during each block; these were followed by an evaluation block where the agent was tested on each task it would encounter during the scenario. Each scenario type used a different number of tasks and variants to increase the complexity and difficulty of evaluations over time. The organization of tasks and variants encountered was determined by the scenario type as described in Table 5.

Table 4. Learning Scenarios by Evaluation Period

Evaluation Period	Scenario Types
Month 9	Permuted + Alternating (simplified)
Month 12	Permuted + Alternating
Month 15	Condensed + Distributed
Month 18	Condensed + Distributed

Program metrics were accumulated across several runs of each learning scenario for the learning agent. Each run included randomization in the order of tasks and task variants. Finalized metrics were calculated by the Johns Hopkins University Applied Physics Lab (APL), the independent test and evaluation partner for the program, using logs generated during each run.

Table 5. Scenario Types

Scenario Type	Tasks	Variants	Ordering
Permuted (simplified)	3	1 variant, equivalent across tasks	1 learning block for each task, order randomized
Alternating (simplified)	2	1 variant, equivalent across tasks	3 learning blocks for each task, order alternating
Permuted	4	1 variant for each task	1 learning block for each task variant, order randomized
Alternating	2	1 variant for each task	3 learning blocks for each task variant, order alternating
Condensed	3	2 variants for each task	1 learning block for each task variant, order randomized
Dispersed	3	2 variants for each task	3 learning blocks for each task variant with total number of samples equivalent to condensed, order randomized

3.4.4 Out-of-Nominal Recovery Experiments

Separately from the learning in the embodied agent, we carried out two additional experiments during the program focusing on adaptations under noisy conditions. These experiments explored how uncertainty caused by deviations from expected operating conditions could be used to recover performance in a trained system. The first experiment explored uncertainty as a means of retraining a deep neural network to handle noisy inputs on a character recognition task. The second experiment expanded the applications of the UML algorithm into reinforcement learning (RL) by detecting errant model states caused by corrupted frames in an ATARI game and adjusting the behavior of the RL agent to minimize performance loss.

3.5 Assumptions

To develop a learning agent with self-supervised learning capabilities, our application included an object tracker enabled through the testing environment (see Section 3.6.1). This tracker provided unique identifiers for each object within the field of view, which were used to monitor detections of a given object over time, to provide ground truth labels as needed, and to filter objects not directly related to any evaluated tasks. Transitions with this architecture would benefit from a separate object tracker (implemented within C1) to enable similar functionality in a functional agent.

Additionally, for our chosen application it was assumed that task descriptions, including relevant assets types and contexts, were defined by a human operator and provided to the agent during initial learning. The agent is capable of improving task performance and learning new classes in a self-supervised manner, but it relies on such definitions to identify the boundaries of a given task within which it can then improve autonomously.

3.6 Procedures

3.6.1 Data and Feature Extraction

Data for each experiment, demonstration, and learning scenario was collected using AirSim [32], a software package capable of generating simulated imagery for autonomous vehicle and drone platforms. Data consisted of sequences of RGB imagery captured during varied movements and flight patterns within a custom urban environment with accompanying segmentation information, operating altitudes, and in specific cases distance information to objects within view. Each task-relevant object was assigned a unique segmentation id, which was subsequently used to track objects over multiple views and assign ground truth values as needed.

Following data collection, RGB imagery was run through a feature extractor (YOLO v3 [30] or YOLO v4 [33]) to produce features and bounding boxes of relevant objects and imagery. The feature extractor was pre-trained on the COCO dataset [34], and features were collected from a convolutional layer towards the end of the network. Bounding boxes provided an attentional mechanism for the system and were used to localize objects within view. Vectors of features representing individual objects were formed by spatially averaging all pixels within each bounding box.

3.6.2 Asset Management Tasks

Learning tasks in this program consisted of asset recognition tasks wherein a learning agent was required to recognize target assets within an environment and report the appropriate class and later relevance to the active task. Asset recognition tasks were designed to mirror potential applications wherein a given agency, such as the department of transportation, would task the learner with locating and identifying resources relevant to that department. Lifelong learning would be necessary for such an agent both during operation to improve its performance over time and in the case where another agency acquired the agent and assigned it to their own resources.

Tasks utilized assets from the following four major categories: personal vehicles, emergency management agency vehicles (EMA), department of transportation resources (DOT), and urban clutter. Table 6 gives assets included in each category. Asset categories were used to define the general bounds for training and testing. Phase 1 utilized resources from all four categories in different combinations as described in respective experiments; Phase 2 utilized resources from the EMA and DOT assets, specifically. Additionally, certain evaluations split a given asset category into multiple tasks based on agent altitude, requiring analysis of context as well as asset appearance.

Table 6. Asset Categories

Asset Category	Assets
Personal Vehicles	Sedans (including sedans, taxis), vans (including mail, delivery, and police vans)
Urban Clutter	Benches, crosswalk signs, fire hydrants, trash cans, and patio umbrellas
Emergency Management Agency (EMA)	Fire trucks, ambulances, police cars
Department of Transportation (DOT)	Traffic lights, stop signs

3.6.3 Phase 1a Proof of Concept Experiments

(Detailed descriptions of POC 1-3 is available in the related publication [19].)

The first proof of concept (POC) experiment in Phase 1a utilized principles of uncertainty to drive continual learning to recover skill performance under a new context without affecting performance on the prior context. First, imagery of personal vehicles and urban clutter was collected at ground level (0-2 m) within a custom simulated environment. Assets within the imagery were organized into the following three sets:

- Set A: sedans, including all sedan objects from the personal vehicle asset category as well as police cars
- Set B: vans, including all van objects from the personal vehicle asset category
- Set O: other, including all urban clutter category objects (except for umbrellas) as well as traffic lights and stop signs

Example imagery from Sets A and B as well as the simulated environment are shown in Figure 18: (a) shows an overhead view of the custom AirSim environment used during Phase 1a development and testing; (b,d) and (c,e) show paired ground and aerial images used in data collection. The graphic is taken from a related publication [19].

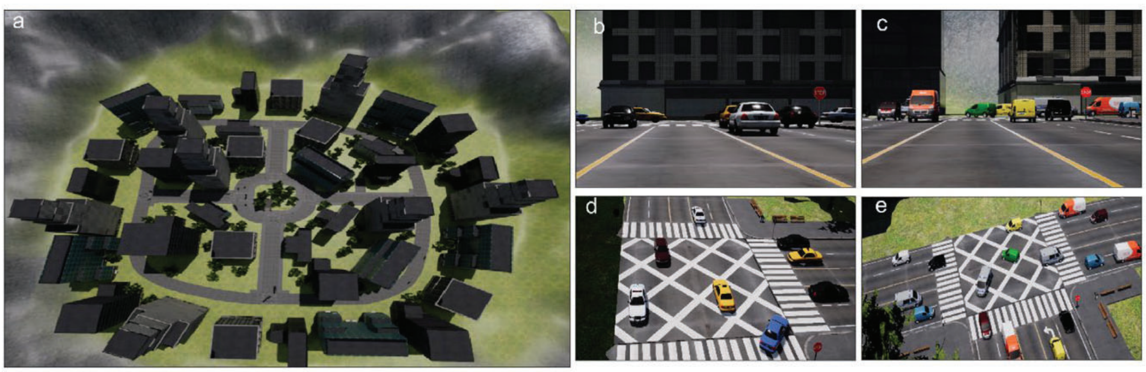


Figure 18. AirSim Environment and Imagery

Following object localization and feature extraction (YOLO v3), this ground-based imagery was then randomized and used to train a UML network. All samples within a given set were assigned the same learning label, producing a network with three total classes. This formed the baseline ground-based network.

Next, imagery of the assets from Set A was captured from the air (5-35 m). As before, object instances and features were identified using the same feature extractor. Object instances were randomized and split to form training and testing sets, respectively.

Finally, the ground-based network was trained sequentially on samples from the aerial Set A training set to learn to recognize them from a substantially different context and view. Following principles of uncertainty, the network only learned on samples it identified with high uncertainty, such as having a poor fit with existing network expectations or following an incorrect classification. The network was configured such that one of the three classes was returned for each sample, as opposed to returning an “unknown” designation. Accuracy on the ground-based Set A and B imagery and the aerial Set A testing set were calculated at regular intervals during training to monitor learning performance on both tasks over time.

The second POC experiment in Phase 1a built upon POC 1 to show recovery of performance on additional resources without catastrophically forgetting prior ones. This experiment built on the network formed in POC 1, which could classify objects from Set A and B from the ground and Set A from the air. Similarly to POC 1, imagery of assets from Set B were captured from the air (5-35m), and object instances and features were collected using the feature extractor. Subsequently, training and testing sets were formed from the imagery through sample randomization. The training set was then used to progressively train the network from POC 1 on Set B assets from the air using principles of uncertainty. Accuracy on the ground-based Set A and B imagery, aerial Set A testing set, and aerial Set B testing set were calculated at regular intervals during training to monitor learning performance on each task over time.

The third POC experiment in Phase 1a continued from POC 2 to demonstrate few-shot learning on entirely new information using the additional image set using new objects defined as: Set C: fire trucks. Imagery for Set C was collected from the air (5-35 m), and extracted instances and features were randomized and split to form training and testing sets. The trained model from POC 2 was subsequently trained on the Set C training imagery without any learning labels, allowing the network to form new classes as needed to accommodate the new information. Afterwards, any new classes formed were assigned the same class label, and this updated network was used to calculate accuracy on the Set C test set and all prior test sets to show the acquisition of a new class without impacting performance on prior classes.

3.6.4 Phase 1a Demo

(A detailed description of this demonstration is available in the related publication [19].)

The first program demonstration showcased an embodied learning agent capable of using self-supervised learning to recover skills under a new context and acquire new skills while learning on-the-job within an open environment. This demonstration built on principles examined during POC 1-3 and illustrated their use on a potential application for the algorithms developed during this program.

The mode of self-supervised learning used during this demo was developed using two additional, smaller experiments. In the first, it was found that training on multiple altitudes boosted performance for a given altitude, providing transfer during learning across altitudes [19]. In the second experiment, it was observed that using continuous, related imagery reduced network memory requirements without affecting learning performance. These smaller experiments provided confidence in the mode of learning employed below.

During the demo, a drone agent pre-loaded with the baseline ground-based UML network from POC 1 was flown through three intersections. At each intersection, objects and features were collected using the feature extractor, and features for vehicular objects were passed to the UML network for classification and potentially learning. In all cases, objects were classified prior to learning on any given sample, and detection/classification labels were assigned to each object following evaluation of uncertainties in their temporal characteristics.

At intersection 1, the agent first viewed sedans from the air at 30 m, where it has not been trained. Following principles of self-supervised learning as illustrated in Figure 17, the agent flew to a lower altitude, where it could better identify objects with lower uncertainty and acquire ground truth labels for them. The agent subsequently flew back upwards to its operating altitude then in a semicircle around the sedans to learn to recognize them from this new view and context, using object tracking and principles of uncertainty to learn on each object using its acquired labels as needed.

At intersection 2, the agent encountered both sedans and vans from the air. Similarly to intersection 1, the agent flew to a lower altitude to acquire self-supervised learning labels for each object. Then, it flew back up to its operating altitude of 30m, learning to identify vans from the air. The agent was then flown back to intersection 1 to demonstrate its performance on prior information.

Finally, at intersection 3, the agent encountered fire trucks, which represented a class the agent had never before encountered. Here, the agent's uncertainty thresholds were modulated to promote identification of novel information and subsequent self-supervised learning of novel classes. After being allowed to develop new classes based on individual objects in view, new classes were assigned the same unique label, achieving one/few-shot learning for the new object type.

