Survival-Critical Machine Learning

Eric Mark Sturzinger

CMU-CS-25-113
April 2025

Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Mahadev Satyanarayanan, Chair
Padmanabhan Pillai
Jeff Schneider
Rashmi Vinayak

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science.

This research was sponsored by the U.S. Army Research Office and the U.S. Army Futures Command under Contract No. W519TC-23-C-0003, by the United States Navy under award number N00174-23-1-0001, and by the National Science Foundation under grant number CNS-2106862. The content of the information does not necessarily reflect the position or the policy of the government and no official endorsement should be inferred. This work was done in the CMU Living Edge Lab, which is supported by Intel, Arm, Vodafone, Deutsche Telekom, CableLabs, Crown Castle, InterDigital, Seagate, Microsoft, the VMware University Research Fund, IAI, and the Conklin Kistler family fund. Any opinions, findings, conclusions or recommendations expressed in this document are those of the authors and do not necessarily reflect the view(s) of their employers or the above funding sources.

Keywords: Survivability, Autonomous System, Edge Computing, Live Learning, Adversarial Environments

I would not have been able to complete this program without the unwavering support of my family. To my wife, Christina, who has supported me my entire career, especially during long periods of separation. To my kids, Marcus, Dominic, and Lauren, for their understanding and resiliency through many relocations.

Abstract

Autonomous systems must be able to survive in adversarial or hostile environments where threats evolve and morph. Under conditions in which a class of adversarial agents is novel but rare, these systems must rapidly learn and adapt. We introduce Survival-Critical Machine Learning (SCML), a new ML paradigm that defines how autonomous systems that rely on machine learning can negotiate such adversarial environments. Inspired by the ability of a biological entity's immune system to develop defenses against new viruses, SCML systems leverage the workflow of Live Learning to iteratively improve ML models for threat detection.

Beyond the conceptualization of SCML, the main contributions of this dissertation are an analytical model, a prototype implementation, and experimental results of the SCML design tradeoff space. We evaluate the impact on survivability of the various design parameters and demonstrate the intimate relationship between SCML and Live Learning. Notably, we evaluate the impact of the availability of finite countermeasures (CMs), the CM deployment threshold, the number of deployed systems, and the average threat arrival rate, among others, on the probability of survival of a given mission duration. We also examine various mission success criteria and the effects of divergent performance metrics between SCML and Live Learning. Additionally, we model SCML as a Markov Decision Process (MDP) to demonstrate how it can be analyzed within existing, well-understood ML frameworks such as MDPs and Reinforcement Learning (RL). We also demonstrate Live Learning's extensibility to domains other than visual data, such as short range radar, critical in many environments where SCML systems will need to operate.

Our experimental results confirm that learning can indeed improve survivability in an SCML system. It further shows that the CM deployment threshold and the number of available CMs have a significant impact on survivability. Allowing flexibility in the CM deployment threshold during the mission enhances such survivability under most conditions. Similarly, Live Learning improves the probability of mission success by increasing the likelihood of accurately classifying actual threats (true positives) and decreasing the likelihood of wasting CMs on non-threats (false positives). By defining an SCML MDP, we also show how an SCML system can optimally adjust its CM deployment threshold as a function of state, defined by the number of remaining CMs and the time until mission completion.

Acknowledgments

I have learned a great deal over the last few years from my colleagues at Carnegie Mellon University. I would first like to express my gratitude to my advisor, Prof. Mahadev Satyanarayanan (Satya), for supporting me in completing this program on a constrained timeline. He helped appropriately scope my research topic and provided invaluable guidance and feedback as it evolved and developed over time. I am also thankful for the advice and feedback of the other members of my committee: Padmanabhan Pillai, Jeff Schneider, and Rashmi Vinayak. Their recommendations enabled me to sharpen various parts of my research and strengthen the overall contribution.

I would like to thank everyone else I have worked with at CMU. Specifically, I collaborated the most with Shilpa George and Jan Harkes. Their technical advice and explanations of complex concepts helped me define problems and formulate solutions to solve them. Jan was instrumental in helping me think through the practical challenges of new ideas and their many associated tradeoffs. He also helped greatly in software development, maintenance, and organization, which allowed for rapid design, implementation, and testing of different features. I am also thankful for Tom Eiszler, Mihir Bala, Anthony Chen, Aditya Chanana, Qifei Dong, Jingao Xu, Roger Iyengar, Jason Choi, Jinho Yi, Jim Blakely, and Emily Spencer.

Finally, I would like to thank my family for their support and understanding through this long journey. My wife, Christina, endured many late nights and weekends that enabled me to focus on making progress toward completion. Thank you to my kids, Marcus, Dominic, and Lauren, for their strength, hard work, and perseverance over the years.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Survival in Adversarial Environments

Autonomous systems research is a diverse field, spanning internal system architectures to collaborative path planning [1–16]. One common example of path planning is the grid coverage problem in which a system or team of systems must visit every sector of a particular area in as few steps as possible given some physical terrain constraints. Agents operate in such benign environments where system features and the environment itself are all that impact the agent's ability to achieve its objective. In contrast, adversarial environments contain agents whose goal is to prevent friendly agents from completing their task, or at least to reduce the performance by which they are evaluated (e.g. force the coverage of an area to take much longer than in benign environments). Out of necessity, more complex path planning algorithms have been designed to optimally avoid and mitigate the threats from adversarial agents while achieving the desired performance. Critical to the performance of any such approach is the agent's ability to *survive* threats from adversarial agents in the pursuit of accomplishing its task.

Both adversarial and benign environments are typically modeled with a physical representation, where the severity of a threat is, at least partially, a function of its location. What lacks in the literature is modeling threats from adversarial agents as a function of time. Necessarily, the metric by which performance is evaluated is also a function of time: the probability of surviving a given mission duration. Compared to threat distribution across physical space, they are instead distributed across time. In practical terms, a single adversarial agent threatens the friendly agent once. However, any number of agents each may pose a separate threat to the friendly agent over the course of a "mission", which represents the execution of a task over some finite amount of time. Therefore, an adversarial agent poses a threat to a friendly agent at a moment in time rather than at a physical place. The survivability of a system (the probability that it survives the duration of the mission) is therefore influenced, in part, from the cumulative series of threats during a mission.

Susceptibility represents the ability of an autonomous system to avoid a threat by an adversarial agent. In previous work, the presence of a threat or its relative location determines its realized negative effect on the friendly agent. However, the friendly agent can possess an internal Machine Learning (ML) model with some level of performance that is able to detect the

presence of a threat and potentially avoid or mitigate it. The ability of the system to avoid potential destruction by an adversarial agent increases with the quality of its model. The agent uses its model to make a real-time determination as to whether a threat is present at any instant in time. If a threat is determined to be present, an agent can take some action to potentially neutralize the threat, increasing its probability of survival. Humans therefore cannot be in the loop due to the real-time nature of this decision in adversarial environments. Survivability is strongly dependent on model performance in this concept of adversarial environments, and therefore it is critical that autonomous systems possess the ability to improve their models on fresh, relevant data to maximize such survivability during the mission.

This dissertation introduces, defines, and characterizes a novel ML paradigm named Survival-Critical Machine Learning (SCML) in which a model's performance defines, in part, the probability that an autonomous system accomplishes its mission in adversarial environments. This work is meant to lay the foundation for SCML system design, analysis, and evaluation in scenarios where threats evolve and morph dynamically with minimal training data available *a priori*. It identifies the primary system parameters and quantifies their impact on survivability. It also introduces derivations of existing and new approaches to online and continuous learning and defines features that maximize the effectiveness of SCML systems from a traditional ML and user-interface perspective.

1.2 The Need for Urgent Learning

The vast majority of recent ML research has assumed correctly that the model training and validation process has access to sufficiently large and representative datasets. To achieve such scale, these datasets have been carefully collected, curated, and annotated over some period of time. Most contemporary and advanced ML systems run in the cloud on robust compute infrastructure because they do not need to interact with the physical world; rather they receive input and can provide output over a web interface. Edge computing can perform a scaled-down capability for drones and ground robots, for example, to run models for navigation and object recognition in a physical environment. However, the core ML systems that run within them are designed, analyzed, and evaluated with traditional ML parameters and metrics where data collection and training are strongly decoupled. In classical ML, the time delay between collecting data from a sensor to extracting its value and embedding the resulting knowledge into a model is long. In other words, learning is not urgent. This leisurely approach may be optimal when such systems operate in simple, benign environments, but sub-optimal when operating in adversarial or hostile environments where threats morph and evolve continuously, and when learning *is* urgent. In the former setting, there is no penalty for a long time delay between data collection and new model deployment, but there is a severe penalty in the latter setting. The Key Performance Indicator (KPI) of ML systems when learning is urgent in hostile environments is the time delay between the capture of a data sample and embedding the knowledge derived from it in the model. This KPI is illustrated in Figure 1.1 below.

Some threats in hostile environments attempt to remain "undetected" so that they have a better chance of damaging or destroying the ML-protected system, preventing it from completing its mission. As a result, it is likely that threats will evolve (e.g. gain intelligence) or deploy new

Figure 1.1: Key Performance Indicator (KPI) when Learning is Urgent

camouflage techniques, for example. Wholly new threats could emerge, challenging the ability of a traditional ML system pre-trained on a (small or large) dataset with a fixed number of classes and samples to accurately detect threat instances. Thus, the capabilities to adapt to current novel threats and retrain ML models on fresh, real-time data are ideal for a "survivable" ML system. To accomplish this, the primary components of the ML process: data collection and curation, annotation, training, and inferencing, must be tightly coupled and run concurrently in order to learn from fresh data as quickly as possible.

The need to rapidly adapt to fresh threats exposes fundamental limitations of the traditional ML workflow. The strongly decoupled approach ensures offline training does not even start until engineers determine there is sufficient data to train a model with the potential to reach desired performance objectives. However, in hostile environments there is no such luxury as we expect to see novel (of an unknown class) or rare (of a known class for which we currently have few) samples. Their features must be trained into the model for future classification or detection *on the same mission* (i.e. online).

1.3 Parallels with Biological Immunity

The intuition behind SCML is motivated by the ability of biological immune systems to evolve in order to survive. Humans receive a robust set of immunizations early in their life. Such immunizations are developed by pharmaceutical companies as a result of infections, sicknesses, and even deaths of others in current and prior generations. Whether in an emergency scenario like the COVID-19 pandemic or in longer-term testing programs, vaccines are developed, tested, approved by regulatory agencies, mass-produced, and delivered to the population. This process is analogous to the "train in the cloud, deploy at the edge" approach in contemporary ML practices. Although distribution of vaccines during a pandemic is considered urgent to save lives in a short period of time, the mechanics by which it occurs accurately represents current ML practices. Previously learned knowledge of pathogen characteristics expected to be encountered by the body is embedded into the immune system to prevent future infection. The primary advantage to this behavior is that it could prevent infection all together or at least minimize the severity of symptoms and lethality of an infection, increasing the likelihood of survival. As with a model possessing perfect recall, a perfect vaccine would prevent infection from any of the target pathogens, guaranteeing 100% survivability, at least from the respective threats. Thus, this practice is ideal when data is abundant and demand for a vaccine is not urgent (learning is not urgent), yet it is impractical when a rare or novel pathogen is encountered and it is expected to take years and copious amounts of data, e.g. human infections, to produce a vaccine.

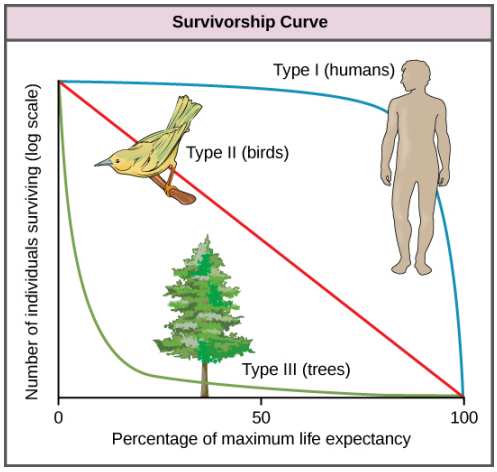


Figure 1.2: Survivorship Curves (Source: [17])

For reference, Figure 1.2 depicts generic survivability curves for humans, birds, and trees. The curves depict the number (or similarly percentage) of the original population still alive at a given percentage of the respective maximum life expectancy. Most humans survive childhood and middle age while most trees do not. Birds have a (relatively) uniform death distribution across the typical lifespan. The shape of the human survivability curve would look quite different from hundreds of years ago when the body's only defense against deadly diseases was the immune system's ability to learn during a single lifetime and leverage knowledge through inherited traits via evolution. Still today, we are often exposed to pathogens for which there is no vaccine. While it is possible as in the recent pandemic to develop a vaccine on the order of months or a small number of years, the same cannot be said in the era of pre-modern medicine. In any case where there is at least some period of time in which a vaccine has yet to be developed, the immune system is forced adapt to any infection to which it is exposed on its own. A form of online learning must take place where the immune system learns from potentially lethal exposure to various pathogens in order to maximize the probability of future survival. This is commonly known as "natural immunity", where the body is initially exposed to the pathogen without any (or very minimal) existing pre-learned defenses to it, and adapts such defenses to prevent potential future infection. When an appropriate vaccine is not available for a rare or novel pathogen, the biological equivalent of online learning is the only option to survive (the immediate threat), learn, adapt, and increase the likelihood of survival in the future. As we will show in this work, continuous online learning is critical to the optimization of SCML systems and their long-term survivability.

1.4 Learning/Survival Tradeoffs Under Resource Constraints

As described previously (§1.3), learning from encountered threats is critical when little to no pre-existing knowledge is available. Autonomous systems are expected to have access to at least a weak model at the beginning of a mission. Additional threat exposure is required for the system to learn, thus improving its ability to discern between threat and non-threat. In the

SCML context, this is a unique instance of the chicken and egg problem. Surviving repeated threat exposure is required to learn, and the result of learning (in the form of an improved model) is the increased likelihood of surviving future threats. We describe in this section the tradeoffs between survivability and learning under various degrees of resource constraint across diverse deployment scenarios, which are further analyzed in later chapters.

An SCML system may be exposed to a dense environment of potentially lethal threats early in the mission. As a result, it may be destroyed before it has an opportunity to improve its model. Therefore, it may be prudent to be more generous with its use of *countermeasures* (their impact more practically defined in §3.1.1 and §4.1) during this time interval in the mission. Countermeasures in this context represent a set of generic finite resources that are deployed to neutralize potential threats. In order to survive an early onslaught of threats, countermeasures can be deployed liberally in order to allow for the system to learn. The protection they provide gives the system a much better chance of survival early in the mission. As a result of encountering, and surviving, a greater threat density early in the mission (generating more potential training samples), the system is able to learn more rapidly. The drawback of this approach is that, due to the depletion of countermeasures earlier than expected, the system may be more vulnerable later in the mission. The intentional excessive deployment of countermeasures early in the mission greatly increases the probability of survival up to a certain point, but can leave the system more vulnerable to threats later in the mission if used to the extreme.

If a similar dense threat environment were to be encountered by the system but the employment of countermeasures was somewhat restricted, the likelihood of surviving this time period would be greatly reduced. However, if the system were to survive this dense threat time interval, an increase in long-term survivability would occur as it is more likely that sufficient countermeasures are available for the rest of the mission. In this case, learning would occur rapidly from the dense threat environment, but it is only useful insofar as the system survives long enough to leverage such learning. On the other hand, in a sparse threat environment early survival is easier to achieve with minimal countermeasure deployment, yet learning is slowed significantly. Although overall survivability is increased with reduced threat density, models will likely be weaker due to lower quality or less frequent learning. Even if countermeasure deployment were front-loaded to maximize survivability in the early part of the mission, learning is still limited by the threat density.

Regardless of the mission parameters outside of the control of an SCML system (threat arrival rate, lethality, etc.), strategic resource usage across the duration of the mission can both directly and indirectly (via its impact on learning) impact survivability. Other primary finite resources besides countermeasures in an online learning-capable SCML system are: backhaul network transmission rate from SCML system to a centralized location (if required for generating labels for new samples), associated domain expert labeling rate, the number of SCML systems inferencing potential threat samples, and system compute and storage capacity. The labeling rate corresponds to the number of labels the expert can produce per unit time. Backhaul network bandwidth and labeling rate can have similar effects on the ability of an SCML system to learn: they both control how many samples are labeled through during the mission. Although the samples waiting to be labeled are queued at different locations (SCML system vs. centralized location with domain expert), the system-level effect is the same. The compute and storage capacity on each system limits how many samples per unit time can be inferenced and prioritized

for transmission. Therefore if the goal is to accelerate learning by inferencing, prioritizing, and transmitting more samples to the domain expert, the compute and storage limits on the SCML systems themselves are the limiting factors. These various resource constraints motivate the need to analyze their impact on such an SCML system capable of learning in real-time. In this dissertation, we analyze and evaluate an extensive set of combinations of these constraints on survivability.

1.5 Dynamic System Adaptability for Enhanced Survival

We assume that adversarial environments contain agents with goals of damaging friendly SCML agents while attempting to remain undetected. Whether this agent is an intelligent actor or a natural phenomenon, the analysis remains the same. SCML systems capable of learning can adapt to what it observes in real-time as the internal model is retrained on fresh data collected from the environment. Thus if the domain expert notices additional objects of the threat class that have slightly different features than the original samples, the model will adapt such that future samples containing these modified features will also be prioritized for transmission to the expert for labeling. Naturally, this can cause model bias as the mission progresses where the model prioritizes later samples which are similar to those found earlier in the mission.

Not only must SCML systems adapt to a single threat class that may morph and evolve during the mission, they must be able to detect a separate, sufficiently unique threat class unknown prior to the mission. This is called novel class discovery and is similar to recent work on Open World Recognition, although it has been largely applied in the standard ML context, e.g. offline training where all data is available. The ability to reliably detect a new threat class for which the system has no previous knowledge (training data) is paramount in an unpredictable and adversarial environment. This online adaptability must be a core characteristic of SCML systems that can adjust to real-time changes in the environment, e.g. a completely unique threat class suddenly appears and must be detected accurately in the future. Survivability is clearly degraded if a class of threat objects are present in the environment unbeknownst to all SCML systems that could destroy them. Therefore, core functions of a learning-capable SCML system must be modified during the mission to accommodate this capability.

Similar to novel class discovery, systems must also be able to divide an existing threat class into two or more separate classes. If the original threat class is too broad for a unique mission, e.g. vehicle, but it is more operationally appropriate to separately identify a car from a truck, then the original vehicle class should be split. Of course, this approach could be taken recursively, as each child class can further be subdivided into its own child classes. As with novel class discovery, this allows the system to adapt in real time to operational requirements to maximize survivability. In a certain situation, it may be appropriate to treat cars as low threat but trucks as high threat, enabling the survivability model to account for threats of either class with differential abilities to destroy the SCML system. Similarly, it could also enable more optimal CM deployment. Trucks may require two CMs to be neutralized while cars only need one. Trucks and cars may also require different types of CMs, e.g. Type I for trucks and Type II for cars, effectively preventing the total number of CMs from being shared across threat classes. Forcing the model to classify at a greater degree of granularity can also improve classification performance. Labeling only cars

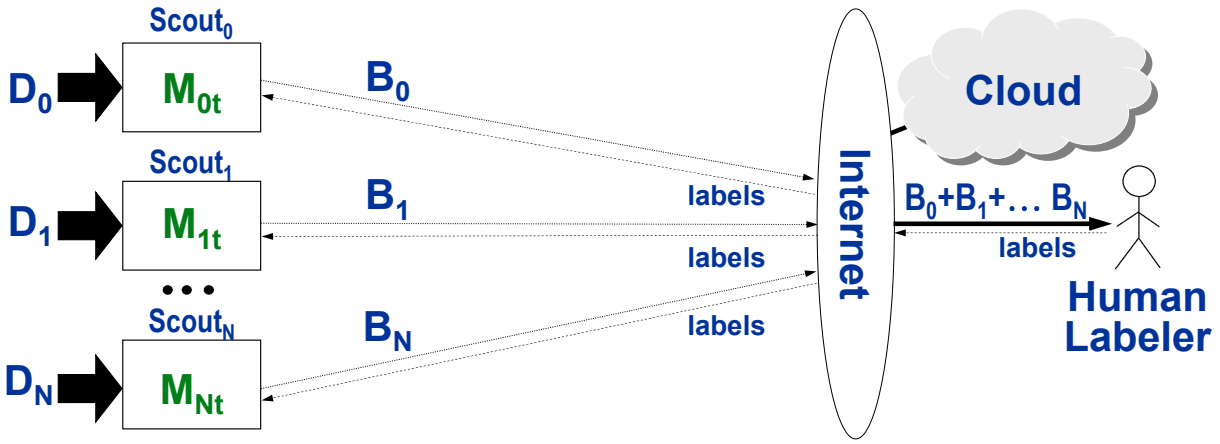
as "cars" and trucks as "trucks" instead of both cars and trucks as "vehicle" will produce more accurate results as each set of class samples will have less intra-class difference between them.

System-level parameters must be adaptable in adversarial environments, as with the labeling process and models themselves. The countermeasure deployment threshold, described throughout §4, has a significant impact on survivability. The threshold must be adaptable in order to respond to current mission conditions. It indirectly controls the rate at which countermeasures are deployed, which affects the rate at which potentially lethal threats are encountered and when countermeasures are exhausted, ultimately influencing overall survivability. The system must be able to evaluate its current state, such as its remaining resource reserves, and take the appropriate action to ensure it has the best chance of surviving the mission duration. This work is detailed later in §4.10.

The motivation for these system adaptations in real-time is the assumption that adversarial agents are threats that can evolve and morph during a mission. Therefore, we must define, explore, and design into SCML systems the ability to counter these anticipated capabilities of such agents. We of course cannot exhaustively design into SCML systems every potential capability that may be necessary to maximize survival in the wild, but we show with this initial subset of adaptations that it is possible to minimize vulnerabilities of SCML systems that can be exploited by adversarial agents. A rigid architecture that cannot adapt to changing conditions is surely more vulnerable than one that can change as needed. SCML systems are designed to react to adversarial agent behavior: if an agent changes its appearance, it may cause the system to recognize a potential novel class, requiring model adaptation with the goal of improved future detection and neutralization. The ultimate goal is to ensure that SCML systems maximize survivability by adapting to and countering any behavior change that an adversarial agent attempts.

1.6 Live Learning for Adversarial Environments

Live Learning, first introduced by George et al. [18] and illustrated in Figure 1.3, is a human-in-the-loop system that performs the tight coupling between ML processes required to adapt to dynamically morphing threats. It provides a unique capability: the construction of a training dataset of a rare class from very few initial samples. A distributed team of scouts sense streaming data samples from the environment, inference them with local ML models, and prioritize them for transmission to the human labeler. Once the domain expert has confirmed a label, it is transmitted back to the scouts to retrain future models. As a result, the models improve their ability to prioritize future incoming samples for labeling. Over time, the system constructs this training set of a rare target class. Live Learning's initial design was based on the intersection of three primary challenges: 1) Severe mismatch between incoming sensor data rate and network backhaul bandwidth from the scout, 2) Expected extremely low base rate ($\sim$0.1%) of positive samples in the incoming stream, and 3) The target class is both new and rare for which we have few samples to train a weak initial model. While Live Learning was originally designed to construct such datasets, it is also able to adapt to threats in real-time, making it ideally suited for adversarial settings. Further, it allows a domain expert to steer the system toward a user-defined objective. The two primary objectives of any system required to operate in adversarial settings where survival is paramount are to: 1) maximize the recall of any samples of the rare target class



Labeling rate $B_i << D_i$

D_i = data rate into scout$_i$ M_{it} = model at scout$_i$ at time t

Figure 1.3: Conceptual View of Live Learning (Source: [18])

(threats) in the environment, and 2) use any finite resources as efficiently as possible to facilitate the former. SCML is thus the optimal application to leverage the capabilities that Live learning provides.

1.7 Thesis Statement

For improved survival of edge-based systems in environments where threats evolve and morph, it is feasible and effective to implement a new model of continuous learning called Survival-Critical Machine Learning (SCML). SCML can be implemented with Live Learning. SCML encompasses a rich tradeoff space. It can be applied to well-known ML paradigms such as Markov Decision Processes (MDPs). SCML can work with a wide range of sensor modalities.

Why is this thesis important? If this statement were true, then future ML systems that operate in hostile environments with continuously evolving threats can be optimally designed and operated such that their survivability is improved. Moreover, it would lay the foundation for a new branch of ML research around the design considerations of survivable ML systems. Finally, it would demonstrate how Live Learning is a key and necessary feature for the SCML paradigm, where little or no existing training data is available to the system.

Why does it not follow trivially from what is already known?
SCML is a new ML paradigm and therefore not an extension or logical next step from any existing framework. Although federated learning enables training at the edge, it assumes data is already labeled. In SCML there is no labeled data available at each training node. The state of practice in ML today is the strong decoupling between data collection, labeling, training, and inferencing. Training and inferencing are separated in that a model will not be served to

customers in production until engineers have validated the achievement of some threshold levels of traditional performance metrics. The time delay between the beginning of data collection to inferencing in production may be months or even years. On the contrary, SCML systems may need to operate in hostile environments on a much shorter time horizon, on the order of minutes or hours (see KPI in Figure 1.1). Systems must operate with the data processed during operation or that which has already been trained into a base model. Therefore, SCML systems will likely have a priori access to a *minimal number of samples of threat objects expected in the hostile environment*.

The strong coupling between ML processes inherent in Live Learning renders SCML a further departure from existing practices by performing all tasks concurrently. There currently exists a wide array of quantitative metrics such as precision, recall, accuracy, and area under the precision-recall curve (AUC), among others, to evaluate the performance of ML systems based on their respective tasks, such as classification or detection. SCML derives new metrics partially from traditional ML metrics as well as from various SCML system-level parameters. The result is that SCML system design tradeoffs are impacted by both expected operational conditions and technical model capabilities, significantly departing from the current standard practices of solely targeting metrics like average precision or AUC.

1.8 Thesis Validation

In general, the objective in survivability analysis is to determine the amount of time until some negative event occurs, typically death. ML techniques have recently been applied to survivability analysis to better approximate such estimates, but it has yet to be applied to ML systems themselves. Our goal is to make survivability an integral part of the design of an ML system and its operation. ML system characteristics and traditional properties of survivability models must be carefully integrated. The resulting SCML system maximally leverages the underlying power of the ML models it runs and optimizes the surrounding system-level tradeoffs to improve survivability. Similar to historical applications, SCML systems are evaluated by the probability of survival as a function of time.

Live Learning was originally created for reasons that have nothing to do with survivability. It was created for self-improving transmission of training data under conditions of extreme low bandwidth and extreme class imbalance. An important contribution of this work is to demonstrate that Live Learning can improve survivability. Building on the existing capabilities of Live Learning includes designing and implementing new features and evaluating their effects on the broader SCML model. The intent is to modify Live Learning such that the core capabilities it provides optimizes SCML performance. The primary strength of Live Learning is that it can improve its ability to detect threats in an online setting, which is critical for SCML when threats morph and evolve over time. Design tradeoffs therefore exist between Live Learning parameters, SCML system parameters, and operational conditions that must be balanced appropriately. A key consideration is how to optimally manage the model retraining process based on current SCML system status. The training frequency on a tunable number of samples and other parameters directly affect survivability. If the system retrains too late, it may be destroyed before retraining

a superior detection model. If the system retrains too early, the new model may not sufficiently improve upon the previous, also potentially reducing survivability. The goal is to design an SCML system that survives long enough to continuously improve its ability to accurately detect and counter threats. The final result is that the SCML system will have survived the required duration to complete its specific task in a hostile environment. Integrating Live Learning within an SCML system is a critical part of validating this thesis.

For experimental validation, I will use open source datasets that contain classes of common, but benign objects in the environment. The classes of objects defined as threats in any particular case may not be the exact class an SCML system is likely to encounter in a real-world hostile environment. Examples might be an adversarial drone, tank, etc. However, the classes contained in these datasets have representative features of anticipated threat classes, such as: airplane, ship, helicopter, large vehicle, etc. (all classes of the DOTA dataset [19]). Therefore, we use these available classes as surrogate threat classes for experiments. In SCML, encountering a threat is expected to be rare. As a result, SCML systems operate in low base rate conditions, where the number of threat objects expected across some period of time is very low compared to the total number of processed samples. Datasets from other domains separate from the traditional visual spectrum are also considered, such as radar. This will demonstrate that SCML is applicable to systems with diverse sensor modalities, which may be required according to the respective operational context.

Resource allocation tradeoffs are also analyzed. In SCML, finite resources are employed to enhance survivability, however, such resources can be exhausted. Design tradeoffs will be examined against ML model training and inference properties. The optimal combination of traditional ML system parameters and more system-oriented properties is the most critical task to designing a survivable SCML system. In the SCML paradigm, where compute, memory, energy, and time are finite resources, real-time decisions must be made that take these into account.

Hawk is a real distributed ML system that performs Live Learning [20]. We validate this thesis by adding SCML functions to Hawk to evaluate the survivability of a distributed team of SCML systems. We also add various capabilities to evaluate unique Live Learning-based and SCML-focused features and design parameters. We defined the KPI in §1.2 as the time delay between data collection and embedding knowledge into the model. We translate this into the derived KPI of survivability as a function of time in $[0, 1]$. The main contributions of this dissertation are as follows:

1. We define SCML in an analytical framework to provide mathematical intuition as to how such Live Learning-based systems perform. This is crucial as the models, and thus their performance, changes during the mission, which affects survivability.
2. We develop an experimental model for SCML and evaluate it on real datasets from both the visual and radar domains to demonstrate extensibility across multiple sensor modalities.
3. We explore design tradeoffs in complex SCML scenarios to assess costs, benefits, and the primary optimization variables of different approaches. Complex survival criterion, adaptive thresholding, staggered scout deployment, novel class discovery, and imperfect performance metric alignment between Live Learning and SCML are also explored.
4. We model and evaluate SCML as a Markov Decision Process (MDP) to adapt it to an existing core ML paradigm (Reinforcement Learning).

1.9 Thesis Roadmap

The remaining chapters in this thesis validate the thesis statement at the beginning of §1.7. Each chapter in whole or in part contributes to the validations of the three main axes of the thesis statement. The remainder of this thesis is organized as follows:

- Chapter 2 identifies recent and related work with respect to historical applications of survivability analysis and Live Learning. It also describes more recent applications like network survivability along with critical enabling features of SCML such as edge computing.
- Chapter 3 introduces the SCML analytical model, combining Live Learning with SCML mathematically to describe SCML system survivability as a function of time.
- Chapter 4 provides the experimental model and SCML performance results, which document the tradeoffs between various SCML system parameters. It also describes tradeoffs between precision and recall in Live Learning-based SCML systems. Additionally, it explores more complex survival criterion, the impact in imperfectly aligned performance metrics between Live Learning and SCML, adaptive thresholding, staggered scout deployment, and novel class discovery.
- Chapter 5 extends Live Learning to short range radar applications. It covers the unique characteristics of radar data and demonstrates the extensibility of Live Learning to modalities other than visual.
- Chapter 6 formulates the SCML model into a Markov Decision Process (MDP) to adapt it to an existing ML framework comprised of agents, states, actions, and rewards. This approach formalizes SCML into a state-action-state process whereby a new threshold is selected after each inferenced sample.

Chapter 2

Background and Related Work

SCML is a canonical use case for Live Learning (§2.2) in a world where autonomous systems, e.g. drones, are relied upon to complete critical tasks in adversarial environments. In benign environments, survivability can be reduced to the endurance of the system (energy capacity). However, in adversarial or hostile environments, the system's ability to accurately detect and neutralize threats under finite resource constraints also dictates survivability. SCML is a new ML paradigm and therefore survivability analysis for autonomous systems is virtually non-existent. Edge computing (§2.1) enables Live Learning and its core functions (§2.2), which are critical to an SCML system's ability to improve its relative survivability over time. General survivability analysis has been applied to a wide range of application domains (§2.3), much of it applied to areas like life expectancy and other medical research. With a robust literature in robotics and autonomous agents, we also document such work related to adversarial environments (§2.4).

2.1 Edge Computing

One of the foundational components of SCML is edge computing. Adversarial environments (at least in the physical world) are by nature at the "edge", which is the physical overlap in which friendly and adversarial agents interact. Thus, edge computing powers the intelligence required in SCML systems operating at the edge in the presence of adversarial agents. Edge computing is partially defined relative to cloud computing, as shown in Figure 2.1. What we think of as the "cloud" today is depicted as Tier-1 in the figure. Tier-2 serves as an edge computing presence: it can have significant computing capability while maintaining very low latency to endpoint applications and devices. Tier-3, e.g. small drones, security cameras, are quite restricted in size, weight, and power and thus reside at the extreme edge where humans or adversarial agents typically interact with them directly.

Recent work has addressed the need for edge computing in adversarial or hostile environments, recognizing that the unique characteristics of such environments: dynamic, complex, contested, etc. render the cloud much less dominant in applicability. The role of "tactical cloudlets" has gained traction for environments under degraded network conditions or where complete disconnections and other cyber-physical threats are possible [22]. Other similar use cases for edge computing via tactical cloudlets include the requirement to support continuously mobile Tier-3

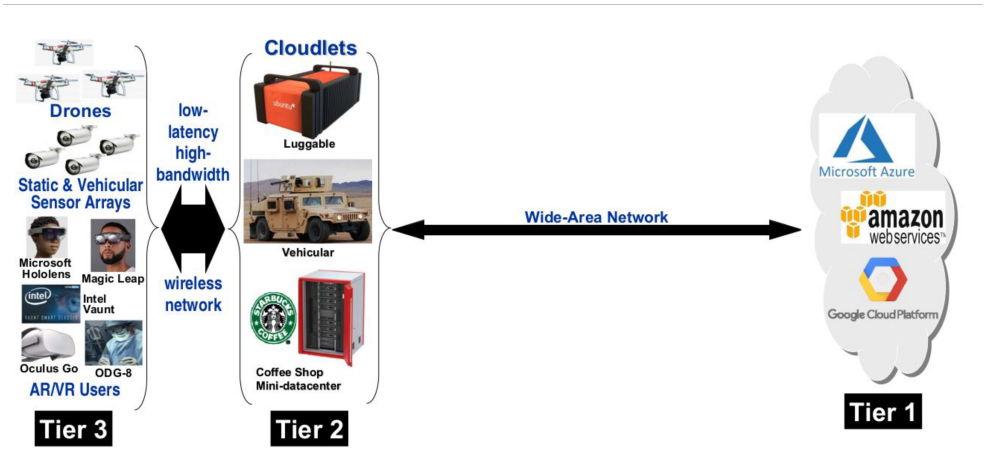


Figure 2.1: Three-Tiered Model of Computing (Source: [21])

devices [23]. Tier-2 cloudlets may even be under threat themselves and must be intelligently located to maximize computing survivability. The broad emergence of edge computing and a comprehensive summary of its benefits and likely future were recognized in [24]. Much of the initial focus was on providing robust computing at greatly reduced latency to end point human-centered applications and devices, driven by an expectation of explosive growth of interactive applications and IoT devices of all types. Partially analogous to tactical environments, industrial settings are another core application domain for edge computing. It has been specifically targeted to enhance resilience and survivability of cyber-physical systems that house Industrial Control Systems (ICSs) [25, 26], many of which run critical infrastructure and commercial manufacturing.