3.6.5 Phase 1b Proof of Concept Experiment

The Phase 1b proof of concept experiment, POC 4, focused on the application of context signals to modulate agent behavior on recognition of a task-relevant asset. A task was set up where the appropriate reaction to a given asset was based on both the identity of the asset and the context in which it was recognized, thus requiring multiple layers of processing and providing modulation to the actions taken by a given embodied agent.

For this experiment, data was collected that mimicked operation of a drone learning agent flying down a street attempting to count vehicles while changing altitudes to avoid traffic lights. Views of traffic lights from high altitudes (5-30 m) were to trigger AVOID actions to avoid collision, while views from low altitudes (0-2 m) were to trigger IGNORE actions. For vehicles, views from high altitude were to trigger COUNT actions to perform the given task, while views from low altitude were to trigger AVOID actions to avoid collision. Vehicles included objects from Sets A, B, and C from POC 1-3, and samples were presented without any distinct order. Figure 19 shows examples of the training and testing data for POC 4.

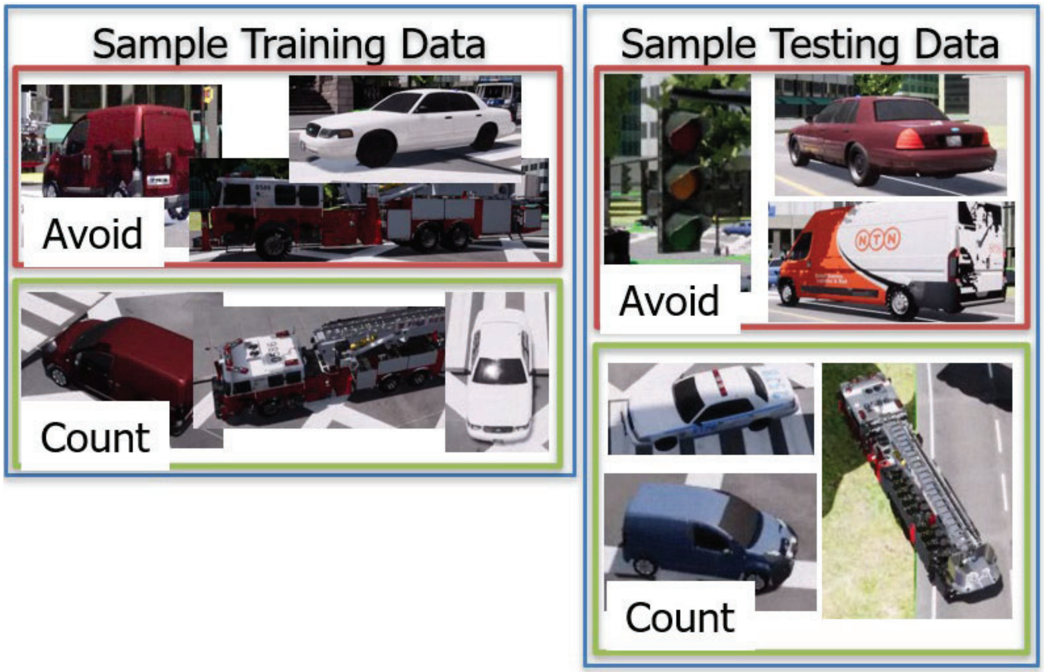


Figure 19. Sample Training Data for Phase 1b Proof of Concept

To learn this task, a hierarchical UML network with two layers representing object class and agent action was constructed. Inputs to the first layer consisted of object features, and inputs to the second layer included the object features as filtered by the first layer. Additionally, drone altitude was included as a feature for the second layer to provide context to learning. During training, the network identified when and what to learn on provided samples following evaluation of uncertainties, and labels were provided as needed.

Training and testing of the network took place in three stages. Firstly, the lower layer of the hierarchy was first trained using all available samples to be able to accurately classify objects from multiple altitudes. Secondly, the upper layer was trained on a sparse subset of the training data, providing it with the means of returning any of the three actions (AVOID, IGNORE, and COUNT) during inference. Finally, the upper layer was trained sequentially on the remainder of the training data. Following the second and third stages, network performance was measured against the whole dataset as a "test" set, showing performance before and after sequential training. During the third stage, performance of the upper layer was monitored before learning on every sample, allowing evaluation of the network as it learned. Additionally, during the third stage, the layer was configured such that it could also return an "unknown" designation if the sample was too high uncertainty, representing an identification of samples outside the distribution of previous samples.

3.6.6 Phase 1b Demo

The demonstration for Phase 1b incorporated principles from POC 4 into the trained agent from Phase 1a, adding self-supervised flight capabilities enabled through recognition of appropriate actions given recognized assets and current context. Additionally, this demo included elements of discovery, allowing the agent to use a second type of contextual signal to provide a means of discovering and learning on novel object classes requiring new flight patterns.

Here, the embodied learning agent as described for the Phase 1a demo was upgraded in two ways. Firstly, its learning network was updated to use hierarchical UML as described for POC 4; this enabled it to learn appropriate actions for recognized assets using altitude as a contextual signal. Additionally, the agent was given a proximity alarm utilizing AirSim depth imagery; this alarm allowed the agent to discover new objects it previously had not been trained to recognize, and it would initiate new actions when those objects were subsequently detected using more standard imagery.

The demo started by having the agent run through the intersections of the Phase 1a demo with its new configuration, learning to perform a COUNT action for all vehicular assets from Sets A, B, and C from the air. Subsequently, the agent's proximity alarm was triggered by a patio umbrella directly in its path, which it did not recognize through RGB imagery due to its lack of similarity with other learned objects. Once the alarm was triggered, the learning agent initiated a pre-programmed flight pattern to learn to recognize the object from multiple altitudes, to AVOID the object at operational altitudes, and to IGNORE the object at great altitudes. The agent used self-supervised methods to learn the new object, assigning it an internally consistent label unrelated to prior resources and using uncertainty in object views to determine when learning was necessary. Use of this method of learning was supported through two small experiments, which showed that uncertainty could be used to identify appropriate views for learning of objects, and that uncertainty in views could drop over time, providing a means of self-guided learning.

After learning to recognize and apply appropriate flight patterns when viewing an umbrella, the drone then encountered a traffic light in its path. This again triggered the proximity alarm, initiating self-supervised learning on the object in the same fashion and creating another internally consistent label for it. The agent then encountered both traffic lights and umbrellas again in turn, where it would recognize them from its RGB imagery and autonomously maneuver to AVOID them. Finally, the agent encountered vehicular assets again to show a lack of catastrophic forgetting after discovering the two new classes and learning on them in a few-shot manner.

3.6.7 Phase 2 Learning Scenarios

In Phase 2, the learning system code was reconfigured to operate within the L2M architecture described in Section 3.1, and learning scenarios were utilized to evaluate the whole system's learning abilities. Scenarios at each evaluation period increased in difficulty over time according to Table 4 and using numbers of tasks and task variants outlined in Table 5.

Our learning scenarios utilized a flying drone agent learning to perform a series of asset recognition/management tasks. In each portion or "block" of the scenario, the agent would encounter assets for single task as defined in Table 7 and potentially learn on streaming samples in a self-supervised manner. To expedite the execution of a given scenario, the agent was assumed to have already acquired a self-supervised label for each object by flying to a low-uncertainty view prior to the block. As in the Phase 1 demos, objects were classified prior to learning on any given sample, and labels were reported for each detection following evaluation of uncertainties in their temporal characteristics. Scenarios increased in difficulty in later evaluation periods, encountering task-specific assets under multiple times of day (see Table 7).

Table 7. Scenario Tasks and Task Variants

Evaluation Periods	Scenario Types	Tasks	Variants
Month 9	Permuted (simplified) Alternating (simplified)	EMA Low (10 m)	Noon
		EMA Med (20 m)	Noon
		EMA High (30 m)	Noon
Month 12	Permuted Alternating	EMA Low (10 m)	Dusk
		EMA High (30 m)	Morning
		DOT Low (2 m)	Noon
		DOT High (5 m)	Afternoon
Month 15 Month 18	Condensed Dispersed	EMA Low (10 m)	Morning Dusk
		EMA High (30 m)	Morning Dusk
		DOT (2 m)	Morning Dusk

Learning scenarios consisted of repeating units of evaluation blocks and learning blocks. During a learning block, the agent would be allowed to learn as determined necessary through uncertainty analyses on a single task variant as outlined in Table 7; task performance was calculated using labels inferred for each task-relevant detected object and logged using APL's l2logger software. During an evaluation block, the agent would perform on imagery corresponding to each of the run's task variants in sequence, obtaining a task score for each. The agent was allowed to adapt to changing circumstances in evaluation blocks, but not to make any changes to long-term weights. Each learning block was preceded and followed by an evaluation block to monitor how continual learning on a task variant affected performance on all task variants, and the total number of learning blocks was determined by the scenario setup and number of task variants used. Additionally, for condensed and dispersed scenarios an offline learning block was often inserted at the end of each learning block. During these blocks, the agent used model self-evaluations or samples from prior learning blocks to optimize its internal representations, functionally executing a "sleep" cycle.

Using logs generated by individual runs of each scenario, program metrics were generated both by sending APL training logs and by running APL's l2metrics software internally. Additionally, training logs were generated for a single-task expert (STE) for each task variant used in a scenario; here, an STE was defined as a learning agent described above which encounters only samples from a single task variant. STE logs were used to enable calculation of two program metrics, as noted in Section 3.3.3.

To continue development on lifelong learning algorithms, updates were made to the learning system continuously between Phase 2 evaluation periods. Teledyne Scientific lead development efforts and both designed and implemented in the L2M architecture, C1, C2, C4, C5, and the interface layer (see Figure 14 for component definitions). Missouri S&T developed algorithms in C3, and UCI and UCSD developed algorithms in C6. Updates incorporated into the learning system for each evaluation period are noted in Table 8; updates included changes in model pretraining, adding components and connections, updating components, and updating application metrics as needed.

Table 8. System Updates for Evaluation Periods

Evaluation Period	System Updates
Month 9	**L2M Architecture** – Implemented. C1 – Enabled with with Yolo V3. C2 – Enabled with communications with other components. C4 – Enabled with UML. Pretrained on ground imagery.
Month 12	Interface – Implemented. C4 – Hierarchy added with task relevance. Pretraining removed. C5 – Initial version implemented but not enabled.
Month 15	Metrics updated to use precision. C3 – Enabled, but context output not fed to other components. C4 – Learning rules and uncertainty calculations updated. C5 – Enabled with optimization rules updated.
Month 18	C1 – Updated to Yolo V4. C3 – Context fed to to C4 hierarchy. C4 – Hierarchy updated to apply top-down modulation of recognition results. Learning rules and uncertainty calculations updated. C5 – Optimization rules updated.

3.6.8 Noisy Conditions Experiments

The noise filtration experiments examined how uncertainty analyses could impact recovery in trained systems encountering noisy inputs or circumstances. The first experiment examined how a deep neural network classifier could be retrained to recover performance on a character recognition task with noisy inputs. The second experiment applied UML alongside a reinforcement learning (RL) agent playing an ATARI game and modulated its actions during corrupted frames to recover performance.

For the character recognition task, a small convolutional network was initially trained to classify clean, well-defined handwritten characters from the MNIST dataset [35]. Following this, the network was evaluated on the noisy-MNIST dataset [36], which includes low-contrast versions of the characters with noise added. Classification accuracy was recorded to show how performance changed in the presence of such noise. Reflecting potential operations in a system capable of recognizing the change in operations state represented by the noise, the network was subsequently trained on test data from the noisy MNIST dataset, but with a substantially reduced learning rate, promoting minor tuning of parameters. Following this, performance on both the MNIST and noisy-MNIST test sets was tested to examine how this modulation to training affected both variations.