Finally, more recent studies have explored how edge computing is a critical enabler for tactical IoT architectures [27] and for distributed, human-in-the-loop, AI-enabled operations [28]. Such operations can be mobile and dynamic in which edge computing enhances tactical agility on the increasingly technologically complex battlefield.

2.2 Live Learning and Hawk

As originally conceived, Live Learning addresses the problem of severe bandwidth mismatch between incoming sensor data rates at scouts and wireless backhaul bandwidth. This mismatch is overcome via a self-improving ML-based transmission system embodied in an open source implementation called *Hawk* [20]. Our repurposing of Live Learning for SCML requires changes to Hawk. Specifically, low bandwidth is no longer a motivation — Live Learning is valuable at any backhaul network bandwidth. We describe the original implementation of Hawk below, and describe our modifications later in the document (§4.1).

Starting from a weak model that is trained via few-shot learning (FSL) on just a few initial examples, Hawk seamlessly pipelines semi-supervised learning, active learning, and transfer learning, with asynchronous bandwidth-sensitive data transmission to a distant human for labeling. When a significant number of true positives (TPs) have been labeled, Hawk trains an improved model to replace the old model. This iterative workflow is Live Learning, which continues until

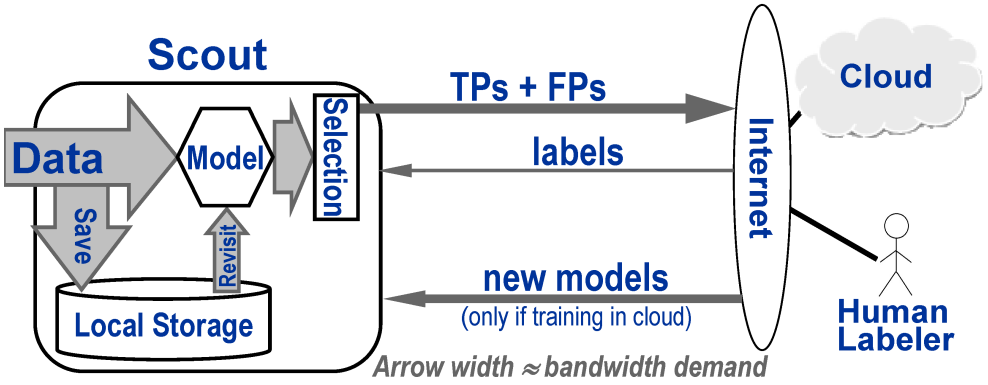


Figure 2.2: End-to-End Pipeline of Hawk (Source: [18])

a sufficient number of TPs have been collected. The workflow is independent of the specific deep neural network (DNN) architecture that is used for learning. The scout abstraction in Hawk combines Tier-3 & Tier-2 computing. An actual implementation may involve a monolithic Tier-3/Tier-2 (e.g. a spacecraft with substantial on-board compute) or a disaggregated Tier-3/Tier-2 (e.g. a Tier-3 drone with a ground-based Tier-2 cloudlet connected via a tactical 5G wireless network).

Figure 2.2 illustrates the Hawk processing pipeline. Fresh data arrives continuously at a high rate into a scout. Only a tiny fraction of this incoming data can be transmitted for labeling. Incoming data is both written to local storage, and processed. The processing consists of three steps. The video stream is first decoded into individual frames. Each frame is broken up into small tiles, and the tile stream is inferenced using the current model. After inferencing, a small subset of the tiles is selected for transmission and labeling. A tunable data sourcing policy guides processing new incoming data versus re-processing previously discarded data. Transmitted items are queued in the cloud for labeling by a human expert. This data is also available for training in the cloud. Labels are transmitted back to the scout as they are generated. Hawk can choose between training in the cloud or on the scout. If training in the cloud is chosen, the new model is transmitted to the scout after training. With multiple scouts, precious TPs are shared across the team so that models improve at their cumulative TP discovery rate. For training on scouts, negatives are obtained locally. For cloud training, negatives are obtained from archives.

2.3 Pre-ML Work on Survivability

For centuries before ML emerged, the concept of survivability was developed in diverse contexts. The first known use of this concept was in the creation of a mortality or life table in the late 17th century [29]. Modern actuarial analysis for life insurance is a direct descendant of that early work [30–33]. In the 20th century, medical epidemiology has proved to be a major driver of survivability analysis. Examples of work relating to survivability in this domain include [34–41].

For technologically complex systems that must endure harsh and unforgiving environments, survivability analysis has been an integral part of the design process. Examples of such systems include spacecraft [42–44] and military systems [45–47]. More recently, survivability analy-

sis has been used in digital and pervasive system design and maintenance [48–50]. Most recently [51, 52], novel ML methods have been used to improve survivability analysis in existing domains. In contrast, this work aims to use ML as a functional part of an operational system. Live Learning is thus integral to such a system, not a tool for analyzing its survivability.

The most recent predecessor research thread to autonomous and ML system survivability is that of network survivability. Prior to the recent surge in AI/ML-focused research, overall network and Internet research dominated as the Internet and all associated technologies have driven economic growth and added value to our lives across many domains. The more specific research thread of network survivability began in the pre-ARPANET days of the cold war with concerns over the destruction of critical inter-site links by nuclear attack [53]. Ubiquitous, high-speed, reliable Internet access by definition requires a high level of survivability [54]. Additional research has evolved from redundancy of core point to point backbone links to that for Elastic Optical Networks (EONs) [55], mobile, ad-hoc, and space-based networks [56]. Survivability of mobile, wireless, ad-hoc networks have been of additional interest given their complex characteristics that cause instability [57, 58]. Other studies on generic information systems have described core survivability characteristics and evaluation measures assessing the likelihood that the system can survive both major and minor failures [59, 60]. Further, in military settings, survivable IT systems were shown to be critical for mission success [61].

2.4 Autonomous Agents in Adversarial Environments

Current literature relating to autonomous systems largely revolves around navigation within diverse environments to achieve some goal. Often the goal is to reach a destination as quickly as possible or to use one or more agents to maximize the coverage of an area in the shortest amount of time. While some work addresses survivability, none has analyzed it in terms of survival as a function of time partially based on the performance of the internal ML model. Relative to the sophistication of today's robotic systems, those of the 1980s and 90s can be considered primitive and were largely confined to activity in static, sanitized environments. In the early 90s, Arkin first observed the need for robotic systems to survive in hazardous and highly dynamic environments using both reactive control (to respond to its own perception of the environment) and homeostatic control (the state of its internal resources) [62, 63]. This work enumerated the core elements robotic systems use to remain survivable in such conditions. Survivability frameworks have also been developed with a biological context. Needs and emotions comprise the state representation where the needs represent the gaps between the current and desired system states and survivability is defined as the ability of a system to fulfil its needs [64, 65]. Other similar frameworks, such as the Adaptive Adversarial System of Systems (AASS) has gained attention [66], which defines design principles such as perception, communications, and security, and reliability architectures for teams of autonomous systems in adversarial settings.

In this work, an environment is adversarial in that a physical (kinetic) threat appears to the SCML system and is sensed and classified as such by an image classification model. A threat could also present itself in the cyber domain. As discussed earlier, network survivability research has been a small piece of the large body of general network and Internet research the past few decades. However, survivability studies in the cyber domain explicitly within adversarial envi-

ronments, while critical, is much more narrow in focus [67]. Others have documented ML-based optimization of critical infrastructure like roads, sewer, and other utilities to secure them against diverse threats [68]. Another study defined survivability as a layered model where the lowest 1) is to stay alive, 2) the ability to communicate, and 3) maintaining the availability of a service [69]. The survival time is the "duration of life span, communication, or service availability before death or a malfunction". Although survivability in much of the referenced literature is treated as a single event: a device fails or it doesn't fail, for example, this is one of the only studies that more explicitly treated survivability as a function of time.

The aforementioned coverage problem in robotics becomes more difficult with the presence of adversarial agents attempting to spot the friendly agent. The adversary thus influences the set of feasible paths to the target, increasing the amount of time required to complete the task [70]. Various methods are involved to solve these types of problems, including providing the static threat map offline prior to execution as well as performing the spanning-tree and greedy approaches [71]. Additional work has investigated the impact of strategic (intelligent) and non-strategic (unintelligent) adversarial agents' impact on survivability in various application domains such as search and rescue, intelligence, and multi-robot patrolling [72]. With clear military applications, survivability and resilience of autonomous vehicle system of systems is paramount. DoD resilience is defined as broad utility, the ability to repel, resist, or absorb, the ability to adapt, and the ability to recover [73]. Safety is also critical in military settings as unsafe systems can be an internal threat to survivability. The system's speed, maneuverability, situational awareness, and countermeasures are all local capabilities that ultimately impact survivability, even if not an attribute of an adversarial threat [74]. Markov Decision Processes (MDPs) have been recently employed to optimize the frequency of target observations in tasks similar to surveillance counter-surveillance tasks where the goal is to minimize exposure to ballistic threats [75]. Further, tactical navigation via road networks have been optimized as a function of threat location and target distance (somewhat similar to the coverage problem) using Q-Learning [76].

As stated, most research on autonomous system development largely focuses on perception and understanding of the environment and taking an action, such as path planning, recognizing objects, and navigating around obstacles toward a destination. Further, typical problem definitions are modeled on some sort of 2D X-Y grid representing physical space. In frameworks with adversarial agents, they attempt to spot the friendly agent or block its path in some manner. We take an entirely different approach in that we take into account the performance of the internal ML model as it directly impacts the ability to detect the presence of a threat, and thus its survivability. Moreover, with Live Learning we improve our model as a function of time as the SCML system is exposed continuously to threats during the mission. We can then model survivability as function of time derived from a broader set of factors, including model performance, appropriate finite resource deployment, threshold settings, etc.

Chapter 3

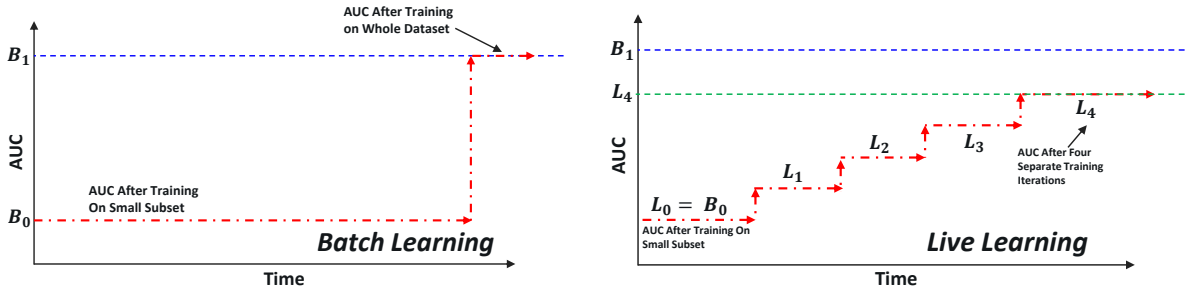
SCML Analytical model

Since SCML is a new concept, we need to gain an understanding of its tradeoff space. Towards this goal, we develop an analytical model of SCML. Our intent is to capture the highlights at an intuitive level, rather than to rigorously model minutiae. To do so we derive a pure mathematical model that represents survivability as a function of time. The model integrates traditional ML metrics with mission-specific and threat class-specific parameters. In a later section (§4), we ground this model in the context of an implementation and obtain experimental results that reflect the constraints of a real system. For simplicity, we focus on the ML problem of classification. In practice, an SCML system also needs to localize the objects that it is classifying — i.e. the underlying problem is object detection on live data, not just classification. Since that further level of complexity adds no new insights to core SCML functions, we focus on classification. For every incoming data item, a scout asks: *Is there a threat at this moment?*

3.1 Precision & Recall Alone are Incomplete

Historically, time has played no role in characterizing the accuracy of a classifier. The well-known metrics of precision and recall are all that matter. The time needed to create and use a model (i.e., data collection, labeling, training, and inferencing) is not relevant. These are viewed as secondary metrics, typically outside scope in discussions of accuracy. In SCML, time can no longer be relegated to a secondary role. A single classification error can lead to an undetected threat destroying the scout. There will be no further opportunity for that model to redeem itself through future successes. This is different from classic ML systems in which a classification error is not fatal — often, it merely lowers profit. In SCML, an early mistake means that the system dies young. Early survival, on the other hand, leads to later mistakes being less likely because of model improvement. Thus, time has to be viewed on an equal footing with precision and recall in SCML.

The role of time is captured by Figure 3.1, which contrasts Batch Learning (classic ML) and Live Learning. The Y axis is the area under the precision-recall curve (AUC). In Batch Learning (Figure 3.1(a)), training is deferred until ample data has been collected and labeled. Missions are not even attempted until an accurate model exists. Unfortunately, SCML has to cope with a world in which threats are not static, but evolve. Critical missions may have to be attempted



(a) Batch Learning (classic ML)

(b) Live Learning (see Figure 4.7 for concrete examples)

These two graphs compare classifier AUC as a function of time between Batch Learning and Live Learning. In Batch Learning, the AUC is not improved for the current mission during data collection, labeling, training set construction, and training. The fruits of this work are available only to future missions. In contrast, Live Learning improves AUC during the current mission. Although it may be unable to achieve the same optimal model as Batch Learning (because of the constraints of online rather than offline learning), it improves survivability for the current mission. This improvement is correlated with the increase in area under the AUC curve (i.e., integral of AUC over mission duration).

Figure 3.1: AUC vs. Time for Batch Learning and Live Learning

even without a high-quality model for a new threat class. For that class, a weak model that is based on a small training set may be all that is available at the start of a mission. During the mission, Live Learning can be used on incoming data to improve the model. More threats will be encountered on longer missions. Hence, there is greater danger that a threat will get through undetected and destroy the scout. However, if the scout survives the early part of a long mission, Live Learning has significant opportunity to improve the model. There is thus a delicate balance between survival and learning that involves time at its heart, as shown by Figure 3.1(b).

The key advantage of Batch Learning is that all data is available up front for analysis to create an optimal training set. This can yield the best model achievable from that training data for a given model architecture. Unfortunately, this optimality is of little value in an SCML setting — if the mission is attempted, the scout is unlikely to survive long enough to benefit.

Like the proverbial tortoise relative to a hare, Live Learning settles for much smaller incremental gains. However, these gains are available during the current mission. The gains are suboptimal relative to Batch Learning because they have to be made in an online setting. As Figure 3.1(b) illustrates, the result is a sub-optimal final AUC ($L_4 < B_1$). However, survivability is improved (§3.1.4).

3.1.1 The Mission Abstraction

Without loss of generality, we define a mission to be the successful delivery of some valuable payload by a scout from a source to a destination. The mission involves the risk of destruction of the scout by threats during its journey. For example, the payload may be life-saving medical supplies, food, or essential spare parts; the threats may be bullets, missiles, etc. The scout is

protected by an SCML system that continuously tries to detect incoming threats. Once a threat is detected, it can be neutralized with certainty by some *countermeasure (CM)*. Exactly how this happens is not relevant to SCML, and is therefore outside the scope of this dissertation. In practice, of course, the neutralization may not always be successful, and there may only be a limited number of CMs available on the scout. We also assume that a single scout is assigned to the mission. In practice, a convoy or swarm of scouts may be assigned. We ignore these complications initially, in order to simplify the analytical model, and thereby gain an intuitive understanding of SCML tradeoffs. We include these aspects in the experimental evaluation presented later (§4.3 and §4.6).

3.1.2 Characterizing Survivability

Survival is inherently probabilistic in nature. On an unlucky day, even a superb model may fail to detect just one threat out of many that it correctly detects. That unfortunate single mistake may result in scout destruction. Conversely, the scout may have a lucky day. A weak model may wrongly classify every threat, yet none of these undetected threats destroy the scout, and it survives unscathed. Both extremes are unlikely, yet either can happen on a real mission. A Bayesian prior is the density of threats during a mission. Regardless of model accuracy, higher threat density is inherently riskier. It may vary during a mission, being correlated with space (e.g., physical location) and time (e.g., greater chance of being discovered on a longer mission). For simplicity, our modeling assumes a constant threat density during a mission.

In epidemiology, the metric of *Life Expectancy* characterizes survivability. This is the mean of the probability distribution of the time interval from birth to death of a large population of organisms. We adopt this metric for SCML. Mission start corresponds to birth. Mission end may occur because of successful completion, or because of scout destruction. Too small a life expectancy leads to many mission failures.

3.1.3 Key Parameters and Relationships

Figure 3.2 summarizes the key parameters of survivability α, β, and γ, which are explained below. We assume that the stochastic process generating threats is memoryless. The threat interarrival time is then exponentially distributed. In other words, the probability distribution of the interarrival time, t, of threats is given by

$$p(t) = \alpha e^{-\alpha t} \tag{3.1}$$

where $1/\alpha$ is the mean of the distribution. More complex interarrival time distributions can be considered, but this simple model is adequate for gaining intuition about SCML.

From an ML and Live Learning viewpoint, a key consideration is *class imbalance.* In other words, how rare are threats relative to all other data classes that are seen during a mission? Extremely rare threats are good news from the viewpoint of survivability. However, they also imply slow learning because few TPs can be collected for a training set. Frequent exposure to threats (and survival by good luck) leads to much faster learning and model improvement. This

α	threat arrival rate
β	recall of SCML system
γ	lethality of threats

Figure 3.2: Parameters Relevant to Survivability

subtle relationship between risk and learning is expressed by:

$$\alpha = \frac{\text{total data samples}}{\text{mission duration}} \times \text{threat class base rate} \tag{3.2}$$

Small values of threat class base rate correspond to extreme class imbalance. For a given mission duration, this leads to low α even with a large number of data samples.