For the ATARI task, an RL agent was trained to play Pong using original frames that were visually ideal and uncorrupted by noise or perturbation. Concurrently, a variational auto-encoder was trained to predict the next frame of the game using the current frame and the chosen action from the RL agent; the portion of this network following encoding then represented a high-dimensional feature representation of a prediction for the next frame. A UML network was then trained in an unsupervised manner on activations of this encoded region using uncorrupted imagery and RL agent actions. After training on the ideal data, the UML network was set to inference mode to identify novel, high-uncertainty predicted states and produce a modulatory signal back to the RL agent.

Following training, the RL agent was tasked with playing Pong but with some percentage of frames randomly corrupted prior to input to the agent. Methods of corruption included adding a level of noise (10% or 30%) or perturbing the image vertically or horizontally, shifting portions of the image to new locations. Gameplay performance of the RL agent in the presence of such corruption was recorded. Subsequently, the UML network was connected to the RL agent such that it could modulate the action chosen by the agent for a given frame prior to its impact on the environment. The UML network was run in parallel with the RL agent, evaluating each predicted state during gameplay; when UML identified a high uncertainty state, one that did not fall within the known state-action distributions, it would change the action taken by the RL agent to NO-OP and preventing any action from being taken that frame. In this way, the UML network modulated the RL agent’s actions in the environment under high uncertainty conditions. Gameplay performance of the RL agent was again evaluated on corrupted games but with UML modulation engaged.

3.6.9 Hardware

All training and testing were performed using high-performance desktop units running Ubuntu 16.04 or 18.04 with NVIDIA GPUs and related software. Operations involving a feature extractor/C1 were run on a single GPU, while operations involving UML were run on multithreaded CPUs. Phase 1 proof of concept experiments and Phase 2 learning scenarios utilized stored imagery to improve reproducibility during software development, while the Phase 1a and 1b demonstrations operated in real-time with continuous data collection and streaming. Individual runs of Phase 2 learning scenarios were completed using Docker images run on a Slurm cluster to enable parallel execution. All software and associated documentation were packaged and delivered as part of our end-of-phase submissions.

4.0 RESULTS AND DISCUSSION

The experiments in Phase 1 were focused on refining the core UML algorithm to exhibit characteristics of lifelong learning. These capabilities would be demonstrated in a set of demonstrations at the midpoint (Phase 1a demo) and the end of the phase (Phase 1b demo). For detailed descriptions of each of the experiments conducted, please refer to Section 3.5.

4.1 Proof of Concept Experiments

To ensure that we had confidence in our algorithm improvements to support lifelong learning capabilities, we set up three proof-of-concept (POC) experiments that would sequentially build capability towards the Phase 1a demonstration. Each POC focused on a simple version of a component of the Phase 1a demo. POCs 1-2 both used two sets of vehicle targets, sets A and B. Each set has both ground and aerial viewpoint images that are used for training and testing. POC 3 included aerial images of a third vehicle target type, set C.

4.1.1 POC 1

This experiment focused on transferring knowledge from a known task to a new task. Set A and B vehicles were initially classified with high accuracy from the ground, and low accuracy from the air. The system was allowed to incrementally learn on aerial views of Set A. The results showed that it could learn the new task of aerial classification of Set A without forgetting the previously known task of classifying set A and B from ground level. Figure 20 shows a graph of the results of this experiment as the learning is occurring. When allowed to learn on more than 3 targets, the system recovers to within 3% of the ground level accuracy. These results in POC 1 show that the algorithm can acquire new viewpoints of known objects without catastrophic forgetting.

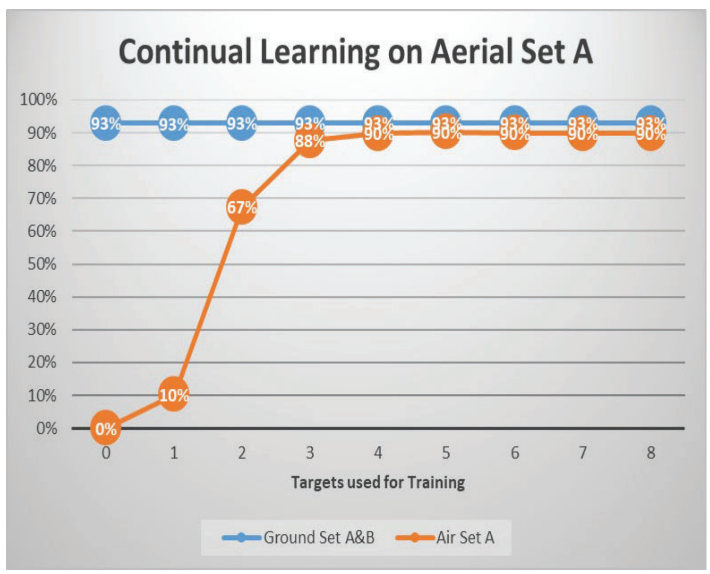


Figure 20. Proof of Concept 1 Results

4.1.2 POC 2

This experiment focused on extending the capability demonstrated in POC 1 to a second aerial set, displaying the capability to learn new viewpoints of multiple types of known targets. This POC was functionally a continuation of the behaviors in POC 1, and this is shown in the timeline graph in Figure 21, where the first half of the figure is same as POC 1, but the second half of the graph shows the acquisition of new viewpoints of Set B.

Figure 21 shows a repeat of the learning of Set A aerial views (orange line), and includes a new line (gray), which is the performance of Set B aerial views. The performance curve shows that the system can learn aerial viewpoints of the Set B objects without forgetting either the ground level views or the previously learned Set A aerial views. These results gave us confidence that UML could learn multiple new tasks without catastrophic forgetting.

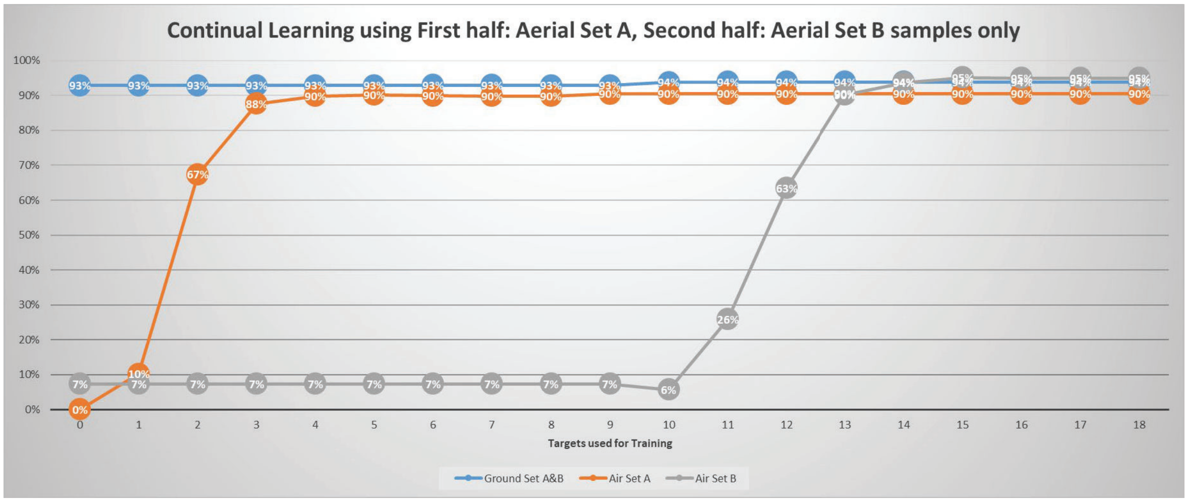


Figure 21. Proof of Concept 2 Results

4.1.3 POC 3

This experiment again follows on after the previous two POC experiments and tests self-supervised acquisition of previously unknown targets, vehicle Set C in this experiment. A new target is presented to the system and triggers an attentional mechanism. In this experiment, the target is an unknown vehicle, which triggers the attentional mechanism due to its similarity to other known vehicles.

The self-supervision mechanism in UML identifies that the object is relevant and begins to learn its appearance for future classification tasks. Table 9 shows the results from POC 3 and confirms that the self-supervised acquisition of new tasks does not interfere with the previously learned tasks, either the initial knowledge, or tasks learned through continual learning. The results from these three proofs of concept gave confidence that the UML algorithm could achieve the full scope of lifelong learning tasks displayed in the Phase 1a demo.

Table 9. Results Showing System Performance Through POC 1-2 and into POC 3.

	Ground – A&B	Aerial – Set A	Aerial – Set B	Aerial – New Set C
1. Initial Performance	99.9	88.9	97.5	0%
2. After Few-shot Learning Aerial C	99.9	88.9	97.5	**42%**

4.2 Phase 1a Demo

This demonstration tied together all the different capabilities demonstrated in the POC experiments in a single continuous "lifetime", where our embodied drone agent, equipped with UML, would interact with the environment, and achieve all the previously demonstrated capabilities, but continuously and in real-time.

There were three major results from the Phase 1a demo that mirror the results shown in POCs 1-3. First, new variants of a previously known task were learned without catastrophic forgetting (POC 1). Second, new variants of a second previously known task are learned without catastrophic forgetting, and backward transfer is observed where previous tasks improve from learning on new tasks. Third, self-supervised acquisition of new tasks was shown to effectively acquire new classes without interfering with old tasks.

The results from this demonstration are shown in a narrative set of images, containing a view of the environment that the embodied drone agent exists within, an inset view of the classification performance of the agent and some text overlay to indicate what is occurring in each frame.

It is important to note the system architecture to correctly interpret these images. The UML algorithm is fed objects through a pretrained object detection algorithm, YOLO v3. This algorithm provides the attentional mechanism that drives UML to classify and learn in a self-supervised manner. However, the detection rate of YOLO v3 is not 100%, and occasionally objects are not detected. UML is only able to operate on detected objects, so a failure in YOLO detection will cause a failure to classify the object in UML.

In the inset classification images for each of the demo stages, some target objects do not have boxes drawn around them. This indicates a failure in the detection stage, and as UML operates afterwards, this causes a lack of classification for that object. For the purposes of evaluating the lifelong learning characteristics of UML, we do not consider YOLO v3 false negatives in the accuracy evaluation of UML

4.2.1 Initial Performance of System

At the beginning of the demonstration, the system exhibits high performance at ground level on sedans (marked in blue in the following figures), but poor performance on the same sedans from an aerial view. This can be seen in Figure 22, where the sedans are misclassified as vans (green in the following figures). This demonstrates the baseline performance of the system before any learning has occurred.

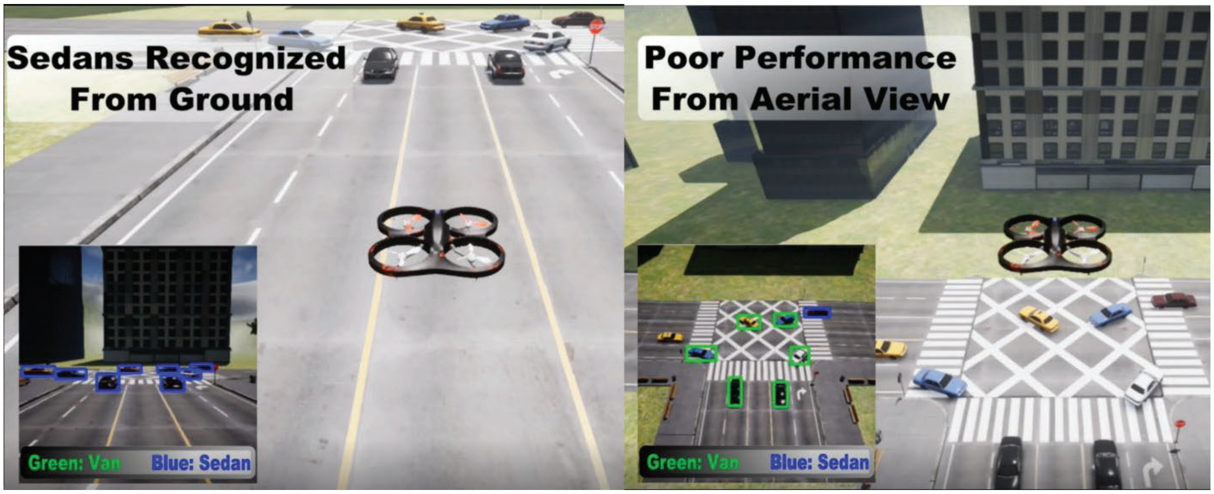


Figure 22. From Demo 1a: Initial Classification Performance of the Drone at Ground and Aerial Viewpoints

4.2.2 Learning of Aerial Views of Sedans

After establishing the baseline performance, the objects are re-acquired from a ground view to establish a self-supervised ground truth, and then the drone flies to the desired altitude and position to acquire the required new skill. The objects are tracked frame to frame to associate the new views with the known object labels, which drives continual learning of the tracked objects as the drone moves. Figure 23 shows the results of this learning, where the previously misclassified sedans are now correctly identified from a new altitude (indicated by the blue boxes on all the targets).

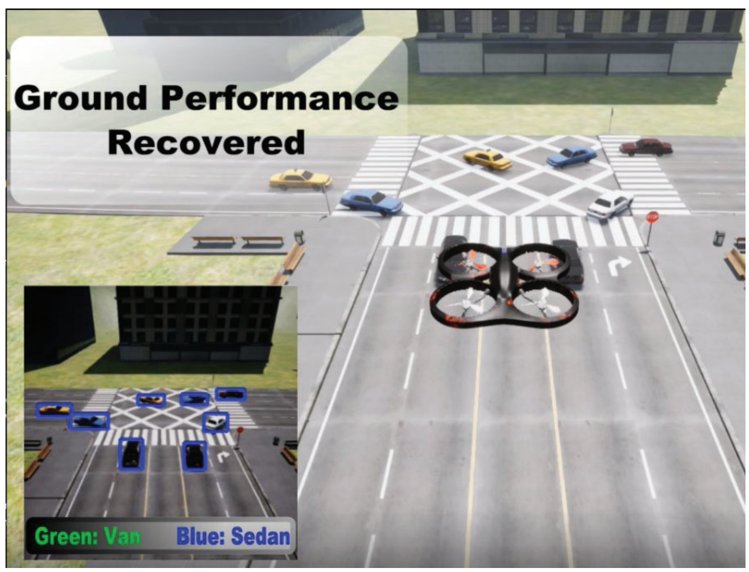


Figure 23. From Demo 1a (Continued): Continual Learning on Sedans Results in Recovered Ground Viewpoint without Catastrophic Forgetting

4.2.3 Learning of Aerial Views of Vans

The drone travels to a new location where vans and sedans are intermixed. The initial performance on the intersection is poor, with some confusion of vans as sedans. Learning occurs in the same manner as before, where the drone acquires the self-supervised ground truth by flying to a location where the objects are expected to be classified correctly, i.e., ground level. The objects are tracked, and the drone moves while engaging continual learning to classify the objects correctly. Figure 24 shows a “before and after” set of images, where the vans are incorrectly classified before continual learning (left) and correctly classified at the new altitude after continual learning (right). Performance on previously learned sedans is maintained.

Figure 24. From Demo 1a (Continued): Performance on a Second Class of Objects Before Continual Learning (Left) and After Continual Learning (Right)

4.2.4 Retention of Performance on First Task

After learning the aerial views of the vans, the drone navigates back to the initial location containing only sedans and verifies that there is no catastrophic interference on the sedan

Figure 25. From Demo 1a (Continued): First Set of Sedans Still Classified Correctly After Learning of New Objects

classification knowledge by learning the van classification task. This is shown in Figure 25, where all the sedans are correctly classified from the aerial viewpoint.

4.2.5 Self-Supervised Acquisition of a New Class

Following the verification that no catastrophic forgetting has occurred in the continual learning process, the drone now moves to a new location where two unknown vehicles, fire trucks, are parked. These vehicles are not part of the knowledge base of YOLO or UML, and as such are not able to be classified correctly. The following figures in this section omit the environmental view containing the drone and the scene and will exclusively contain the classification output of the agent for clarity.

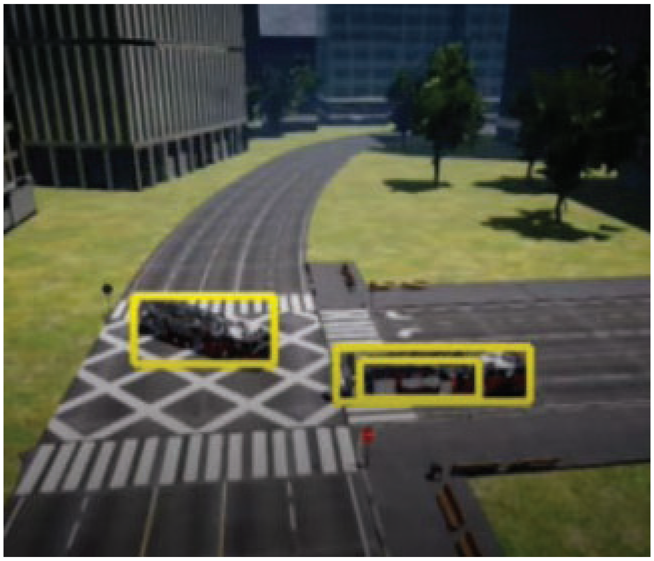


Figure 26. From Demo 1a (Continued): Classification Output When Exposed to "Interesting" but Unknown Objects

The drone approaches the two fire trucks, and detections begin to appear on them. This is due to their similarity to other known objects in the YOLO knowledge base, but the classification of them is incorrect, as neither YOLO nor UML have a label for fire truck. This is shown in Figure 26. The yellow boxes indicate a detection has been generated, but the classification ID is unknown.

These detections in Figure 26 trigger a self-supervised response, where UML engages in an unsupervised learning mode to learn the views of the objects. One consequence of this process is that the two fire trucks are learned as distinct unsupervised objects. The tracking ability ensures that views of the same object are given the same label, but there is no evidence so far to indicate that these objects should share a supervised label. This process is illustrated in Figure 27, where differently colored boxes are drawn around the two fire trucks, indicating they have unique self-generated labels. However, these labels are now fixed and future observations of these unique objects will result in a correct classification.

Once the two objects have been learned, the drone simulates a process by which a human operator can review the self-supervised objects that have been learned and associate and group them with human-meaningful labels. In our demonstration, a pop-up window appears on the user interface asking if the user wants to associate the two objects learned through self-supervision together, with a label of "fire truck". Once the user selects "Yes", the objects are assigned the label and both mapped onto the same class. The result of this process is shown in Figure 28, with both objects sharing a red bounding box, indicating they are fire trucks.

The Phase 1a demo shows that UML exhibits continual learning without catastrophic forgetting and that it can use self-supervision to drive the acquisition of new tasks. Numerical results associated with these capabilities will be explored in the Phase 2 results discussions.

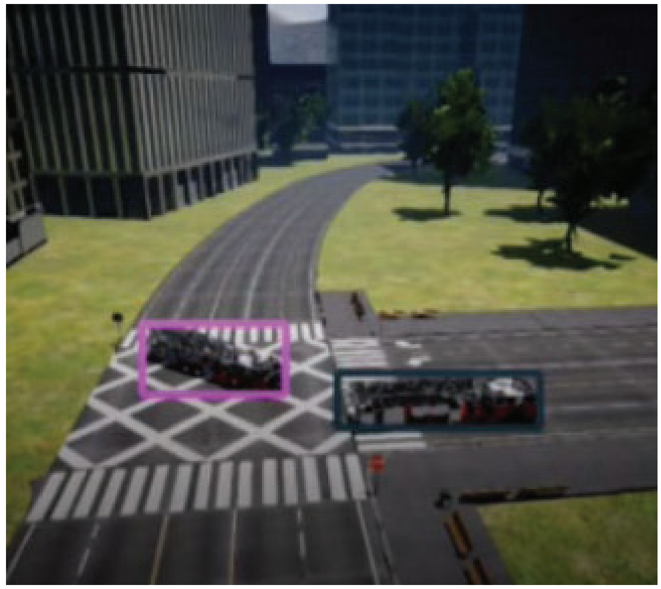


Figure 27: From Demo 1a (Continued): Self-Supervised Learning Process Where Each Unique Object Has a Different Label

4.3 Phase 1b Proof of Concept Experiment

Phase 1b focused on exploring new mechanisms and capabilities that could be supported through the self-supervised properties of UML. A POC experiment (POC 4) was designed to show capability in this area that would lead to a demonstration in a real-time simulation environment.

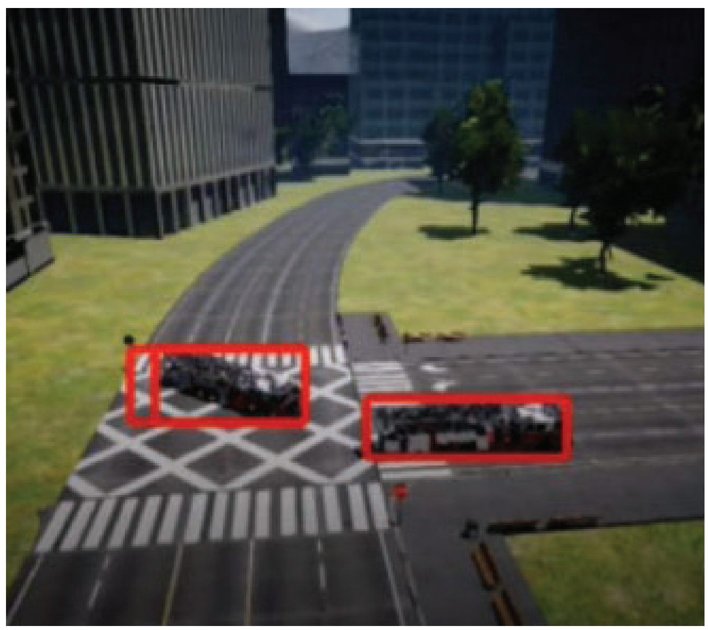


Figure 28. Demo 1a (Continued): Objects Collapsed into Single Class After a Human Operator Provides a Label to Both

UML was arranged in a hierarchical structure with the first level classifying target ID (van, sedan, fire truck, etc.) and the second level assigning a drone behavior to that target (COUNT, AVOID, IGNORE). The second layer is given access to the environmental context to help inform the behavior of the object within its context. In POC 4, the contextual signal was the altitude of the drone. Objects should trigger different behaviors in different contexts, i.e., a ground view of a sedan should trigger an AVOID behavior, while an aerial view of a sedan should trigger a COUNT behavior. Conversely, an aerial view of a traffic light should trigger an AVOID behavior, while a ground view of a traffic light should trigger an IGNORE behavior.

The results of POC 4 show that a hierarchical UML can achieve multiple functions: 1) It can learn multiple hierarchical tasks, 2) It can effectively leverage context to trigger classification changes, 3) It can leverage self-supervision to monitor its own performance and trigger continual learning to recover performance.