Since an SCML system performs classification (§3), its ability to detect and neutralize a potential threat is given by the classic ML measure of recall, symbolized as β

$$\beta = \frac{\text{TP}}{\text{TP + FN}} \tag{3.3}$$

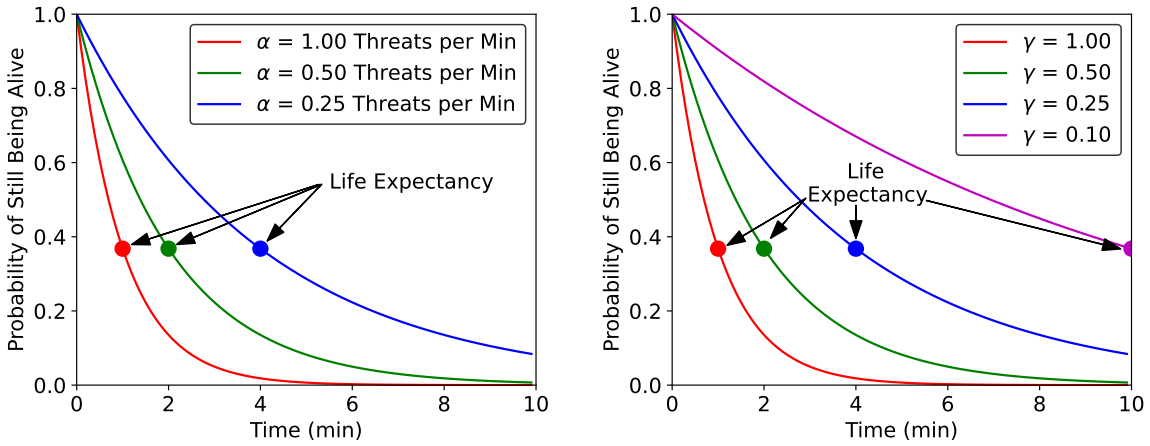
where FN correponds to false negatives (i.e., positives that are wrongly classified as negatives). Every FN is an undetected threat that may destroy the scout. Values of β close to 1.0 imply an SCML system that is effective in detecting threats. If threat neutralization is imperfect, an additional term to account for this will be needed in the analysis. As discussed earlier (§3.1.1), the classic ML metric of precision is important from the viewpoint of conserving CMs. Each false positive (FP) results in a CM being wasted. Achieving high recall at poor precision is suboptimal because it may result in a finite supply of CMs being exhausted long before mission completion. The scout is then vulnerable to all further threats encountered.

In practice, not every undetected threat is fatal. In the context of an immune system, for example, a pathogen that is undetected may still not successfully infect an organism or kill it. We use the term *lethality*, symbolized as γ, for the probability that an undetected threat leads to scout destruction. For "dumb" threats such as bullets, γ may only be a few percent. With precision guided munitions, it may be much higher and approach 100%.

Based on the parameters in Figure 3.2, the cumulative distribution function (CDF) $S(t)$ represents the probability that an SCML system will survive at least until time t into the mission, and is given by:

$$S(t) = e^{-\alpha(1-\beta)\gamma t} \tag{3.4}$$

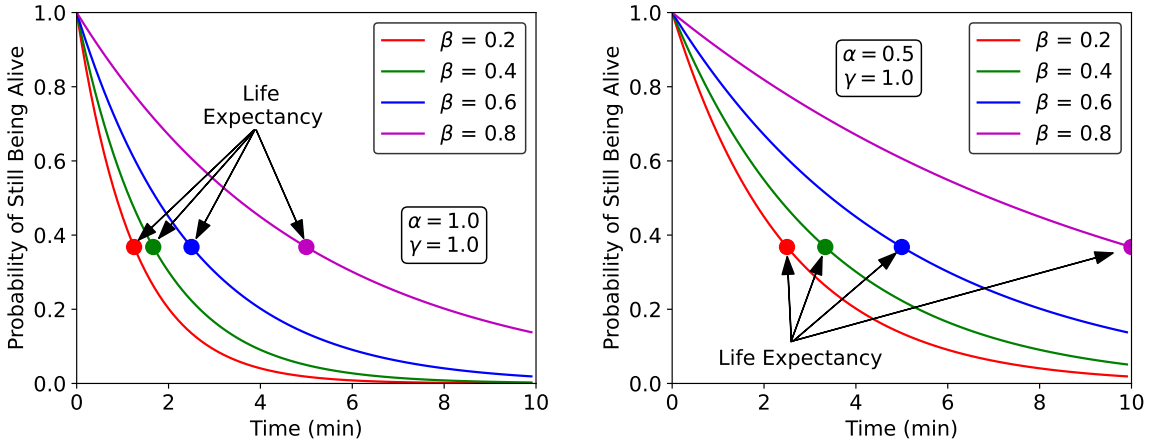
Figure 3.3(a) graphically illustrates this equation for different threat arrival rates (i.e., values of α), when $\beta = 0$ and $\gamma = 1.0$. In other words, SCML is nonexistent or inoperable, and hence there are no CMs. Further, every threat is lethal. As α decreases, the curve moves further to the right. Longer missions become less suicidal. The X-coordinate of the mean value of each curve represents life expectancy; it increases as α is lowered. Figure 3.3(b) shows the impact of different lethalities, when $\alpha = 1.0$ and $\beta = 0$. In other words, the threat arrival rate is fixed and SCML is nonexistent or inoperable. As γ decreases (i.e., threats become less lethal), the curve moves further to the right and life expectancy increases. As Figures 3.3(a) and 3.3(b) show, α and γ can be viewed as scaling parameters that extend life expectancy as their values are lowered. However, they do not change the shape of the curves.



(a) $\beta = 0$ and $\gamma = 1.0$ (b) $\alpha = 1.0$ and $\beta = 0$

For all curves above, the mean value is $1/e$, which is ˜0.368.

Figure 3.3: Impact of Threat Arrivals & Lethality (no SCML)



(a) $\alpha = 1.0$ and $\gamma = 1.0$ (b) $\alpha = 0.5$ and $\gamma = 1.0$

For all curves above, the mean value is $1/e$, which is ˜0.368.

Figure 3.4: Using a Fixed Pre-Trained Model in SCML

Figures 3.4(a) and 3.4(b) show that this observation is also true when SCML is used with a pre-trained model of fixed β (i.e., no learning is done during a mission). Although life expectancy improves as β improves, the shape of the curves is unchanged. If SCML systems existed today, the state of the art would be that of classic ML operations: ML models are pre-trained in the cloud, and then deployed in scouts. Improving β during a mission changes the shape of the curves, as discussed below.

3.1.4 Model Improvement During a Mission

Suppose β_0, β_1, β_2, ..., β_N correspond to improving models that are installed at times t_0, t_1, t_2, ..., t_N during a single mission. Exactly how these new models are obtained by a scout can be ignored for this high-level discussion. For example, they may be transmitted wirelessly from the cloud, based on new models created via learning from other concurrent missions. Another possibility is for Live Learning on the current mission to train the models on the scouts. Regardless of how improved models are obtained, the CDF for survivability, $S(t)$, is now composed of piecewise segments corresponding to different values of β_i, representing the recall of model i. During the earliest part of the mission, β_0 applies; for the next segment, β_1 applies; and so on. Using Equation 3.4, the decay during each segment relative to its start can be expressed as:

$$S_0(t) = e^{-\alpha(1-\beta_0)\gamma(t-t_0)} \text{ for the piece between } t_0 \text{ and } t_1 \tag{3.5}$$

$$S_1(t) = e^{-\alpha(1-\beta_1)\gamma(t-t_1)} \text{ for the piece between } t_1 \text{ and } t_2 \tag{3.6}$$

$$\dots$$

$$S_i(t) = e^{-\alpha(1-\beta_i)\gamma(t-t_i)} \text{ for the piece between } t_i \text{ and } t_{i+1} \tag{3.7}$$

These equations represent the stand alone survivability values for each time interval, which currently do not take into account their respective distance from the origin (start ofthe mission). Using the notation $\lambda_i = \alpha(1-\beta_i)\gamma$, we can derive the final expression applicable to any time interval, which is dependent on all previous time intervals. First, from Equation 3.5, because $t_0 = 0$, we have: $S_0(t_0) = 1$ according to the inverse CDF of the exponential distribution. Again, because all $S(t)$ in interval i are dependent on previous intervals $0 \rightarrow i-1$, we must compute the initial value of $S(t)$ for each interval. For instance, the initial value of interval $i = 1$ is $S_1(t_1)$. In a piecewise distribution, $S_1(t_1) = S_0(t_1)$, as t_1 is where the function transitions from interval 0 to 1 ($S_0(t_1)$ is the final value of $S(t)$ in interval 0). To compute $S_0(t_1)$, we simply evaluate Equation 3.5 at t_1 (using λ_i from above):

$$S_0(t_1) = e^{-\lambda_0(t_1-t_0)} \tag{3.8}$$

We combine this result (the initial value for the interval $t_1 \rightarrow t_2$) with Equation 3.6:

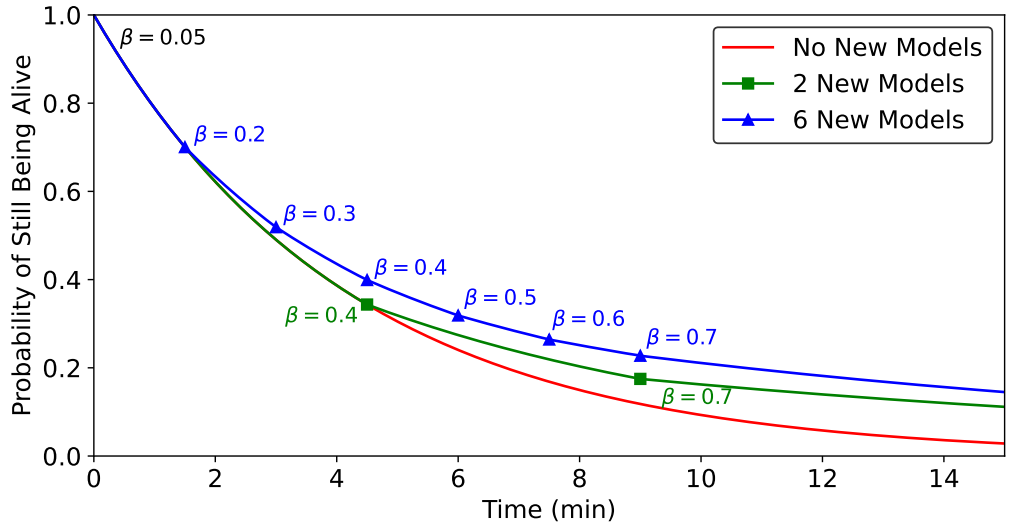
$$\begin{aligned} S_1(t) &= S_0(t_1)e^{-\lambda_1(t-t_1)} \\ S_1(t_2) &= S_0(t_1)e^{-\lambda_1(t_2-t_1)} \end{aligned} \tag{3.9}$$

and extend it to the interval $t_2 \rightarrow t_3$:

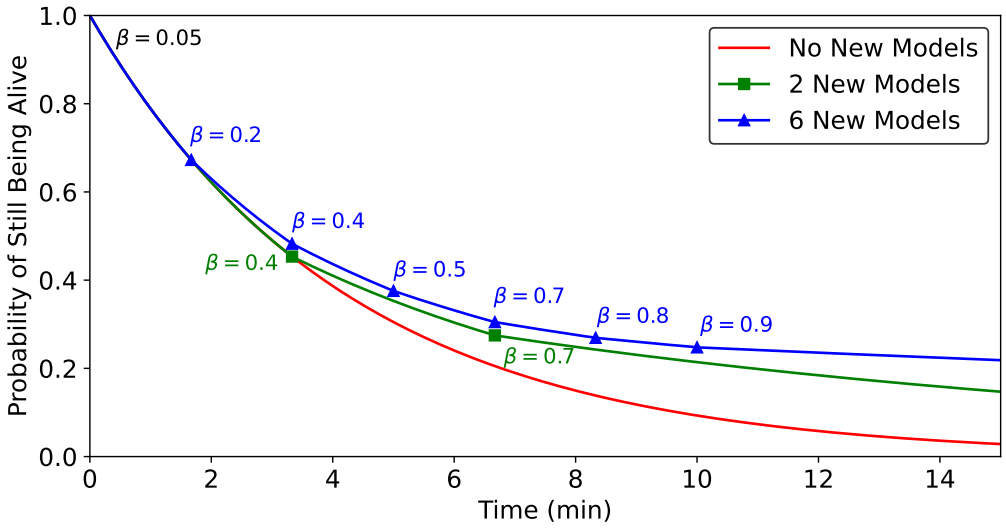
$$\begin{aligned} S_2(t) &= S_1(t_2)e^{-\lambda_2(t-t_2)} \\ S_2(t) &= S_0(t_1)e^{-\lambda_1(t_2-t_1)}e^{-\lambda_2(t-t_2)} \\ S_2(t) &= e^{-\lambda_0(t_1-t_0)}e^{-\lambda_1(t_2-t_1)}e^{-\lambda_2(t-t_2)} \end{aligned} \tag{3.10}$$

From Equation 3.10, we have a pattern that can be generalized below:

$$S_i(t) = e^{-\lambda_0(t_1-t_0)}e^{-\lambda_1(t_2-t_1)} \dots e^{-\lambda_{i-1}(t_i-t_{i-1})}e^{-\lambda_i(t-t_i)} \tag{3.11}$$



(a) Similar Asymptotic Fates



(b) Divergent Asymptotic Fates

For all curves, $\alpha = 1.0$, $\beta = 0.05$ at $t = 0$, and $\gamma = 0.25$.

Figure 3.5: Survivability with New Models of Different Asymptotic Fates

where the combination of all terms prior to the final term represents the initial value of the interval i. A final generalized expression for $S(t)$ in the interval $t_i \leq t \leq t_{i+1}$ is then given by:

$$S(t) = e^{-\left(\sum_{j=0}^{i-1} \lambda_j (t_{j+1} - t_j)\right) - \lambda_i (t - t_i)} \quad \text{for } t_i \leq t \leq t_{i+1} \tag{3.12}$$

We illustrate the subtleties of this survivability equation using two slightly different examples in Figure 3.5. Figure 3.5(a) shows $S(t)$ for two hypothetical missions, indicated by the green and blue curves. Both missions improve β from a miserable value of 0.05 at the start of the mission to a respectable value of 0.7 for the last part of the mission. However, the number of steps in which this improvement occurs and their timing are different for the two missions. Both of these curves in Figure 3.5(a) diverge significantly from the red curve (when no learning occurs) as the mission progresses. This confirms that learning during a mission improves survivability. The figure shows divergence between the blue and green during the period from $t = 2$ to $t = 9$. However, the two missions have the same final values of $\beta = 0.7$, and they decay at the same rate asymptotically beyond $t = 9$. In other words, regardless of the early part of their journeys, all missions arriving at a certain β will have comparable future fates beyond that point.

In the example illustrated by Figure 3.5(b), the green and blue missions reach values of $\beta = 0.4$ and $\beta = 0.7$ at the same moments in time. But their histories of model improvement are different elsewhere. Most importantly, the blue mission continues to receive model improvements of $\beta = 0.8$ and $\beta = 0.9$ beyond the last model of $\beta = 0.7$ shared with green. Hence, the later parts of the blue and green curves are divergent — survivability is asymptotically higher for the blue mission.

3.1.5 Estimating Life Expectancy

Figures 3.5(a) and 3.5(b) qualitatively show that model improvement during a mission extends life expectancy. Quantifying this improvement requires computing the mean of a piecewise exponential distribution. Fortunately, a closed-form expression for this can be derived. We start by adapting the definition of the expected value (life expectancy) of an exponential distribution to that of a piecewise exponential in Equation 3.13:

$$E[T] = \sum_{j=0}^{R} \int_{t_j}^{t_{j+1}} I_j t e^{-\lambda_j (t - t_j)} dt \tag{3.13}$$

In this equation, R is the number of new models installed during the mission. λ_j is the average arrival rate of lethal, unneutralized or undetected threats for interval j. Improving models means that λ_j decreases over time. $t_j \rightarrow t_{j+1}$ represents the time interval over which model j performs inferencing. I_j represents the scaling factor for each time interval of the associated model, which ensures the total area under the probability density function (PDF) is 1.0. More specifically, it is the Y-coordinate of the PDF at the start of the time interval of model j. Thus for model 0 at $t = 0$, the Y-coordinate is $I_0 = \lambda_0$ as any constant rate exponential distribution would have an initial coordinate of $(0, \lambda)$.