Table 10 shows the results from POC 4, which address the three functions stated in the previous paragraph. 1) Continual hierarchical task learning is supported with high accuracy, as shown in the Accuracy after Learning column. The high accuracy of the predicted behavior and class shows that the system can accurately perform both tasks simultaneously. 2) The context is leveraged in the second layer of the hierarchy and contributes to the high accuracy of the system. 3) The two confusion matrices show the performance of the system as it is continually learning the task. In this paradigm, a small, sparse pre-training dataset is provided to seed the algorithm, and then the full dataset is continually learned. The evaluation for each observation takes place prior to the continual learning step, so any improvement on the whole dataset is because of previous continual learning, not learning on the current test sample.

Table 10. POC 4 Results

<table>
<tr><th>Learning Stage</th><th>Incremental Accuracy</th><th>Accuracy after Learning</th><th colspan="4">Confusion Matrix</th></tr>
<tr><td rowspan="5">Initial Training on Sparse Training Data</td><td rowspan="5">1.45%</td><td rowspan="5">1.45%</td><td></td><td>Count</td><td>Avoid</td><td>Ignore</td></tr>
<tr><td>Unknown</td><td>98%</td><td>100%</td><td>100%</td></tr>
<tr><td>Count</td><td>1.99%</td><td>0%</td><td>0%</td></tr>
<tr><td>Avoid</td><td>0.01%</td><td>0%</td><td>0%</td></tr>
<tr><td>Ignore</td><td>0%</td><td>0%</td><td>0%</td></tr>
<tr><td rowspan="5">With Continual Learning</td><td rowspan="5">82.7%</td><td rowspan="5">99.1%</td><td></td><td>Count</td><td>Avoid</td><td>Ignore</td></tr>
<tr><td>Unknown</td><td>16.3%</td><td>12%</td><td>22.5%</td></tr>
<tr><td>Count</td><td>83.67%</td><td>0%</td><td>0.55%</td></tr>
<tr><td>Avoid</td><td>0.03%</td><td>78%</td><td>16%</td></tr>
<tr><td>Ignore</td><td>0%</td><td>10%</td><td>61%</td></tr>
</table>

The confusion matrix following initial sparse training shows the self-supervised introspection of the algorithm, as most targets are classified with an “Unknown” behavior. This unknown signal can be used to trigger further learning, or to indicate to downstream autonomy that the algorithm is unsure and should not be given much weight. However, as the self-supervised continual learning process is run, the dataset becomes much better understood and most of the unknown classifications transition into correct classifications.

One important point on the self-supervised learning that is seen in Table 10 is the relative lack of incorrect classifications. In each column of the confusion matrix, the unknown classification is higher than the sum of all the incorrect classifications. This shows that even after large-scale learning, the self-supervised introspection can identify knowledge gaps and communicate that in many of the incorrect cases.

4.4 Phase 1b Demo

The Phase 1b demonstration was conducted in a similar fashion to the Phase 1a demo, where a live, real-time agent engaged in the desired activities in a simulated environment. A narrative set of figures below shows the agent behavior through the stages of the demonstration. The goals of the Phase 1b demonstration were to 1) display the hierarchical capabilities of UML, 2) show self-supervised acquisition of new behaviors leveraging the UML hierarchy, 3) show those behaviors engaging in a testing location, and 4) show that the hierarchy does not interfere with the previously learned behaviors shown in Phase 1a.

Functionally, this demo was designed to occur directly following the Phase 1a demo, and the knowledge gained in that lifetime was provided to the agent on startup. The drone has been equipped with a proximity sensor. If any object gets within 5 meters of the drone, a predetermined safety behavior is triggered. This self-preservation "instinct" is captured and uses as a supervisory signal for UML, creating the capability in the agent to self-supervise the acquisition and avoidance of obstacles.

4.4.1 Encountering an Obstacle

The agent approaches an obstacle, a street umbrella, and it breaches the self-preservation proximity threshold. This is shown in the top panel of Figure 29. The object is localized and learning begins. The object is assigned a self-supervised label (L1) and the behavior AVOID. The drone begins a learning maneuver, first at low altitudes, 1-5 meters (shown in Figure 29, lower left), then at high altitude, 15-20 meters (shown in Figure 29, lower right). The low altitude learning keeps the assigned AVOID behavior, but at high altitude, the assigned behavior is changed to IGNORE, as it is no longer a threat to the drone's navigation.

Figure 29. Self-Supervised Acquisition of Target-Relevant Behaviors

4.4.2 Encountering a Second Obstacle

The drone then continues its original path, until it encounters another obstacle, this time a traffic light. The same learning behavior is triggered by the proximity instinct, and the traffic light is learned in both low and high contexts. This encounter and learning are shown in Figure 30. The altitude context learned is different from the umbrella, as the traffic light is higher off the ground. 4-9 meters are learned as the context for the AVOID behavior and 20-25 meters are learned as the context for IGNORE.

After the learning of both of these obstacles, the drone continues to the testing area where the pre-emptive avoidance behavior will be tested.

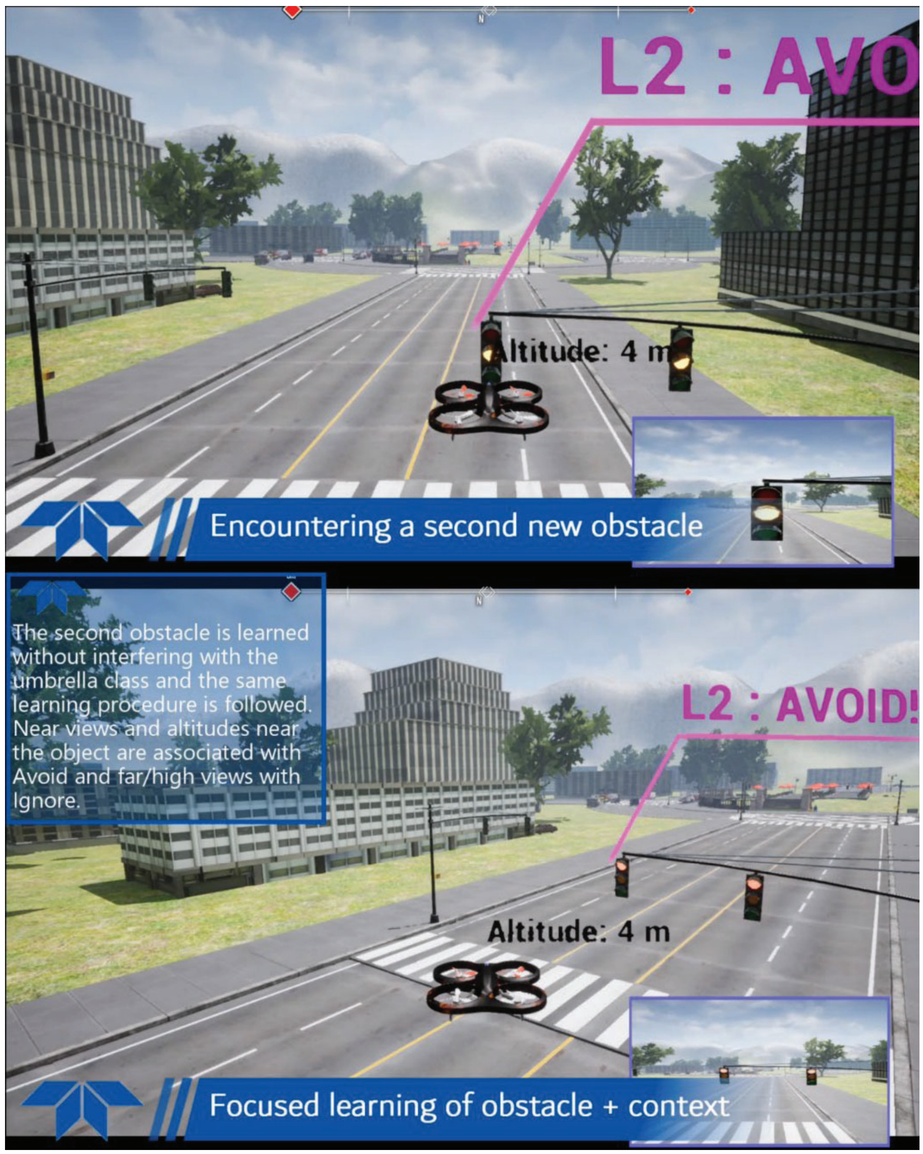


Figure 30. Self-Supervised Acquisition of Target-Relevant Behaviors (Continued): Learning a New Target Does Not Interfere with the Previously Learned Obstacle

4.4.3 Testing Avoidance of Traffic Lights

The drone encounters another traffic light in the navigational path. The drone can detect and plan for the obstacle much further away than previously possible with only the self-supervised proximity instinct. The drone identifies the traffic lights as objects to be avoided, as can be seen in Figure 31. The AVOID behavior triggers an increase in altitude in the drone until the obstacle has been cleared, then the drone returns to the previously planned flightpath.

Figure 31. Self-Supervised Acquisition of Target-Relevant Behaviors (Continued): AVOID Behavior Triggers an Increase in Altitude Until Obstacle can be Cleared.

4.4.4 Testing Avoidance of Street Umbrellas

After the traffic lights have been successfully avoided, the drone encounters a series of street umbrellas, and identifies the close ones as objects to be avoided, shown in Figure 32. The far-field umbrellas are outside of the context under which the AVOID behavior was learned because they are much further away than the training data, and as such are classified as objects to be ignored. The near-field umbrellas are correctly classified as AVOID objects, and

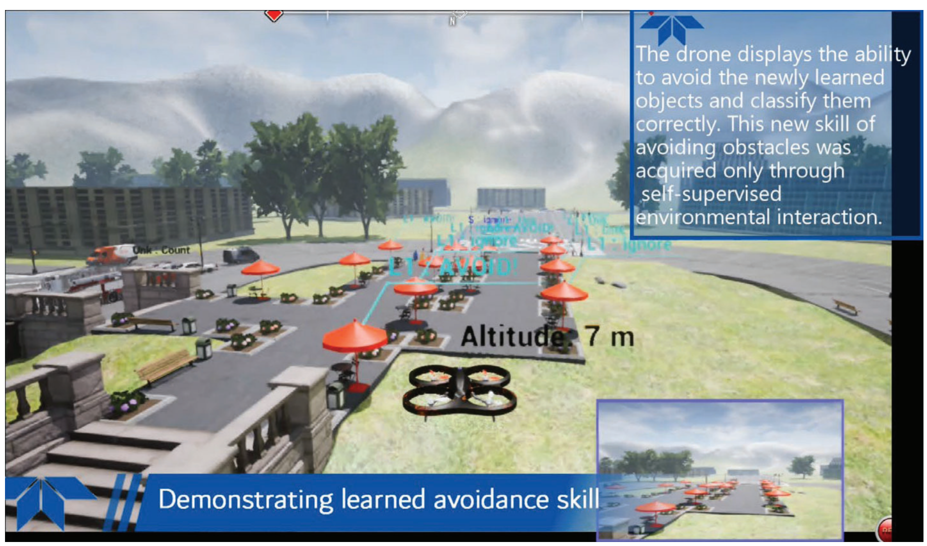


Figure 32. Self-Supervised Acquisition of Target-Relevant Behaviors (Continued): Both the Viewpoint and Context of Object are Considered in Triggering the Avoidance Skill

the drone engages an increase in altitude in response to the umbrella, successfully avoiding a collision.

4.4.5 Retention of Previous Skills

In the final stage of the Phase 1b demonstration, the drone flies to a testing intersection containing sedans, vans, and fire trucks. The system can correctly identify all these objects and assign the COUNT behavior to them. Figure 33 shows this behavior. This shows that the drone is able to learn new objects and new behaviors associated with those objects without any catastrophic forgetting taking place over the old objects and behaviors.

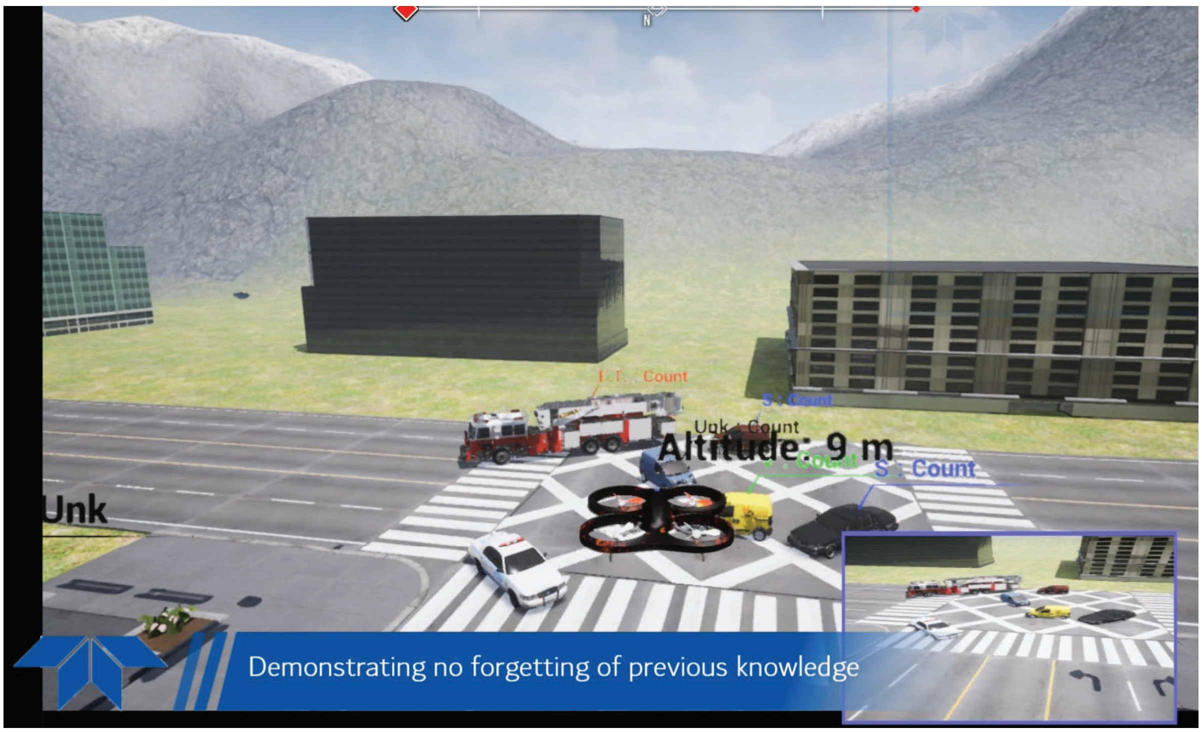


Figure 33. Self-Supervised Acquisition of Target-Relevant Behaviors (Continued): Retention of Previously Learned Skills from Demo 1a

4.5 Phase 2

In Phase 2, the program focus shifted from demonstration of capabilities to measurement of system performance. There were four evaluations that Teledyne participated in as the leader of a System Group (SG). The Teledyne lifelong learning architecture that comprised this system is fully described in Section 3.1.

The evaluations focused on generating statistically significant results in the five program metrics, Performance Maintenance (PM), Forward Transfer (FT), Backward Transfer (BT), Sample Efficiency (SE) and Single-Task-Expert (STE) Relative Performance (RP). As described in Section 3.3, these program metrics are secondary metrics, computed from a primary measurement of the system performance called the “application metric”. For the Teledyne SG, the application metric was classification accuracy in M9 and M12 evaluation periods, and classification precision in M15 and M18 periods. The metric was switched to better capture the performance of the system as is operated, due to saturation in the calculated accuracy metric that negatively impacted the sensibility of the program metrics. The results from M9 and M12 are thus not directly comparable to the results in the M15 and M18 evaluations, as the application metric changed. However, we computed some of the M12 results again using the new application metric to provide a bridge to the new results showing how we are still improving over time.

4.5.1 Overview and Final Results

In the Month 18 (M18) evaluation, the Teledyne SG presented results showing that their lifelong learner met or exceeded the lifelong learning threshold in all five program metrics and exceeded program targets in two of five metrics. The scores with standard deviations for each of the five metrics are shown below in Table 11, with light green indicating that a metric exceeds the lifelong learning threshold, and darker green indicating that a metric exceeds the DARPA program target. This color legend is used throughout the rest of the Phase 2 evaluation results. A "*" in the table indicates a statistical significance of $p < .05$ and "**" indicates statistical significance of $p < .01$.

Table 11. Final Evaluation Results of Teledyne Lifelong Learner

	Performance Maintenance	Forward Transfer	Backward Transfer	Relative Performance	Sample Efficiency
M18 Agent	**0.56**** ± 0.98	11.69 ± 0.47	**1.00*** ± 0.01	**1.03**** ± 0.04	**2.74*** ± 1.70

These results indicate that the Teledyne SG lifelong learner, with UML at its core, is a complete lifelong learning system. Going from left to right on the table, the results can be interpreted by the following statements:

- **Performance Maintenance**
  - Performance on a learned task improves by 0.56% on average as more tasks are learned.
- **Forward Transfer**
  - Initial task performance on a new task T is 11x better after learning a different task as compared to the baseline performance of T at the start of the lifetime
- **Backward Transfer**
  - Performance on a previously learned task is the same after learning a different task
- **Relative Performance**
  - The lifelong learning agent is 1.03x better at learning task T when it can leverage information from other learned tasks, relative to learning task T in isolation
- **Sample Efficiency**

- The lifelong learning agent is 2.74x more efficient (in terms of required learning experiences) at learning a task T when it can leverage knowledge from other learned tasks, relative to learning task T in isolation

Taken as a whole, these results show that the Teledyne SG lifelong learning agent exhibits the characteristics of a lifelong learner, mitigating catastrophic forgetting, leveraging new and old task knowledge to improve task performance and learning speed. The following sections will describe the results of each of the evaluations individually.

4.5.2 Month 9 Evaluation

This was the first system evaluation that included the combination of multiple lifelong learning components into a full system. For full details of the experimental setup, see Section 3.6.7. Simply, there were 3 tasks that were permuted in their order and learned one time to form the permuted scenario. Two of the three tasks were selected and alternated learning 3 times to form the alternating scenario. There were 4 runs completed for each of the permuted and alternating, however due to time constraints only 3 permuted runs were submitted to APL. The results shown in Table 12 are the official evaluation results generated by APL from the 3 submitted permuted runs in the M9 evaluation period, apart from the performance recovery metric. The Performance Recovery metric was not generated by APL in their official results but was generated independently by Teledyne using 4 runs of the alternating scenario. Future evaluations would eventually remove this metric as it is ill-behaved, but it is included here for completeness. The second row of Table 12 provides interpretations of the results.

Table 12. M9 Evaluation Results and Interpretation

Performance Maintenance	Forward Transfer	Backward Transfer	Performance Recovery	Relative Performance	Sample Efficiency
-17.52 ± 4.24	1.19 ± 0.07	0.99 ± 0.01	-28.5	2.61 ± 0.18	1.78 ± 0.14
17.5% worse average task performance after learning more tasks	1.19x on new tasks after learning	0.99x on old tasks after learning	28% slower to recover performance after learning new task than initial learning	2.6x better performance after learning other tasks relative to independent learning	1.78x faster learning after learning other tasks

4.5.3 Month 12 Evaluation

In the M12 evaluation, the same scenario types were used as in M9: permuted and alternating. However, this evaluation did not define three tasks. Instead, two tasks with two variants each were chosen as to form the basis for the learning scenarios. See Section 3.6.7 for a complete description of the experimental setup.

After submitting results to APL, we discovered a discrepancy that impacted the metric calculation in our evaluation blocks. Table 13 shows both the APL submitted metrics and the corrected metrics after discovering the calculation issue. The interpretation of the metrics (final row of the table) will be relative to the corrected metrics (M12'). Also, after the M12 evaluation it was determined by the program stakeholders that the performance recovery metric was too ill-behaved to consider, and as such it stopped being reported as a program metric. The M12' results shown below therefore do not contain a performance recovery metric value.

Table 13. M12 and M12' Results and Interpretation

	Performance Maintenance	Forward Transfer	Backward Transfer	Performance Recovery	Relative Performance	Sample Efficiency
M12 (APL)	5.63 ± 4.88	1.49 ± 0.31	1.19 ± 0.16	-31.88 ± 61.81	1.0 ± 0.01	2.16 ± 3.28
M12'	1.99 ± 3.22	1.80 ± 0.22	0.99 ± 0.01	Not calculated	1.0 ± 0.01	2.16 ± 3.28
	1.99% better performance on learned tasks after learning more tasks	1.8x better on new tasks after learning	0.99x on old tasks after learning new ones		The same performance after learning on other tasks	2.16x faster learning after learning other tasks

4.5.4 Month 15 Evaluation

In the M15 evaluation, the scenario types were changed to a "condensed" scenario, where each task was learned in a single continuous block, and a "dispersed" scenario, where each task learning block was split into 3 mini-blocks. In this evaluation, the application metric changed from accuracy to precision, and thus direct comparison is not possible between M15 and previous evaluation results. To address this, the M12 permuted scenario was run collecting the new precision application metric with both the M12 agent and the M15 agent. This shows the improvement from M12 to M15.

Table 14 shows the M15 scenario results in the top panel and the M12 scenario results in the bottom panel. This evaluation was also focused on generating more statistical significance in the results, and included 33 condensed runs, 34 dispersed runs, and 24 permuted runs. The precision metric better reflected the performance of the system and provided more meaningful metrics, but also resulted in a reduction in metric performance in relative performance and performance maintenance. Every metric apart from Relative Performance improved from M12 to M15, showing increased capability of the M15 agent.

Table 14. M15 Evaluation Results Compared to M12 Evaluation Results

	Performance Maintenance	Forward Transfer	Backward Transfer	Relative Performance	Sample Efficiency
M15 Scenarios					
M15 Both Scenarios	-3.20 ± 4.14	10.51 ± 1.35	0.99 ± 0.04	0.92 ± 0.04	2.20 ± 2.91
M15 Condensed	-5.24 ± 4.92	11.02 ± 1.28	0.98 ± 0.05	0.94 ± 0.04	2.62 ± 3.26
M15 Dispersed	-1.23 ± 1.59	10.01 ± 1.23	1.01 ± 0.03	0.90 ± 0.04	1.79 ± 2.56
M12 Scenarios					
M15 on M12 Permuted	-25.26 ± 5.96	1.46 ± 0.5	0.94 ± 0.32	0.95 ± 0.00	0.64 ± 0.26
M12 on M12 Permuted	-32.24 ± 6.31	1.05 ± 0.00	0.65 ± 0.1	0.99 ± 0.00	0.26 ± 0.14

4.5.5 Month 18 Evaluation

In the M18 evaluation, the condensed scenario was chosen as the primary evaluation scenario, but the dispersed runs were still performed to provide supplemental results. The desired statistical significance of the metrics specified a target of 12 runs per scenario. The goal of the agent improvement in this evaluation was to bring all the metrics above the lifelong learning threshold, and specific focus was paid to the performance maintenance metric. For specifics of how we improved the system by M18, see Table 8.

Table 15 shows the results of the M15 agent and M18 agent on the condensed and dispersed scenarios. The condensed scenario results show an improvement in all metrics from M15 to M18, including meeting the lifelong learning threshold in all metrics and exceeding the program target values in Forward Transfer and Sample Efficiency. The dispersed scenario also shows an improvement in all metrics from M15 to M18. The dispersed metrics exceed the program target in Forward Transfer, meet the lifelong learning threshold in three of five metrics (Performance Maintenance, Backward Transfer and Sample Efficiency) and are within the standard deviation of the lifelong learning threshold in the other two metrics (Performance Maintenance, and Relative Performance).