To solve for this expected value, we must first derive a recursive equation for each I_j. Each I_j is dependent on all previous intervals: their lethal threat arrival rates λ_j and the lengths of each time interval $t_{j+1} - t_j$. We start with the definition of the exponential PDF:

$$\int_0^{\infty} \lambda e^{-\lambda t} dt = 1 \tag{3.14}$$

In order to derive a function for each I_j, we need to decompose the total area of under the PDF into separate subregions, one for each interval. In Equation 3.15, A_i represents the area under the PDF for the single interval i:

$$\begin{aligned} A_i &= \int_{t_i}^{t_{i+1}} I_i e^{-\lambda_i (t-t_i)} dt \\ &= \frac{I_i}{\lambda_i} \left(1 - e^{-\lambda_i (t_{i+1}-t_i)}\right) \end{aligned} \tag{3.15}$$

We can then sum over all the interval subregions for the piecewise exponential distribution and use the result from Equation 3.15:

$$\begin{aligned} \sum_{i=0}^{R} \int_{t_i}^{t_{i+1}} I_i e^{-\lambda_i (t-t_i)} dt &= 1 \\ \sum_{i=0}^{R} \frac{I_i}{\lambda_i} (1 - e^{-\lambda_i (t_{i+1}-t_i)}) &= 1 \end{aligned} \tag{3.16}$$

Of note here is that in the final interval when $i = R$, $t_{i+1} = \infty$ even though we do not assume an infinite mission duration. This is simply to ensure we can derive the final desired result mathematically, described next. Each time we advance one time interval, we combine the cumulative area of the PDF starting from $t = 0$ to t_i and the remaining area in the interval $t_i \rightarrow t_{i+1}$, that together must sum to one. Because we don't have knowledge as to the boundaries of any remaining intervals, we always have to assume that this is the final interval, i.e. $t_{i+1} = \infty$. However, if a new model is retrained, then the area under the current interval is added to the cumulative area and we repeat the process. Equation 3.17 shows this separation. The term on the left represents the cumulative sum of the area under the PDF inclusive of interval 0 to interval $j - 1$. The term in the middle represents the remaining area contained within the interval from $t_j \rightarrow \infty$. This method ensures the total area always sums to one.

$$\begin{aligned} \left(\sum_{i=0}^{j-1} \int_{t_i}^{t_{i+1}} I_i e^{-\lambda_i (t-t_i)} dt \right) + \int_{t_j}^{\infty} I_j e^{-\lambda_i (t-t_j)} dt &= 1 \\ \left(\sum_{i=0}^{j-1} \frac{I_i}{\lambda_i} (1 - e^{-\lambda_i (t_{i+1}-t_i)}) \right) + \frac{I_j}{\lambda_j} &= 1 \end{aligned} \tag{3.17}$$

We can then solve for I_j, the scaling factor for interval j, which clearly is recursively dependent on all previous I_i, back to I_0, which is simply λ_0.

$$I_j = \lambda_j \left[1 - \sum_{i=0}^{j-1} \frac{I_i}{\lambda_i} (1 - e^{-\lambda_i (t_{i+1}-t_i)}) \right], \forall j \tag{3.18}$$

Substituting Equation 3.18 for I_j into Equation 3.13, we derive the life expectancy for a piecewise exponential as:

$$E[T] = \sum_{j=0}^{R} \int_{t_j}^{t_{j+1}} \lambda_j \left[1 - \sum_{i=0}^{j-1} \frac{I_i}{\lambda_i} (1 - e^{-\lambda_i (t_{i+1}-t_i)}) \right] t e^{-\lambda_j (t-t_j)} dt \tag{3.19}$$

with the following final result:

$$E[T] = \sum_{j=0}^{R} \left(\frac{A_j}{\lambda_j^2} \right) \left[1 - (\lambda_j t_{j+1} + 1) e^{-\lambda_j (t_{j+1} - t_j)} + \lambda_j t_j \right] \tag{3.20}$$

Using the above equation, Table 3.1 presents the life expectancies for the curves in Figures 3.5(a) and 3.5(b). We have validated these results against Monte Carlo simulations of the missions depicted in Figures 3.5(a) and 3.5(b), and obtained identical results. These results show that model improvement during a mission can indeed improve survivability. However, quantifying this improvement in Live Learning implementations is not so simple because of nonlinearities in learning algorithms. For a given threat arrival rate, creating more models during a mission means that each new model is based only on a small increase in training set size over the previous model. This may lead to suboptimal training. Hence, experimental results are needed to complement the intuitive understanding of Live Learning presented so far.

Figure	Curve	Life Expectancy
3.5(a)	Red	4.21
	Green	6.22
	Blue	7.23
3.5(b)	Red	4.21
	Green	7.16
	Blue	14.42

Table 3.1: Life Expectancy

Chapter 4

Experimental Validation of SCML

In a real implementation, do the predicted gains in survivability from Live Learning materialize? To experimentally answer this top-level question, we have created a modified version of Hawk from the open source release in GitHub. Our experiments leverage ML insights from the original Hawk work [18] by closely following its experimental setup. We use the same datasets and classes, data arrival rates, ML models, hyperparameter settings, Live Learning triggering criterion for training new models, etc. Our description below first presents our modifications to Hawk for SCML (§4.1), and then presents other relevant experimental details (§4.2–§4.3).

4.1 Hawk Modifications for SCML

As mentioned earlier (§2), the original Hawk implementation focused exclusively on low backhaul bandwidths (down to 12 kbps). In the context of Figure 2.2, this maps to severe throttling of result transmission for labeling from scouts. However, SCML has value at any network bandwidth. Human labeling bandwidth, rather than wireless network bandwidth, may sometimes be the bottleneck. We have therefore modified the Hawk implementation to use end-to-end queue backpressure to throttle result transmission from scouts. This aims to prioritize human labeling effort so that, regardless of whether the bottleneck is the human or the network, it is the most high-value results that are labeled. We achieve this using an end-to-end token-based flow control mechanism [77].

Each scout maintains a queue of results awaiting transmission. This queue is sorted by score received on the ML model that inferenced it. A data item that is processed later by a scout may receive a higher score than many earlier data items. The high score suggests that it is likely to be a TP. Since TPs are rare, they merit early attention by the human labeler in order to rapidly enlarge the training set and speed up learning. A long FIFO queue at the human labeler would delay the labeling of this high-value item. At the same time, maintaining too short a labeling queue may lead to the human labeler being stalled waiting for work during periods of low network bandwidth. Further, human think time may vary widely across data items. Some items may be labeled almost effortlessly, while a few require extended think times. The token-based mechanism balances these disparate forces. Its goal is to ensure that the data ingress rate never exceeds the bottleneck throughput, whose location and value can vary dramatically over

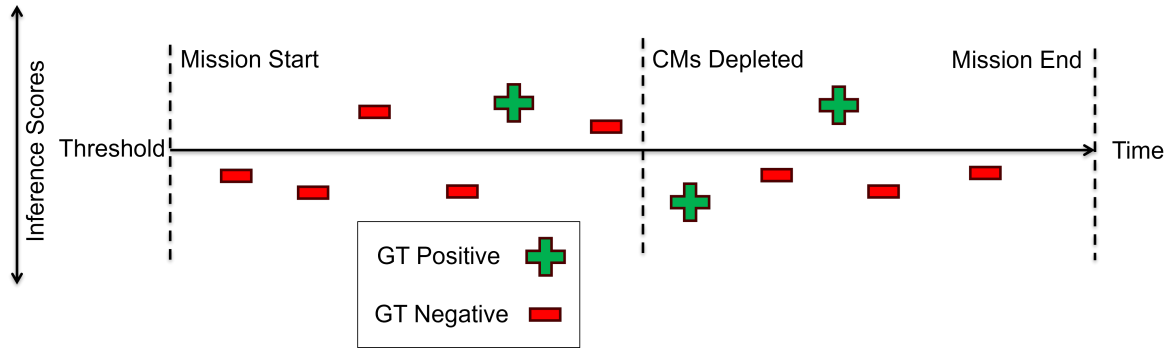


Figure 4.1: Simple SCML Mission Timeline

time. As the bottleneck changes, the token flow adapts to preserve the high value of human labeling effort under all conditions.

We used the terms TP and FN as they are currently known in ML vernacular for the simplified analytical model (§3). However, in the SCML context, we must define each term specifically for the experimental model as they have unique impacts on survivability. The analytical model in part depended on the rate of FNs experienced by the system during the mission, which is determined by the threat arrival rate (α) and the recall (β) of the ML model. $\alpha(1-\beta)$ represents the average arrival rate of FNs. Because the impact of TPs, FPs, and TNs are not accounted for in the analytical model, their impact on survivability is described here. Each inferenced sample is assigned one of four possible outcomes, according to its score and the *countermeasure deployment threshold*. This is the same threshold as the traditional classification threshold.

- False Positive (FP): The inference score is greater than or equal to the CM deployment threshold but there is no ground truth threat object in the sample.
- True Positive (TP): The inference score is greater than or equal to the CM deployment threshold and there is at least one ground truth threat object in the sample.
- False Negative (FN): The inference score is less than the CM deployment threshold but there is at least one ground truth threat object in the sample.
- True Negative (TN): The inference score is less than the CM deployment threshold and there is no ground truth threat object in the sample.

Figure 4.1 depicts a toy SCML mission timeline. Time with key events are shown on the x-axis and inference scores on the y-axis. The solid horizontal black line represents the CM deployment threshold. Once the mission starts, the first red "$-$" represents a TN as it is a ground truth (GT) negative and its score is below the threshold. The third red "$-$" is an FP as its score is above the threshold, for which a CM is wasted. The first "$+$" is a TP as it is a GT positive and its score is above the threshold, which is neutralized by a CM. At some point during the mission, all countermeasures may be depleted, as noted by the dashed line in the middle of the timeline. The second overall "$+$" is an FN as it is below the threshold, which may become a lethal threat. The last "$+$" is a TP so it is properly classified, but CMs have been depleted so it cannot be neutralized and therefore may become a lethal threat. Finally, the mission ends at some point in time. This graphic is meant to show the key components of an SCML mission.

Now that we've shown how an abstracted toy SCML mission flows from start to finish, here

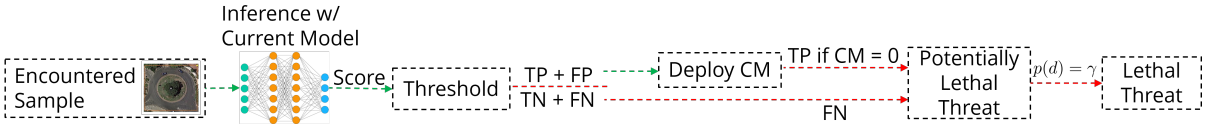


Figure 4.2: Simple SCML Workflow for each Inferenced Sample

we explain the detailed, yet simple logical steps for each inferenced sample. Along with the threat lethality, in the analytical model we could compute the approximate survivability as a function of time with the threat arrival rate and model recall. However, under realistic, resource-constrained conditions, the system may also suffer catastrophic destruction from a lethal threat even when it is accurately detected, such as in the case of a TP (referenced in the previous paragraph). Figure 4.2 depicts the simple SCML workflow logic and the impact of each type of classification result on survivability, chronologically from left to right:

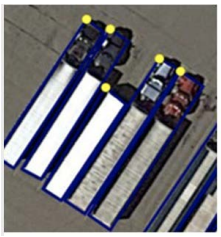
1. The sample is sensed from the environment by the scout and fed to the model.
2. The sample is inferenced by the current model to generate a score in $[0, 1]$, representing the probability the respective threat object is present in the sample.
3. A simple boolean comparison of the inference score and the current CM deployment threshold determines one of four possible classification results: FP, TP, FN, TN.
4. If the result is an FP or TP (score is at least the threshold), a countermeasure is deployed (if available), otherwise no CM is deployed.
5. The sample is a potentially lethal threat under two conditions: 1) The result is an FN or 2) The result is a TP when CMs have been depleted.
6. Any potentially lethal threat becomes lethal with probability γ, the lethality parameter. The current probability of survival (at a given instant in time) of the system is then reduced by this factor. For example, if the lethality parameter is 10%, upon encountering the first potentially lethal threat, the probability of survival is now $(1 - \gamma)^1 = 1 - \gamma = 90\%$. After encountering the second potentially lethal threat, the probability of survival is $(1 - \gamma)^2 = 81\%$. Thus, after encountering the n^{th} potentially lethal threat in any mission, the probability of survival is $(1 - \gamma)^n$.

4.2 Datasets

Our experiments use three publicly available datasets from drone surveillance, planetary exploration, and underwater sensing. Example images from these datasets are shown in Figures 4.3 to 4.5. For all datasets and classes, the base rate is below 0.1%.

(a) Roundabout
(TPs=336)

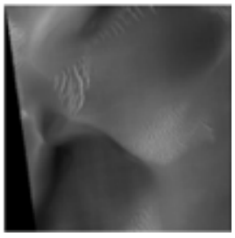
(b) Swimming Pool
(TPs=335)



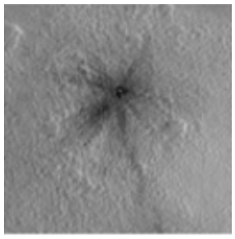
(c) Large Vehicle
(TPs=357)

(d) Airplane
(TPs=350)

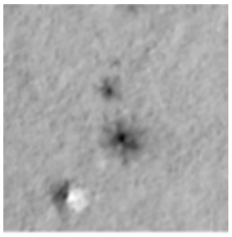
Figure 4.3: Examples of DOTA Target Classes



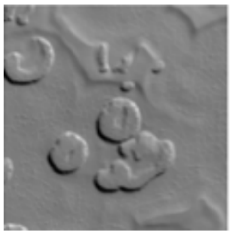
(a) Dark Dune
(TPs=64)



(b) Impact Ejecta
(TPs=64)

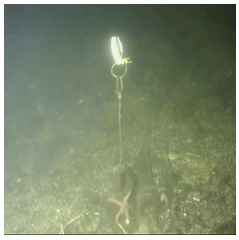


(c) Spider
(TPs=64)



(d) Swiss Cheese
(TPs=64)

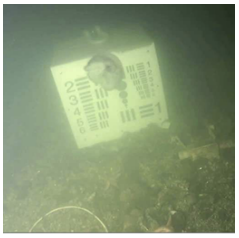
Figure 4.4: Examples of HiRISE Target Classes



(a) Starfish
(TPs=370)

(b) Shrimp
(TPs=564)

(c) Small Fish
(TPs=564)



(d) JellyFish
(TPs=584)

Figure 4.5: Examples of Brackish Target Classes

Drone Surveillance (DOTA) [Figure 4.3]: The drone-captured 15-class DOTA v1.0 dataset [19] consists of 2806 images, half of which are used as mission data. Image resolution ranges from 800×800 pixels to 4000×4000 pixels. Images are tiled into 256×256 pixels, yielding 252,231 usable tiles. During a mission, a scout receives an average of 180 tiles in 20 seconds.

Planetary Exploration (HiRISE) [Figure 4.4]: The HiRISE dataset [78] consists of images collected by the Mars Reconnaissance Orbiter. There are 73,031 labeled images, tiled into 227×227 pixels. There are 8 target classes. During a mission, a scout receives an average of 100 tiles in 20 seconds.

Underwater Sensing (Brackish) [Figure 4.5]: The Brackish dataset [79] contains labeled images of marine animals in a brackish strait with varying visibility. There are 14,518 images of 1080p resolution tiled into 256×256 pixels, for a total of 563,829 tiles. There are 6 target classes. During a mission, a scout receives an average of 100 tiles in 3 seconds.

4.3 Mission Configuration

Although our simple analytical model (§3) focused on missions with a single logical scout, our experimental evaluation mirrors the original Hawk work by using seven separate physical scouts that share TPs. The learning rate of the whole ensemble is thus determined by their collective rate of threat encounters. The scouts have identical hardware, as described below (§4.4), and connectivity between them is 1 Gbps. In the context of survivability, it is necessary to specify what constitutes mission success when some scouts die but others survive. Our evaluation assumes that all scouts have to reach their destination for the mission to be successful. In other words, the death of even a single scout results in mission failure. The more complex analysis contained within §4.8 defines mission success as only M out of N scouts having to arrive at the destination.

4.4 Hardware

Each scout consists of a 6-core 3.6 GHz Intel® Xeon® E5–1650v4 processor, 32 GB memory, 4 TB disk storage for image data, and an NVIDIA® GTX 1060 GPU. Today, a typical scout would be configured with embedded hardware such as an NVIDIA Jetson AGX Orin [80], weighing 1.58 kg. Such hardware could easily be carried by an autonomous drone, unmanned spacecraft, or underwater submersible. Rather than the austere network settings of the original Hawk work, we assume that ample network bandwidth (1 Gbps) is available. The bottleneck is now human labeling rate (Figure 2.2), emulated by code that returns the ground-truth label after a configurable think time (1 second in our experiments).

4.5 Live Learning Timeline

What is the timeline of Live Learning on a scout? This question forms the backdrop for all other experimental results presented in this chapter. As shown by our analytical model (§3.1.4), the timing and magnitude of model improvement greatly influences survivability. When the improvement is achieved through Live Learning, its timeline depends on the dataset, attributes of the threat class (especially its base rate), and numerous ML hyperparameters. As stated, all scouts begin executing Live Learning at the beginning of the mission and continue as long as they have not been destroyed by a lethal threat. To demonstrate such learning independent of SCML functions, we allow all scouts to learn for the whole mission under the assumption they all survive the full duration.