Table 15. M18 Evaluation Results Compared to M15 Evaluation Results

	Performance Maintenance	Forward Transfer	Backward Transfer	Relative Performance	Sample Efficiency
M15 Condensed Scenario					
M18 Agent	**0.56**** ± 0.98	11.69 ± 0.47	**1.00*** ± 0.01	**1.03**** ± 0.04	**2.74*** ± 1.70
M15 Agent	-1.94 ± 2.33	**12.31** ± 2.19	0.99 ± 0.02	0.97 ± 0.03	1.73 ± 1.54
M15 Dispersed Scenario					
M18 Agent	**-0.18*** ± 2.15	**9.77**** ± 0.77	1.00 ± 0.01	**0.96**** ± 0.11	**1.45*** ± 0.40
M15 Agent	-1.04 ± 1.05	8.70 ± 1.11	1.00 ± 0.02	0.92 ± 0.01	1.18 ± 0.34

Additionally, an ablation study was conducted with the memory optimization component C5 to assess the impact of memory optimization alone on the L2M metrics (see Table 16 below for a summary of results). We found that memory consolidation led to improvements in forward transfer, relative performance, and sample efficiency. However, we did see a slight decrease in relative performance and a larger decrease in performance maintenance when the memory consolidation component was active.

Table 16. Performance Results with and without Memory Optimization Component C5

	Performance Maintenance		Forward Transfer		Backward Transfer		Relative Performance		Sample Efficiency	
	Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.
M15 Condensed Scenario										
With C5	0.56	0.98	**11.69****	0.47	1.00	0.01	**1.03**	0.04	**2.74**	1.70
Without C5	**1.68**	0.77	10.47	0.46	**1.02**	0.03	1.01	0.05	2.33	1.47

4.6.1 Character Recognition

This experiment examined how uncertainty could be used to promote adaptation in deep neural networks under noisy conditions. Here, a short convolutional neural network was first trained to recognize hand-written characters with the MNIST dataset. The first two bars in Figure 34 show the performance of the trained network on the clean MNIST test set and the low-contrast, noise-added noisy MNIST dataset; the drop in performance from 98% accuracy on clean imagery to 74.1% on the noisy imagery reflects a drop in performance caused by alterations to the original inputs. In a real-world scenario, this could be representative of sensor noise or similar issues that often arise when using a system trained on ideal imagery on more realistic information.

Subsequently, the network was assumed to recognize this shift in performance or in image quality, and a learning cycle was triggered to continually learn on the new imagery, but with modulated hyperparameters designed to retain prior performance while adapting to the new circumstances. After such training with the noisy MNIST dataset without re-presenting the clean dataset, the last two bars in Figure 34 show the performance of the network on the clean and noisy datasets. The third bar shows that performance on the noisy dataset recovered from 74% to 92.4%, significantly boosting performance on the corrupted imagery; the fourth bar shows that performance on the clean imagery remained high at 96.5%, retaining performance under prior conditions without catastrophic forgetting. Together, these results

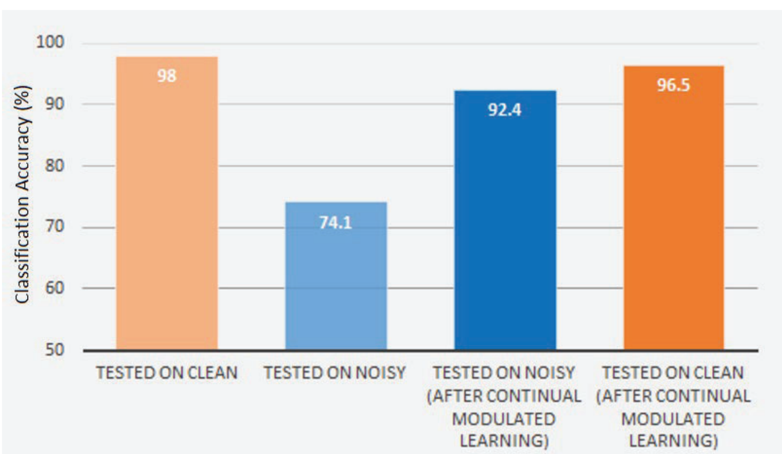


Figure 34. Continual Modulated Learning with Noisy Imagery

suggested that recognizing uncertainty and adjusting learning parameters under such conditions can aid in continual learning of deep neural networks to adapt to new conditions.

4.6.2 Reinforcement Learning

In this experiment, described in Section 3.6.8, UML was tasked to identify out-of-nominal conditions and modulate the action chosen by the RL agent in response to the high uncertainty. In this experiment the action modulation was fixed, and replaced the chosen action (LEFT, RIGHT, NO-OP) with a NO-OP action. This caused the agent to pause in the presence of uncertainty, instead of performing a potentially detrimental action. Figure 35 shows example out-of-nominal conditions on the top of the figure (Additive Noise 10% and 30%, Vertical Perturbation, Horizontal Perturbation), and these are matched with the baseline and UML-enabled performance.

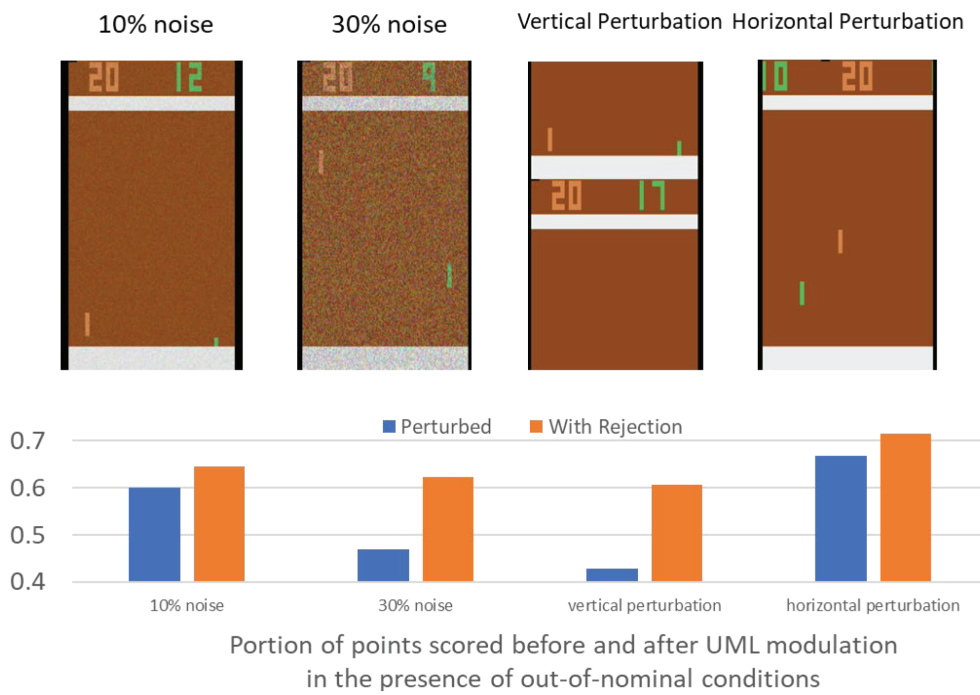


Figure 35. Noise Rejection in Reinforcement Learning

These results show the fraction of points scored as compared to the nominal condition with and without UML modulating the action in response to uncertainty. In the 10% noise case, we see that the baseline agent scored 60% of the points that were available, while enabling UML rejection resulted in capturing 65% of the available points. The 30% noise case was more challenging and showed a greater impact from UML intervention with a 15% difference between the scores. Horizontal perturbation was not a particularly challenging case, but UML still resulted in a 3% increase in score. Vertical perturbation was the most challenging case and showed the largest difference in baseline vs. UML score. The baseline agent scored 18% lower than the UML-equipped agent. These results clearly show that UML can detect out-of-nominal conditions and be used to improve an existing agent's performance through monitoring and responding to high uncertainty situations.

5.0 CONCLUSIONS

Our efforts and results have demonstrated the role of biologically-inspired algorithms in advancing the state-of-the-art in machine learning. In particular, neuromodulation-inspired mechanisms enabled self-supervised adaptation in small networks, multi-layered neural network architectures, and autonomous agents (e.g., simulated drone). UML was validated as a viable algorithm that can be "plugged-in" to state-of-the-art (SOTA) machine learning systems to support lifelong learning capabilities such as robustness under novel conditions and continual learning. Figure 36 presents three different scenarios that were demonstrated during the program and a description of the role of UML in each one of them.

We showed that our lifelong learning algorithm, UML, is capable of supporting self-supervised, online, continual learning in an operationally relevant domain (drone-based object recognition). We also showed that UML can also self-supervise the detection of out-of-nominal states without explicit labels, both in an object classification task and a reinforcement learning task, demonstrating the broad capabilities of the UML algorithm.

We showed that an integrated lifelong learning system has critical components that enable its performance: Attentional mechanisms are critical to manage the information flow within the agent, uncertainty mechanisms are critical to measure internal and external state against expected

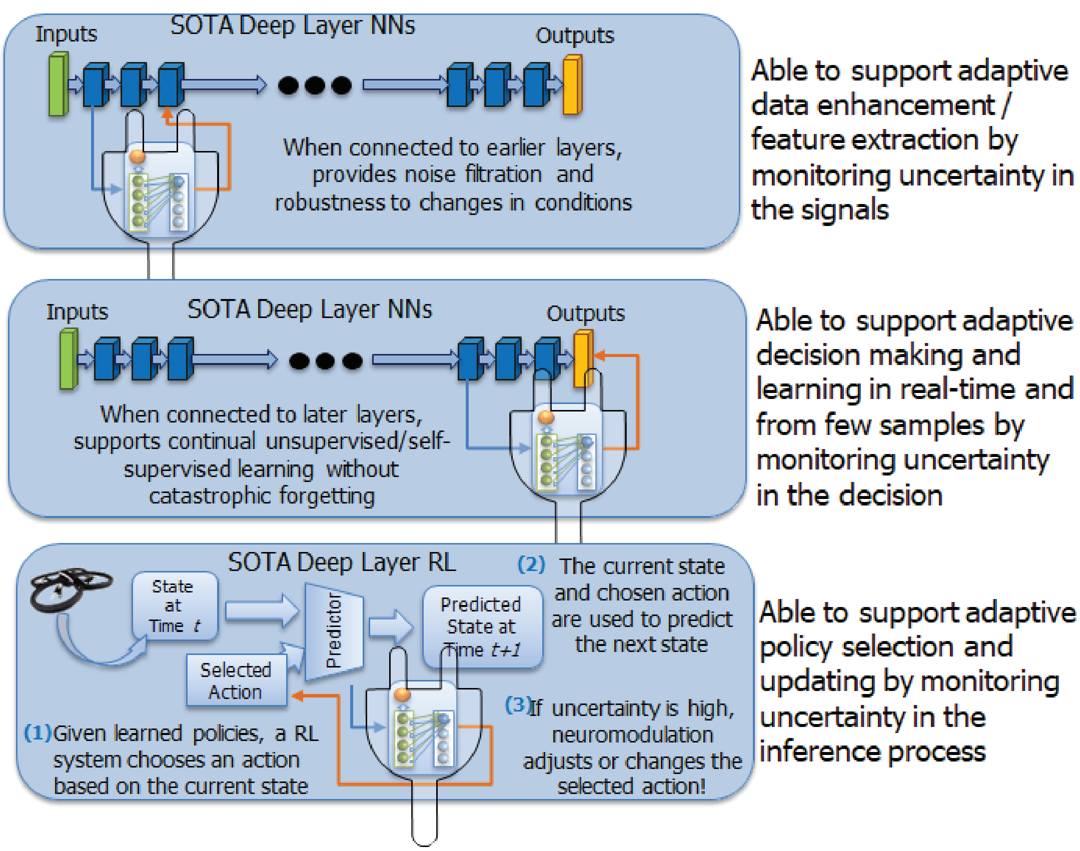


Figure 36. UML is a Pluggable Component that can Support Lifelong Learning Capabilities in State-of-the-Art Machine Learning Systems

performance and drive modulatory responses, and hierarchical mechanisms are critical to derive robust knowledge representations within the agent's short and long term memory.