For the class "Roundabout" from the dataset DOTA, the second and third columns of Figure 4.6(a) show this timeline. The initial model, based on a bootstrap training set containing just 20 TPs, is weak. Its AUC (area under the precision-recall curve on a held-out test set) is merely 0.20. The first new model is installed 146 seconds into the mission, and has a slightly improved AUC of 0.29. At 388 seconds, a better model with AUC 0.45 is installed; at 547 seconds, a model with AUC 0.47 is installed; and so on. The AUC improvement is mostly monotonic, with occasional plateaus or reversals due to the randomness inherent in the learning process. The

	Class: Roundabout		Class: Pool	
Gen	Time(s)	AUC	Time(s)	AUC
0	0	0.20 (0.03)	0	0.02 (0.03)
1	146 (22)	0.29 (0.12)	560 (130)	0.23 (0.08)
2	388 (87)	0.45 (0.01)	797 (95)	0.36 (0.20)
3	547 (100)	0.47 (0.03)	932 (115)	0.51 (0.08)
4	787 (115)	0.54 (0.03)	1219 (106)	0.62 (0.03)
5	1216 (94)	0.54 (0.06)	1537 (102)	0.70 (0.03)
6	1706 (100)	0.59 (0.03)	1973 (83)	0.72 (0.07)
7	2253 (143)	0.58 (0.03)	2502 (41)	0.78 (0.04)
8	2960 (68)	0.62 (0.04)	3103 (55)	0.81 (0.06)
9			3872 (30)	0.79 (0.04)
	4010 (5)		4014 (4)	

(a) Dataset: DOTA

	Class: Impact Ejecta		Class: Swiss Cheese	
Gen	Time(s)	AUC	Time(s)	AUC
0	0	0.45 (0.05)	0	0.71 (0.04)
1	267 (48)	0.49 (0.01)	195 (28)	0.85 (0.03)
2	522 (32)	0.64 (0.07)	425 (49)	0.90 (0.03)
3	919 (116)	0.63 (0.03)	798 (71)	0.90 (0.03)
4	1449 (14)	0.69 (0.04)	1361 (42)	0.95 (0.02)
	1710 (0)		1709 (0)	

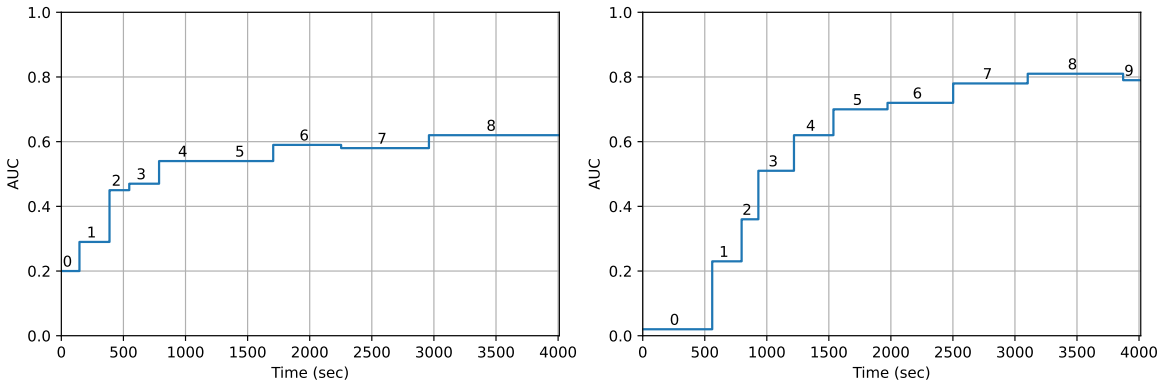
(b) Dataset: HiRISE

	Class: Starfish		Class: Jellyfish	
Gen	Time(s)	AUC	Time(s)	AUC
0	0	0.16 (0.04)	0	0.16 (0.02)
1	238 (158)	0.38 (0.06)	148 (44)	0.28 (0.03)
2	374 (164)	0.48 (0.06)	328 (47)	0.42 (0.03)
3	539 (169)	0.57 (0.06)	472 (62)	0.43 (0.02)
4	768 (133)	0.64 (0.02)	667 (26)	0.49 (0.03)
5	1082 (172)	0.67 (0.04)	907 (34)	0.52 (0.01)
6	1458 (184)	0.72 (0.03)	1287 (71)	0.55 (0.02)
7	1945 (216)	0.73 (0.04)	1750 (78)	0.57 (0.02)
8	2607 (211)	0.78 (0.01)	2211 (91)	0.59 (0.01)
9	3537 (96)	0.82 (0.01)	2909 (15)	0.61 (0.01)
10			3785 (35)	0.64 (0.01)
	4037 (5)		4037 (5)	

(c) Dataset: Brackish

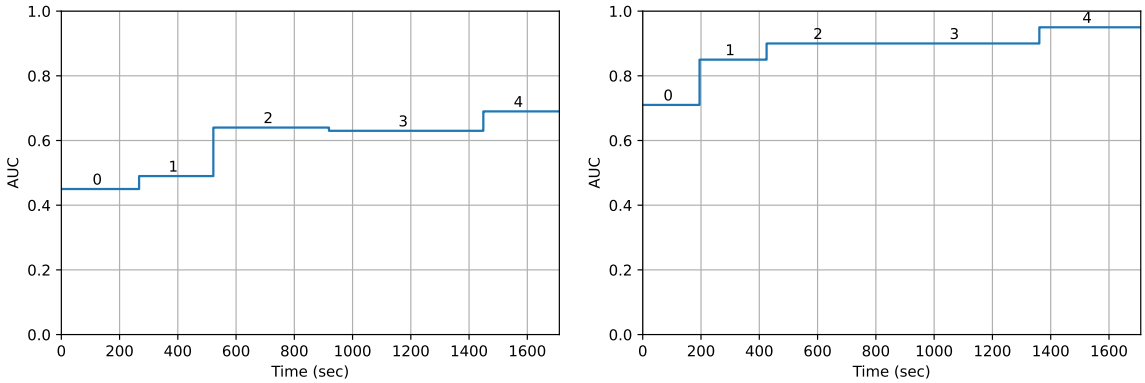
These results are for representative classes of the three datasets used in our experiments (§4.2). For each class, the time into the mission at which a new generation of a model was installed and its AUC (area under the precision-recall curve for a held-out test data set) are shown. The last row shows the time by which all data from the dataset was delivered. Numbers in parentheses are standard deviations across three runs of an experiment with different random number seeds.

Figure 4.6: Mission Timeline for Live Learning



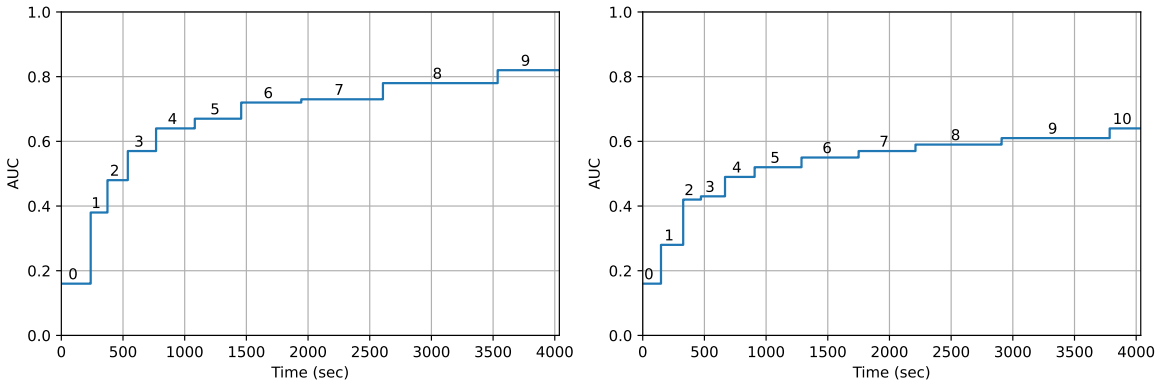
Class = Roundabout Class = Pool

(a) Dataset: DOTA



Class = Impact Ejecta Class = Swiss Cheese

(b) Dataset: HiRISE



Class = Starfish Class = Jellyfish

(c) Dataset: Brackish

Figure 4.7: Visualization of Live Learning Timeline (compare with Figure 3.1(b))

Dataset	DOTA		HiRISE		Brackish	
Class	Roundabout	Pool	Swiss Cheese	Impact Ejecta	Starfish	Jellyfish
Tiles Processed	252,225 (8)	252,216 (18)	60,064 (0)	60,064 (0)	563,755 (90)	563,777 (30)
Tiles Labeled	4117 (216)	4346 (385)	1529 (0)	1529 (0)	4045 (0)	4045 (0)
TPs Found	272 (2)	274 (2)	63 (0)	53 (1)	300 (7)	385 (5)
TPs Total	336	335	64	64	370	584

These results are from the same experiments as Figure 4.6. Numbers in parentheses are standard deviations. "TPs Total" refers to the number of TPs known to be present from ground truth.

Figure 4.8: Transmission & Labeling Efficiency of Live Learning

model with AUC 0.62, installed close to the end of the mission, has clearly improved far beyond the initial model. The last two columns of Figure 4.6(a) confirm that this theme of model improvement during a mission holds for a different class of the DOTA dataset. Figures 4.6(b) and 4.6(c) confirm that it also holds for classes of the HiRISE and Brackish datasets.

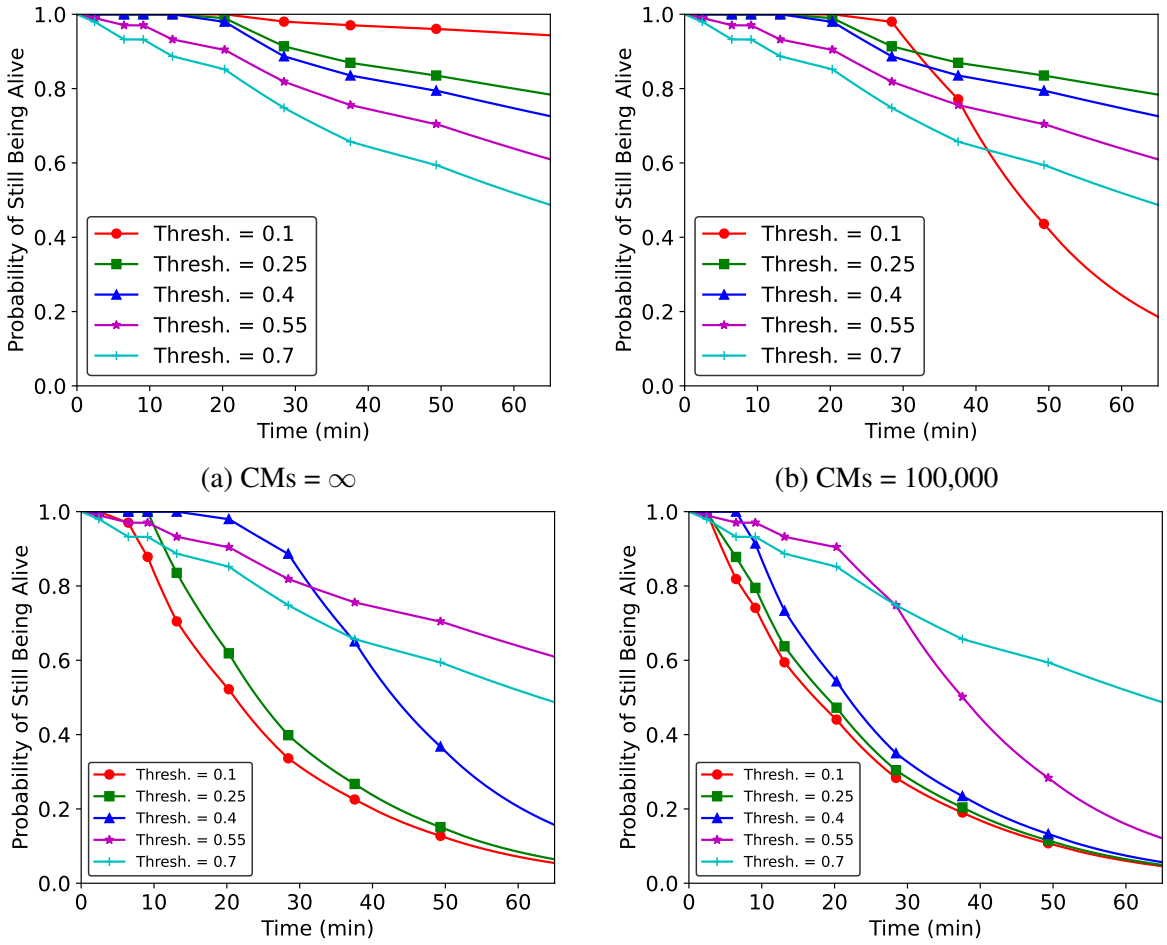
The data in Figure 4.6 is visualized in Figure 4.7 using the canonical Live Learning timeline shown earlier in Figure 3.1(b). For all datasets and classes, significant improvement in model AUC is seen over the life of the mission. The magnitude and timeline of the improvement is dependent on the dataset and class as well as various system-level settings like network bandwidth, domain expert labeling rate, and the number of scouts in the mission. For example, if only two scouts were inferencing a data stream of the same threat density at the same rate, there would far fewer TPs available to label, and thus decrease the frequency of retraining.

Figure 4.8 shows the transmission and labeling efficiency of Live Learning for the experiments shown in Figure 4.6. For class Roundabout of dataset DOTA, the total number of tiles processed by the scouts is 252,225. Of these, only 4117 were transmitted for labeling by the human in the loop. In spite of the extreme class imbalance ($\sim$ 0.1% base rate), 272 TPs were found out of 336 ground truth TPs. This general theme holds for all of the datasets and classes shown in Figure 4.8. These results confirm that Live Learning is frugal in its use of network bandwidth and human effort, but highly effective in discovering TPs.

4.6 Importance of Model Precision

Our analytical modeling (§3.1.3–§3.1.5) focused exclusively on the recall metric (β) of the model used for SCML. High recall ensures that almost all threats are detected. With an infinite supply of CMs, detection ensures neutralization. Our analysis ignored the complementary metric of model precision, which determines how many CMs are wasted. In practice, no real mission can carry an infinite supply of CMs and wasted CMs hurt survivability. If the supply of CMs on a mission is exhausted (see Figure 4.1), no further neutralization of threats is possible. Even perfect ML (i.e., detection of every threat beyond that point) cannot save the mission. Any threat (detected or undetected) may then be fatal, and survival is determined solely by the arrival rate of threats (α) and their lethality (γ).

The balance between recall and precision is determined by the tunable CM deployment



(a) CMs = ∞

(b) CMs = 100,000

(c) CMs = 20,000

(d) CMs = 10,000

All graphs above assume a threat lethality of $\gamma = 0.01$. These results correspond to class Roundabout for the DOTA dataset.

Figure 4.9: Survivability of a Live Learning Mission with Limited CMs at 1% Lethality

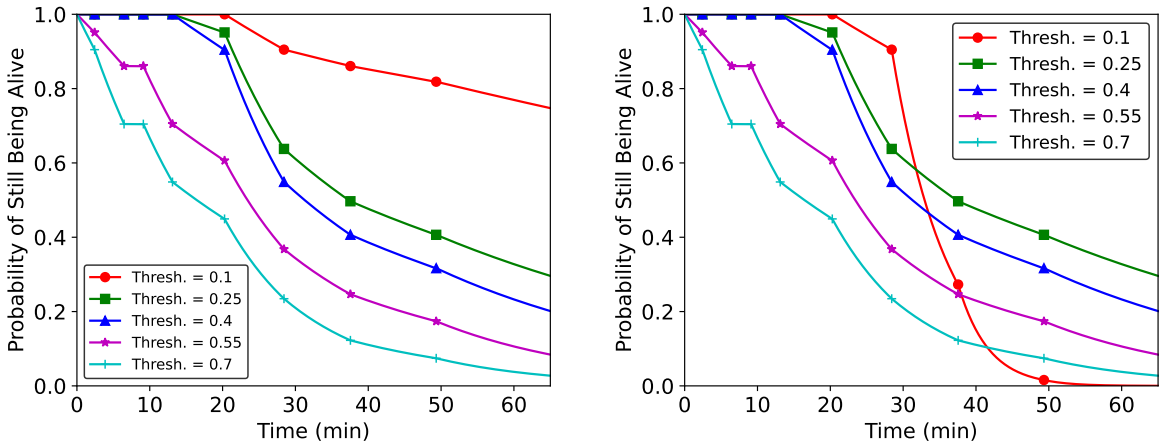
threshold (from § 4.1) in $[0, 1]$. Summarizing the previous explanations of the impacts of each type of classification result, some of the positives may be FPs, and some of the negatives may be FNs. With an ample supply of CMs, there is no incentive to be frugal — FPs don't matter at all, only FNs do. As the supply of CMs falls, FPs become problematic.

Threshold	Live Learning	No Learning
0.10	185,313	206,148
0.25	62,155	138,454
0.40	28,290	97,308
0.55	15,102	69,766
0.70	7,904	48,399

Dataset: DOTA Class: Roundabout

Figure 4.10: Total CM Usage

For class Roundabout of the dataset DOTA and a low threat lethality of $\gamma = 0.01$, Figure 4.9 shows survivability for four different initial supplies of CMs. With an infinite supply of CMs (Figure 4.9(a)), lowering threshold is always a win. A threshold of 0.1 results in a likelihood of survival to mission completion that is over 90%. Higher thresholds lead to lower likelihood of mission success. At a threshold of 0.7, the likelihood of success is only about 50%. Figure 4.9(b)



(a) CMs = ∞

(b) CMs = 100,000

(c) CMs = 20,000

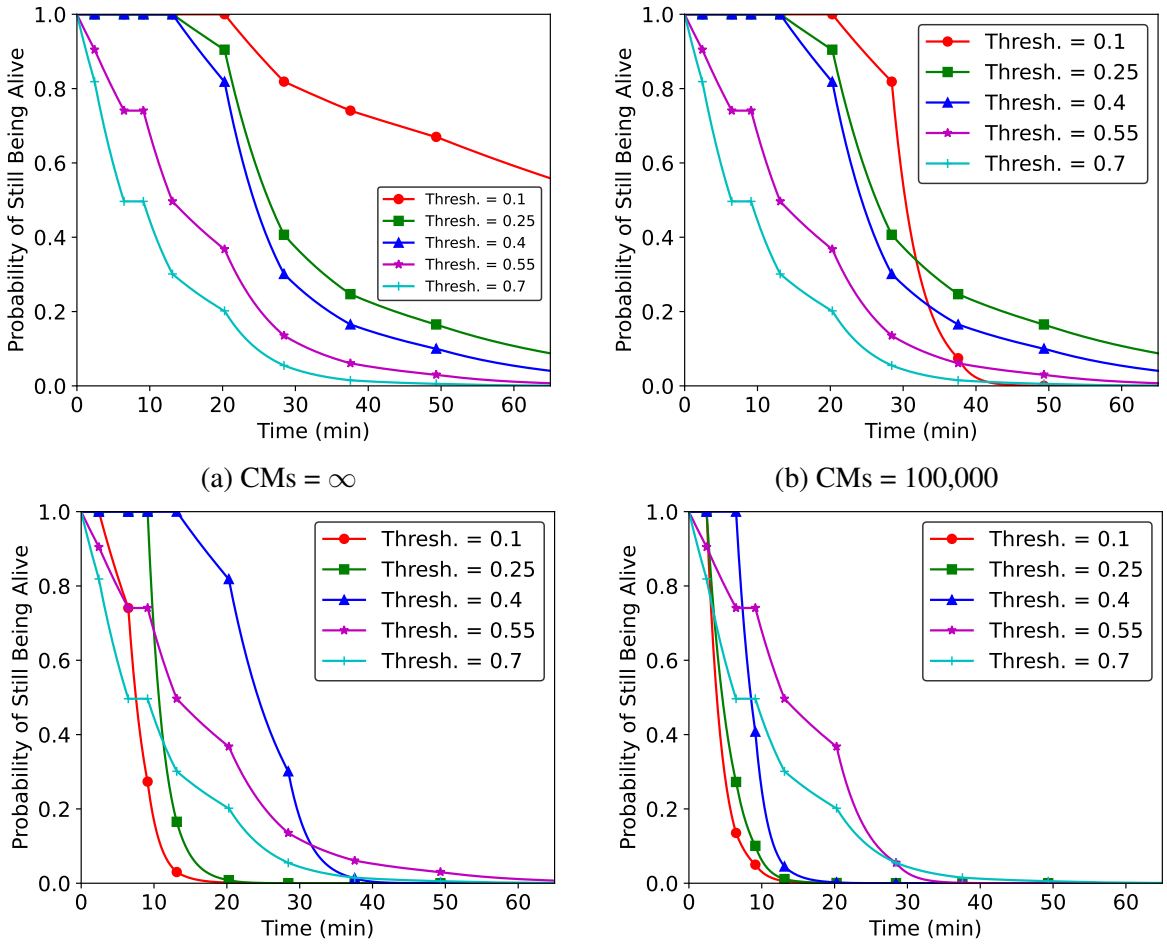
(d) CMs = 10,000

All graphs above assume a threat lethality of $\gamma = 0.05$. These results correspond to class Roundabout for the DOTA dataset.

Figure 4.11: Survivability of a Live Learning Mission with Limited CMs at 5% Lethality

shows that a threshold of 0.1 results in too many FPs when the initial supply is only 100,000 CMs (distributed across all scouts). All CMs are used up by roughly 30 minutes into the mission, and survivability sharply drops after that. For this case, a threshold of 0.25 leads to the highest chances of mission success (roughly 80%). This general trend continues when CMs are limited to 20,000 (Figure 4.9(c)) and 10,000 (Figure 4.9(d)). In the latter case, the high threshold of 0.7 offers the best chances of survival to mission success.

From the viewpoint of conserving CMs, the importance of model improvement during a mission becomes clear by examining the total CMs used on successful missions. Figure 4.10 compares the numbers of CMs consumed by a mission using Live Learning to that consumed when no improvement is made to the initial model. Note that only data from successful missions is included here. The thresholds used in Figure 4.10 are the same as in Figure 4.9. At low threshold, the difference between the CMs used in the two cases is only modest — both are equally profligate in their use of CMs. As threshold rises, the improved model quality of Live



(a) CMs = ∞

(b) CMs = 100,000

(c) CMs = 20,000

(d) CMs = 10,000

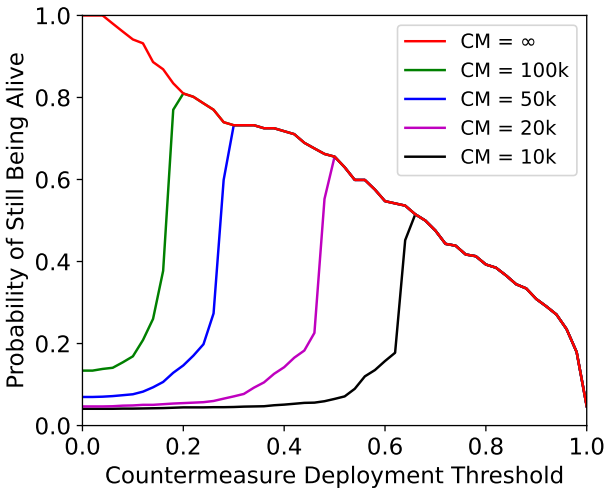
All graphs above assume a threat lethality of $\gamma = 0.10$. These results correspond to class Roundabout for the DOTA dataset.

Figure 4.12: Survivability of a Live Learning Mission with Limited CMs at 10% Lethality

Learning begins to have substantial effect. Far fewer CMs are used at higher thresholds with Live Learning. This is because the iterative retraining process greatly suppresses the scores of all ground truth negatives, which, as the models improve, converts more FPs into TNs (saving precious CMs as a result).

The lethality of threats is low ($\gamma = 0.01$) for the results shown in Figures 4.9 and 4.10. In other words, the impact of FNs is mild — there is only a 1% chance that an undetected threat will destroy the system. What happens when threats are more lethal?

Figure 4.11 and Figure 4.12 show the impact of increasing lethality to 5% and 10% respectively. Relative to Figure 4.9, the drop in survivability is apparent in all cases. Even with an infinite number of CMs and a threshold of 0.1, Figure 4.11(a) shows a mission success below 80%. Figure 4.12(a) shows a mission success just below 60%. These are in contrast to a value well above 90% in Figure 4.9(a). This is because even five outlier FNs under the low threshold of 0.1 in the last 40 minutes of the mission results in a likelihood of mission success at about 57%.



Total CMs	Opt. Threshold	Success Likelihood
10,000	0.65	53.1%
20,000	0.49	65.6%
50,000	0.29	74.0%
100,000	0.19	82.6%

Dataset: DOTA Class: Roundabout $\gamma = 0.01$

The Y axis of the graph on the left shows the probability of still being alive at the end of the mission. The table on the right gives numeric values extracted from the graph on the left.

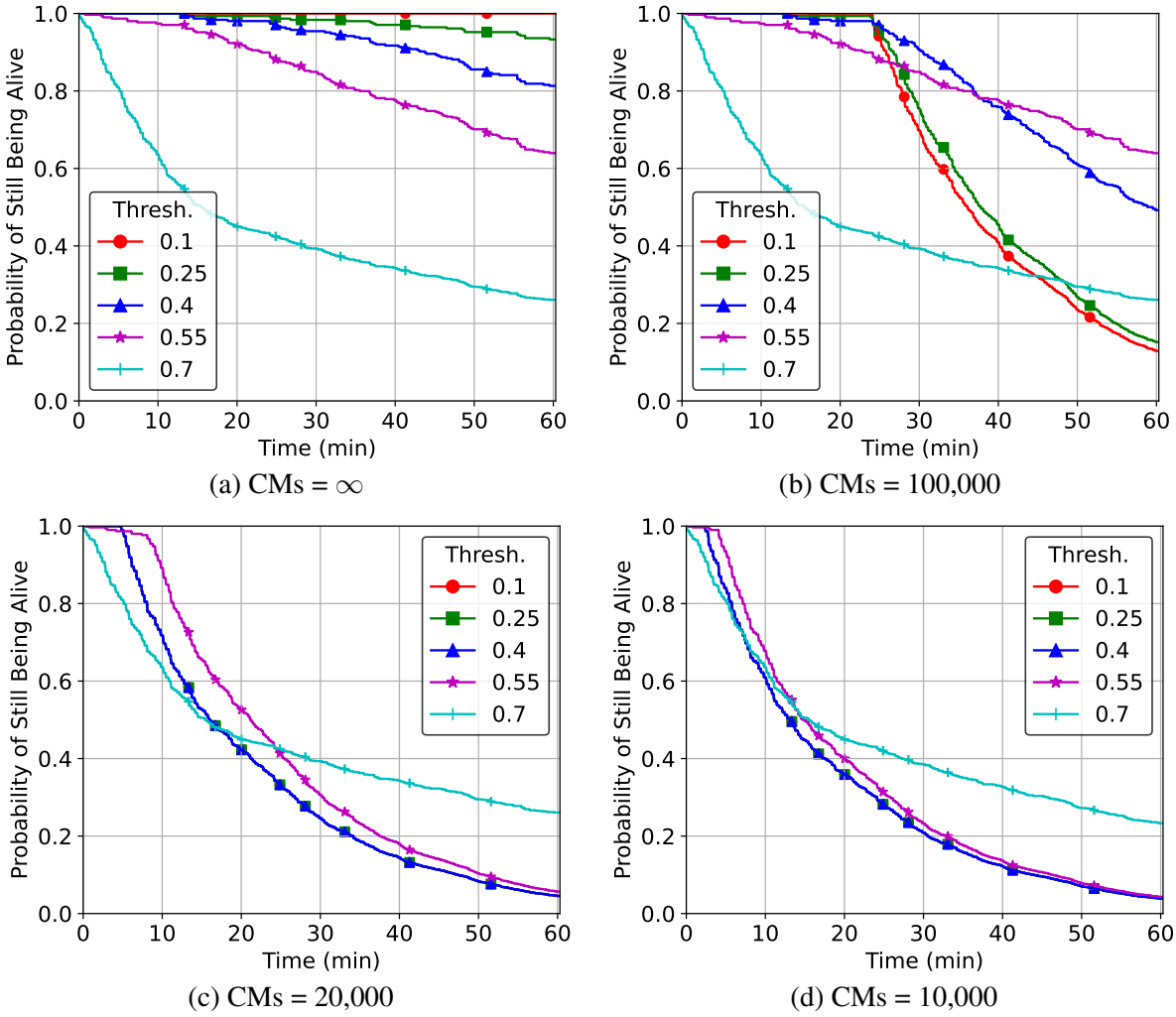
Figure 4.13: Optimal Threshold for Mission Success

At higher lethality, running out of CMs is also riskier. With a limit of 20,000 CMs or lower, Figures 4.12(c) and 4.12(d) show that there is virtually no hope of mission success at any threshold. With 100,000 CMs, Figure 4.12(b) shows a glimmer of hope for mission success at thresholds of 0.25 and 0.4, but none at other thresholds. The same general pattern holds in Figure 4.11. The same general pattern of results also holds across diverse classes and datasets.

If a certain number of CMs are available at the start of a mission, what threshold should a scout select for the highest likelihood of success? If too low a threshold is selected, CMs may be exhausted before the end of the mission. This is likely to lead to mission failure because of inability to neutralize a correctly detected threat late in the mission. If too high a threshold is selected, recall will be unnecessarily low. This will also hurt the chances of survival to mission completion. The optimal threshold results in the last CM being consumed just before mission completion. Ending the mission with CMs left over does not enhance survivability.

With the number of CMs as a parameter, Figure 4.13 shows how threshold affects likelihood of mission completion. With an infinite number of CMs (red curve), there is no point at which CMs are exhausted. Reducing threshold therefore always improves survivability. Operating with a threshold close to zero is optimal. For all cases involving a finite number of CMs, the point of intersection of the relevant curve with the red curve gives the optimal threshold. Any lower a value causes CMs to be used up too soon. Any higher a value leads to CMs being left over at the end of a completed mission.

The previous sets of experimental results have shown survivability as a function of time for class Roundabout of dataset DOTA. It is important compare these results to another class to understand whether and how survivability is affected by the unique characteristics of each class. Figure 4.14 shows the survivability for different numbers of CMs for class Swimming Pool of dataset DOTA. With infinite CMs in Figure 4.14(a), we see similar behavior as Roundabout: the



(a) CMs = ∞ (b) CMs = 100,000

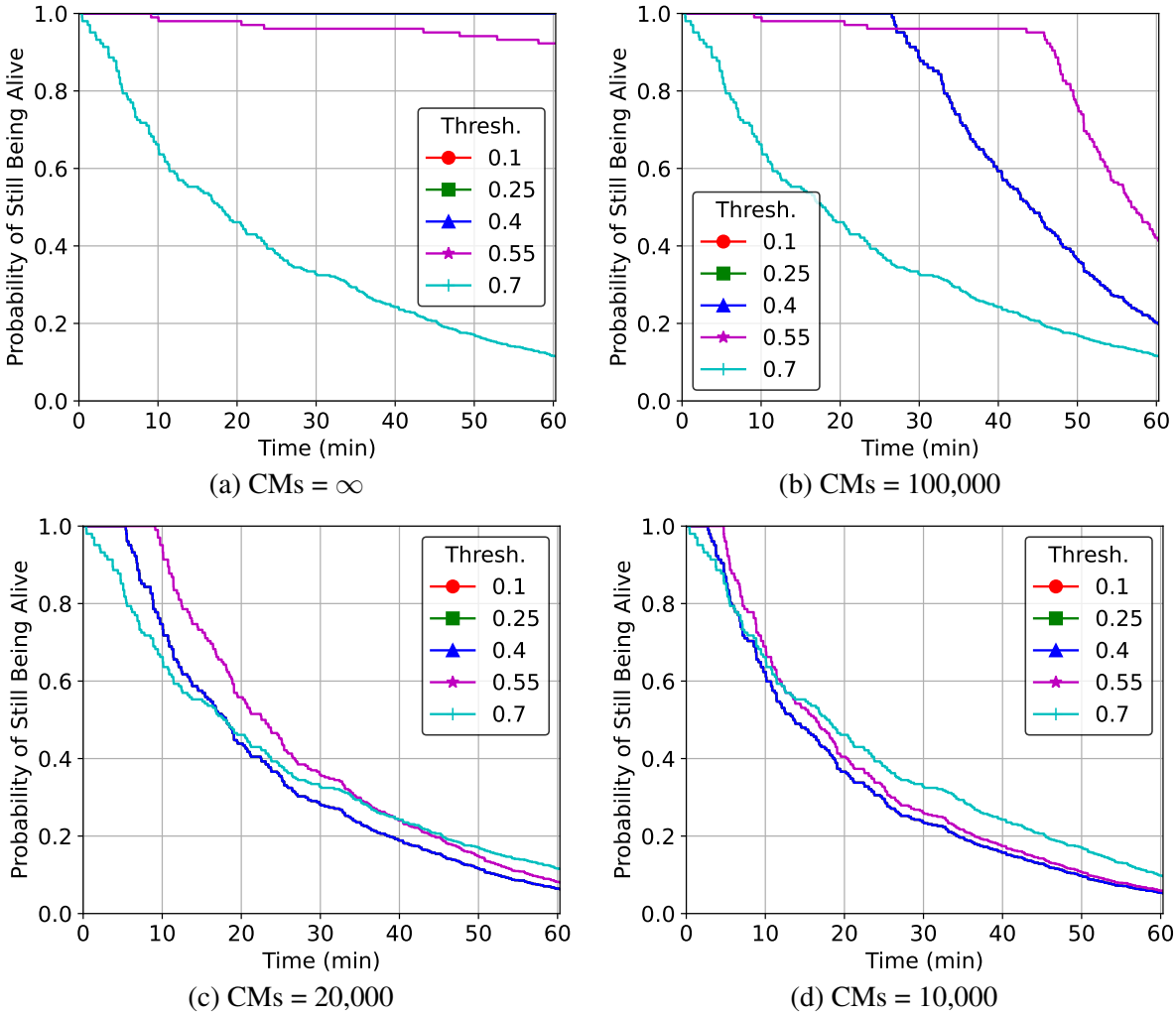
(c) CMs = 20,000 (d) CMs = 10,000

All graphs above assume a threat lethality of $\gamma = 0.01$. These results correspond to a Live Learning mission of class Swimming Pool for the DOTA dataset.

Figure 4.14: Survivability of a Live Learning Mission with Limited CMs at 1% Lethality

highest threshold of 0.7 has the lowest likelihood of success as FNs are the only threats to scouts. Conversely, the lowest threshold of 0.1 in this case achieves 100% likelihood of success as it does not experience a single FN during the mission. As the threshold increases, more FNs occur and thus probability of success decreases.

In Figure 4.14(b), thresholds of 0.7 and 0.55 are clearly unaffected by the reduced number of available CMs (100,000 across all seven scouts), yet the remaining thresholds deplete theirs at roughly 25 minutes into the mission. Notice that the three lowest thresholds all deplete their CMs at adjacent points in time: first 0.1, then 0.25, and then 0.55, but all within a couple of minutes of each other. In the first 25 minutes, virtually all FPs (approximately 14,300 per scout) had scores above 0.4. However, CMs were never depleted for the next highest threshold of 0.55, indicating that few FP scores rose above that level. This confirms that Live Learning is performing one of its tasks, which is to reduce the scores of negatives over time to minimize CM waste.