We also showed that our system engineering approach of interconnected but independent components enabled ablation experiments and lifelong learning evaluations of the whole system and individual components simultaneously. Our system also showed improvement over the course of the program. Table 17 shows the progression of our lifelong learning system against the five core program metrics as documented by our team and validated by APL. Red boxes indicate that performance did not meet the lifelong learning threshold; light green boxes indicated that performance met the lifelong learning threshold, and darker green boxes indicated that performance exceeded the program target. We reached the lifelong learning threshold in all five in M18 while starting only reaching one metric in M12. This improvement was made through both individual component capability increases, as well as better system engineering to connect the components together.

Table 17. Progression of Teledyne System against Program Metrics as Documented by APL

	PM			FT			BT			RP			SE		
	M12	M15	M18	M12	M15	M18	M12	M15	M18	M12	M15	M18	M12	M15	M18
Teledyne															

Top-down modulation in a hierarchical learning system improves the lifelong learning performance of the system. Leveraging the uncertainty generated by mismatch in the hierarchical knowledge base induces task relevant knowledge structures as well as enforces well-conditioned outputs. Introducing this mechanism led to an improvement across all tracked program metrics.

We have also shown that memory consolidation mechanisms can complement online continual learning. Our C5 module showed improvements in transfer and performance against single task experts in the ablation study, confirming that memory optimization is a relevant mechanism to improve performance in a multi-task learning setting.

Finally, we showed that uncertainty-based modulation supports robust lifelong learning, particularly short and long-term adaptation. Our agent can self-supervise both the learning and inference processes in response to measured uncertainty, which can be driven by changes in environment, task, or the agent itself.

6.0 WORKS CITED

1. Carpenter, G.A. and S. Grossberg, *Pattern recognition by self-organizing neural networks*. 1991, Cambridge, Mass.: MIT Press. 691 p.

2. Kravitz, D.J., et al., *The ventral visual pathway: an expanded neural framework for the processing of object quality.* Trends in Cognitive Sciences. **17**(1): p. 26-49.

3. Angela, J.Y. and P. Dayan, *Uncertainty, neuromodulation, and attention.* Neuron, 2005. **46**(4): p. 681-692.

4. Hangya, B., et al., *Central cholinergic neurons are rapidly recruited by reinforcement feedback.* Cell, 2015. **162**(5): p. 1155-1168.

5. Atri, A., et al., *Blockade of central cholinergic receptors impairs new learning and increases proactive interference in a word paired-associate memory task.* Behavioral neuroscience, 2004. **118**(1): p. 223.

6. Buccafusco, J.J., et al., *Long-lasting cognitive improvement with nicotinic receptor agonists: mechanisms of pharmacokinetic–pharmacodynamic discordance.* Trends in pharmacological sciences, 2005. **26**(7): p. 352-360.

7. Hasselmo, M.E. and J. McGaughy, *High acetylcholine levels set circuit dynamics for attention and encoding and low acetylcholine levels set dynamics for consolidation.* Progress in brain research, 2004. **145**: p. 207-231.

8. Levin, E.D., F.J. McClernon, and A.H. Rezvani, *Nicotinic effects on cognitive function: behavioral characterization, pharmacological specification, and anatomic localization.* Psychopharmacology, 2006. **184**(3-4): p. 523-539.

9. Kilgard, M.P. and M.M. Merzenich, *Cortical map reorganization enabled by nucleus basalis activity.* Science, 1998. **279**(5357): p. 1714-1718.

10. Minces, V., et al., *The role of visual cortex acetylcholine in learning to discriminate temporally modulated visual stimuli.* Frontiers in Behavioral Neuroscience, 2013. **7**(16).

11. Chubykin, Alexander A., et al., *A Cholinergic Mechanism for Reward Timing within Primary Visual Cortex.* Neuron. **77**(4): p. 723-735.

12. Muñoz, W. and B. Rudy, *Spatiotemporal specificity in cholinergic control of neocortical function.* Current opinion in neurobiology, 2014. **26**: p. 149-160.

13. Eickhoff, S.B., et al., *Organizational principles of human visual cortex revealed by receptor mapping.* Cerebral Cortex, 2008. **18**(11): p. 2637-2645.

14. Disney, A.A., H.A. Alasady, and J.H. Reynolds, *Muscarinic acetylcholine receptors are expressed by most parvalbumin-immunoreactive neurons in area MT of the macaque.* Brain and behavior, 2014. **4**(3): p. 431-445.

15. Disney, A.A., K.V. Domakonda, and C. Aoki, *Differential expression of muscarinic acetylcholine receptors across excitatory and inhibitory cells in visual cortical areas V1 and V2 of the macaque monkey.* Journal of Comparative Neurology, 2006. **499**(1): p. 49-63.

16. Amar, M., et al., *Blockade of different muscarinic receptor subtypes changes the equilibrium between excitation and inhibition in rat visual cortex.* Neuroscience, 2010. **169**(4): p. 1610-1620.

17. Taylor, P., et al., *The global landscape of cognition: hierarchical aggregation as an organizational principle of human cortical networks and functions.* Scientific Reports, 2015. **5**: p. 18112.

18. Brito da Silva, L.E., I. Elnabarawy, and D.C. Wunsch, *Distributed dual vigilance fuzzy adaptive resonance theory learns online, retrieves arbitrarily-shaped clusters, and mitigates order dependence.* Neural Networks, 2020. **121**: p. 208-228.

19. Brna, A.P., et al., *Uncertainty-based modulation for lifelong learning.* Neural Networks, 2019. **120**: p. 129-142.

20. Grossberg, S., *Competitive learning: From interactive activation to adaptive resonance.* Cognitive Science, 1987. **11**(1): p. 23-63.

21. Yu, A.J. and P. Dayan, *Uncertainty, Neuromodulation, and Attention.* Neuron, 2005. **46**(4): p. 681-692.

22. Grossberg, S., J. Palma, and M. Versace, *Resonant Cholinergic Dynamics in Cognitive and Motor Decision-Making: Attention, Category Learning, and Choice in Neocortex, Superior Colliculus, and Optic Tectum.* Frontiers in Neuroscience, 2016. **9**(501).

23. Dayan, P. and A.J. Yu, *Phasic norepinephrine: A neural interrupt signal for unexpected events.* Network: Computation in Neural Systems, 2006. **17**(4): p. 335-350.

24. Kravitz, D.J., et al., *The ventral visual pathway: an expanded neural framework for the processing of object quality.* Trends in Cognitive Sciences, 2013. **17**(1): p. 26-49.

25. Disney, A.A., H.A. Alasady, and J.H. Reynolds, *Muscarinic acetylcholine receptors are expressed by most parvalbumin-immunoreactive neurons in area MT of the macaque.* Brain and Behavior, 2014. **4**(3): p. 431-445.

26. Grossberg, S., *Acetylcholine Neuromodulation in Normal and Abnormal Learning and Memory: Vigilance Control in Waking, Sleep, Autism, Amnesia and Alzheimer's Disease.* Frontiers in Neural Circuits, 2017. **11**(82).

27. Dasgupta, R., F. Seibt, and M. Beierlein, *Synaptic release of acetylcholine rapidly suppresses cortical activity by recruiting muscarinic receptors in layer 4.* Journal of Neuroscience, 2018: p. 0566-18.

28. Gu, Z. and Jerrel L. Yakel, *Timing-Dependent Septal Cholinergic Induction of Dynamic Hippocampal Synaptic Plasticity.* Neuron, 2011. **71**(1): p. 155-165.

29. Amis, G.P. and G.A. Carpenter. *Default ARTMAP 2*. in *2007 International Joint Conference on Neural Networks*. 2007.

30. Redmon, J. and A. Farhadi, *YOLOv3: An Incremental Improvement.* arXiv, 2018.

31. New, A., et al., *Lifelong Learning Metrics.* arXiv preprint arXiv:2201.08278, 2022.

32. Shah, S., et al. *Airsim: High-fidelity visual and physical simulation for autonomous vehicles*. in *Field and service robotics*. 2018. Springer.

33. Bochkovskiy, A., C.-Y. Wang, and H.-Y.M. Liao *YOLOv4: Optimal Speed and Accuracy of Object Detection*. arXiv e-prints, 2020. arXiv:2004.10934.

34. Lin, T.-Y., et al., *Microsoft COCO: Common Objects in Context.* CoRR, 2014. **abs/1405.0312**.

35. Lecun, Y., et al., *Gradient-based learning applied to document recognition.* Proceedings of the IEEE, 1998. **86**(11): p. 2278-2324.

36. Basu, S., et al., *Learning sparse feature representations using probabilistic quadtrees and deep belief nets.* Neural Processing Letters, 2017. **45**(3): p. 855-867.

LIST OF SYMBOLS, ABBREVIATIONS, AND ACRONYMS

ACRONYM	**DESCRIPTION**
ACh	Acetylcholine
AFRL	Air Force Research Laboratory
AI	Artificial intelligence
APL	Johns Hopkins University Applied Physics Laboratory
ANN	Artificial neural network
ART/ARTMAP	Adaptive Resonance Theory / Mapping algorithm
ATR	Automatic target recognition
BT	Backward transfer metric
CAML	DARPA's Competency Aware Machine Learning program
C[#]	Component [#]; e.g. C1 denotes Component 1 (in the context of our L2M architecture)
COCO	Common Objects in Context
CPU	Central processing unit
DARPA	Defense Advanced Research Projects Agency
DDVFA	Distributed Dual Vigilance Fuzzy ARTMAP
DOT	Department of transportation resources
EMA	Emergency management agency vehicles
ETS	Experience to Saturation
FT	Forward transfer metric
GPU	Graphics processing unit
ID	Identity
I/O	Input/output
L2	Lifelong learning
L2M	Lifelong Learning Machines
M[#]	Month [#] into Phase 2; e.g. M18 corresponds to Month 18 (in context of Phase 2 Government evaluations)
M2	Muscarinic receptor subtype
M4	Muscarinic receptor subtype
ML	Machine learning
MLP	Multi-layer perceptron
MNIST	Modified National Institute of Standards and Technology
NBM	Nucleus basalis of Meynert
PFC	Pre-frontal cortex
PM	Performance maintenance metric
POC	Proof of concept
PNN	Plastic nodal network
PR	Performance recovery metric
RGB	Red, green, blue
RL	Reinforcement learning
ROC	Receiver operating characteristic
RP	(Single-task expert) Relative performance metric

S&T	Missouri University of Science and Technology
SAR	Synthetic aperture radar
SE	Sample efficiency metric
SECTR-CL	Seeker Cost-Transformation Closed-Loop program
SG	Systems group
SE	Sample efficiency metric
STE	Single-task expert
SVM	Support vector machine
T&E	Test & Evaluation
Thal	Thalamus
TRACE	DARPA's Target Recognition and Adaptation in Contested Environments program
UCI	University of California, Irvine
UCSD	University of California, San Diego
UML	Uncertainty-Modulated Learning
V1-V4	Areas 1-4 of the visual cortex
YOLO	You Only Look Once object detection algorithm