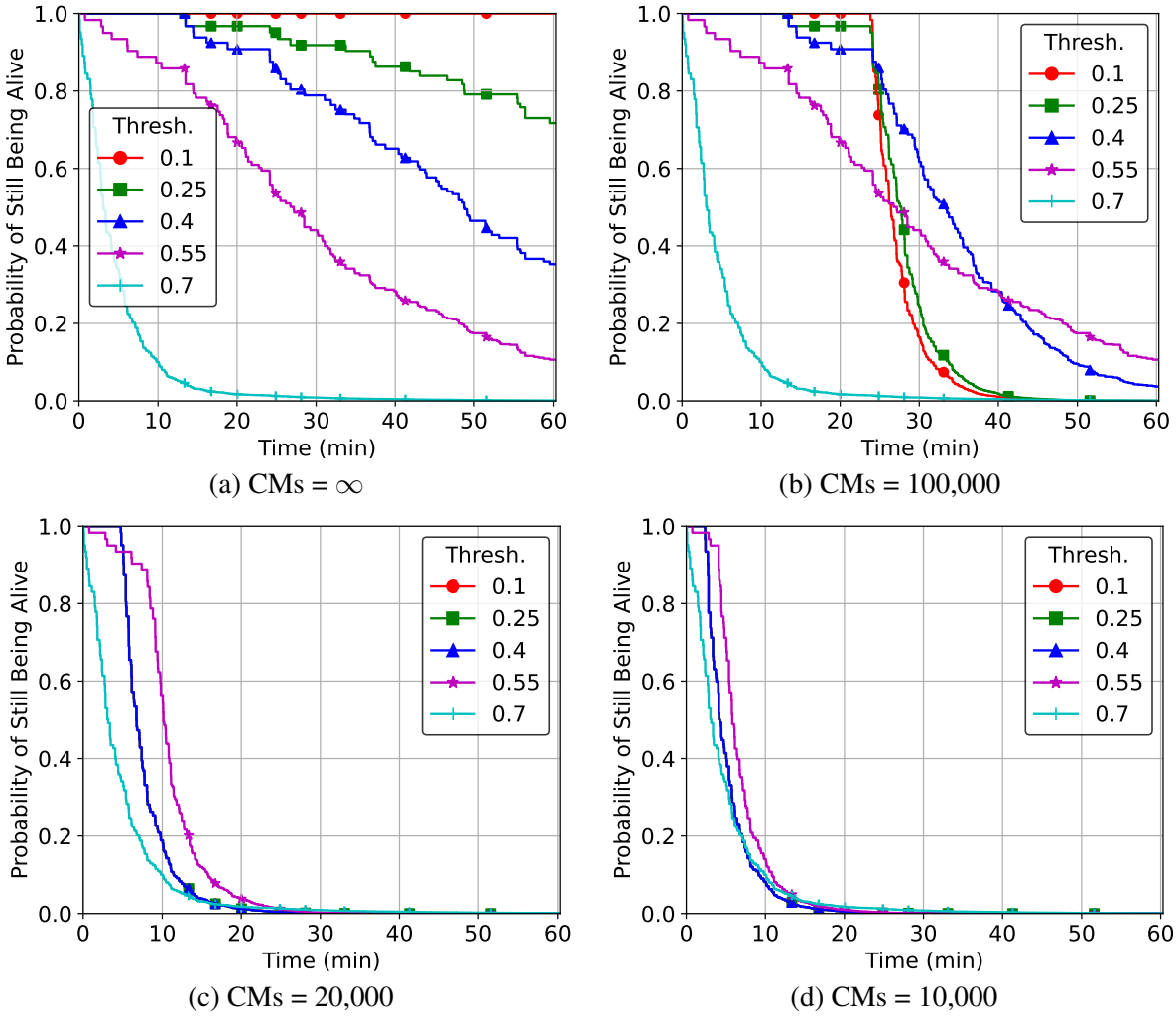
(a) CMs = ∞ (b) CMs = 100,000

(c) CMs = 20,000 (d) CMs = 10,000

All graphs above assume a threat lethality of $\gamma = 0.01$. These results correspond to a No Learning mission of class Swimming Pool for the DOTA dataset.

Figure 4.15: Survivability of a No Learning Mission with Limited CMs at 1% Lethality

With 20,000 total CMs in Figure 4.14(c), the curve with a threshold of 0.7 is still unaffected as it never depletes its CMs. However, the lowest three thresholds have the same exact survivability curve – they all deplete their CMs early in the mission at roughly five minutes. Once all CMs are depleted, all ground truth positives are potentially lethal threats. These three curves are identical because all TPs after CM depletion have scores above 0.4. The 0.55 threshold curve also depletes its CMs, but slightly later than the others at about nine minutes, resulting in a final probability of survival barely above that of lowest three thresholds. Figure 4.14(d) shows a subtle shift downward in the curve with 0.7 threshold in the second half of the mission. The first 30 minutes are identical to the previous curves but the last 30 minutes depict a slight decrease due to slightly earlier CM depletion. With higher thresholds the *CM depletion cliff* is less pronounced as there are likely to be fewer TPs above 0.7 than 0.4, for example. Although few ground truth positives will have scores below 0.7, due to Live Learning, even a small number can produce significant



(a) CMs = ∞ (b) CMs = 100,000

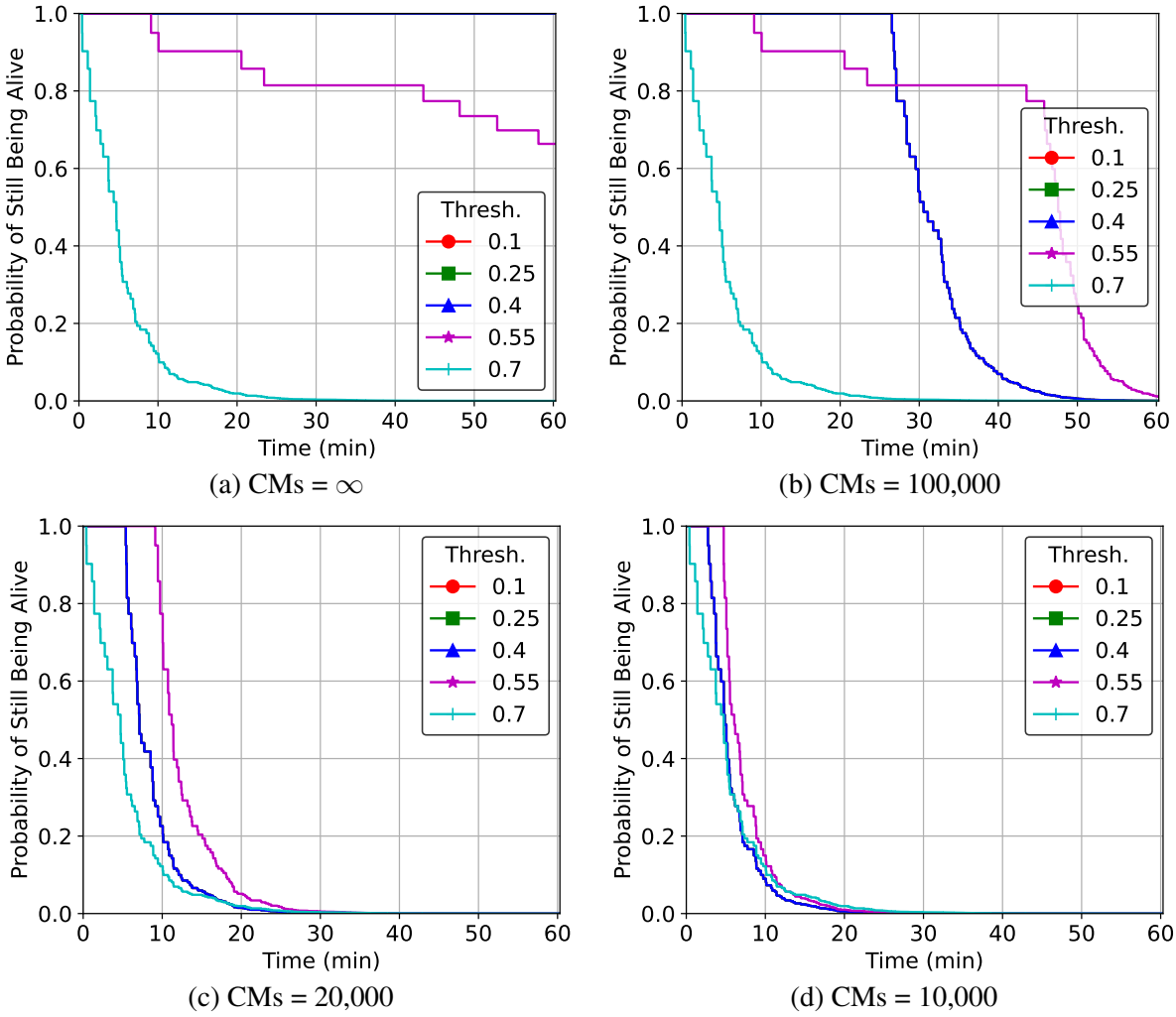
(c) CMs = 20,000 (d) CMs = 10,000

All graphs above assume a threat lethality of $\gamma = 0.05$. These results correspond to a Live Learning mission of class Swimming Pool for the DOTA dataset.

Figure 4.16: Survivability of a Live Learning Mission with Limited CMs at 5% Lethality

differential outcomes in survivability.

We compare our Live Learning results to those of a No Learning mission. Because SCML performance is dependent on individual sample classification and not purely an AUC-like metric, some results respective to a certain threshold can be misleading. Moreover, the number of threat samples in the mission is on the order of 300 so even a handful of poor scores can be quite detrimental to mission success. Figure 4.15 shows equivalent results for a No Learning mission of a class Swimming Pool for dataset DOTA. Figure 4.15(a) only displays curves for the two highest thresholds as the three lowest retain 100% survivability through the end of the mission. One effect of Live Learning is that although it improves the average score across all ground truth positives, it also increases the standard deviation of those scores. This results in some scores being one or two standard deviations below the mean, causing such scores to be lower than they otherwise would be in a No Learning mission. This increase in standard deviation of Live Learning



(a) CMs = ∞

(b) CMs = 100,000

(c) CMs = 20,000

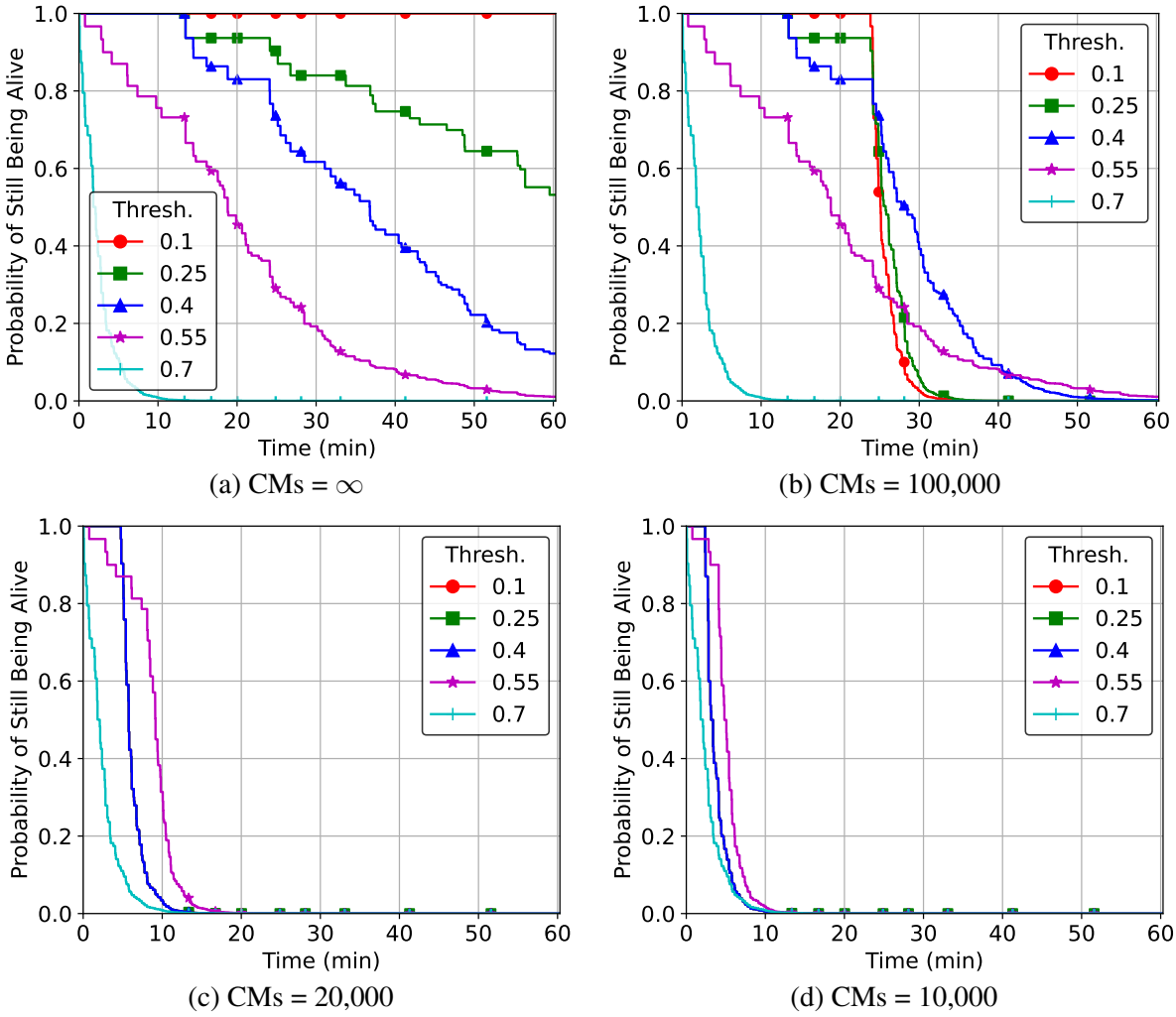
(d) CMs = 10,000

All graphs above assume a threat lethality of $\gamma = 0.01$. These results correspond to a No Learning mission of class Swimming Pool for the DOTA dataset.

Figure 4.17: Survivability of a No Learning Mission with Limited CMs at 5% Lethality

scores can be characterized as *score diffusion*, which is visualized with experimental results in §4.9. The severe corresponding weakness of No Learning is that the mean of all negatives scores are rather high for the duration of the mission as no additional learning occurs. In a mission with infinite CMs, this weakness does not manifest in the results.

Figure 4.15(b) shows the same survivability curves with 100k CMs. Here we have the three lowest thresholds exhibiting the same survivability performance. The three highest: 0.7 curve (13%), 0.55 curve (42%), and 0.4 curve (20%) have lower probability of survival than their Live Learning counterparts: 0.7 curve (27%), 0.55 curve (64%), and 0.4 curve (50%). Thus we observe that even with a modest restriction in CMs, Live Learning increases its ability to enhance survivability. Figure 4.15(c) has similar behavior to 4.14(c) with the exception that the 0.7 threshold curve fell from 27% to 12%. Figure 4.15(d) has almost identical behavior to that of Figure 4.15(c). What we see as an overall theme is that across all thresholds in each



(a) CMs = ∞ (b) CMs = 100,000

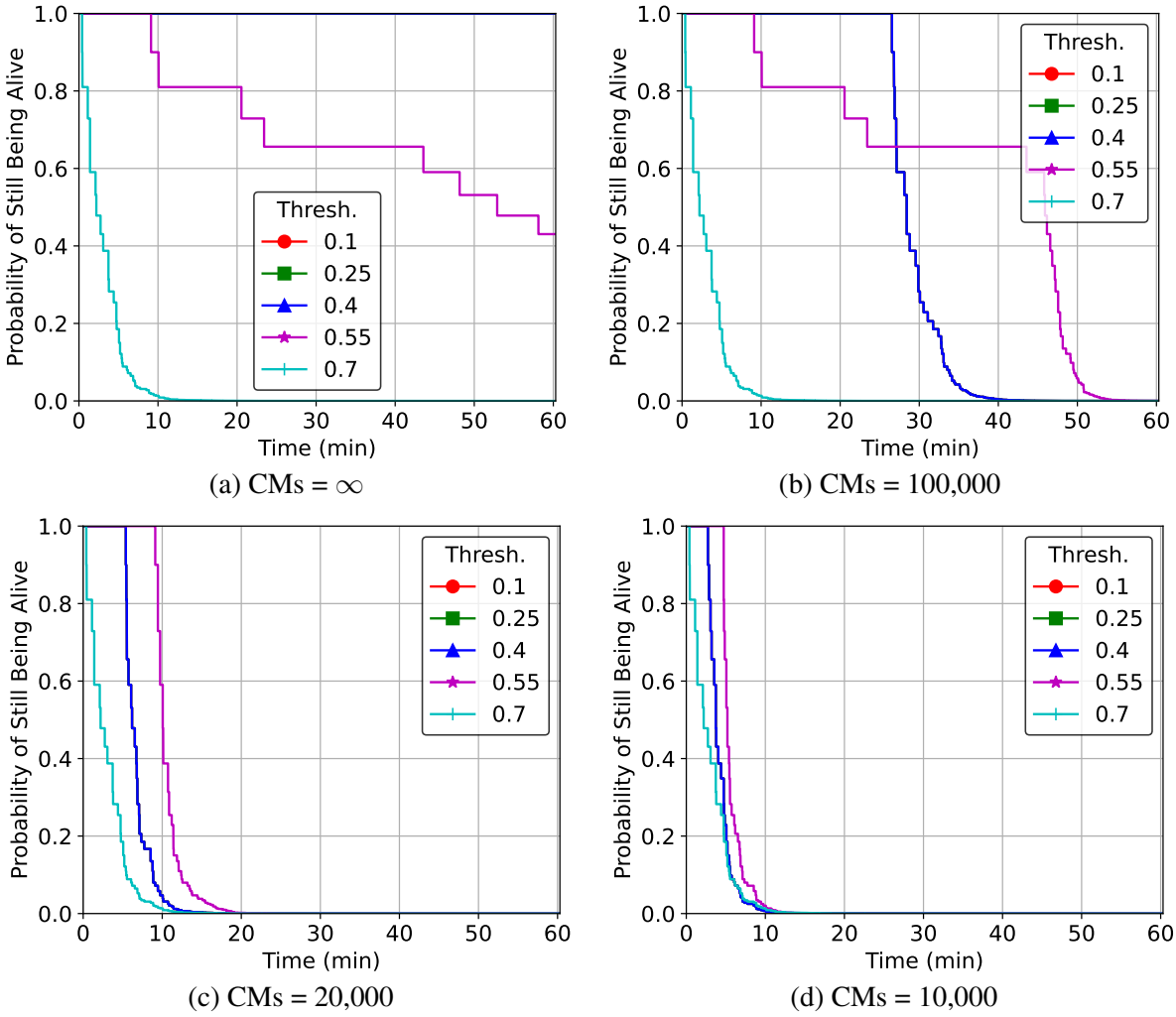
(c) CMs = 20,000 (d) CMs = 10,000

All graphs above assume a threat lethality of $\gamma = 0.1$. These results correspond to a Live Learning mission of class Swimming Pool for the DOTA dataset.

Figure 4.18: Survivability with Limited CMs at 10% Lethality

plot representing each number of CMs, the maximum likelihood of success in Live Learning is always greater than that of No Learning. In other words, even if for a particular threshold and number of CMs that No Learning may outperform Live Learning, the optimal threshold for Live Learning will always result in a greater probability of success than the optimal threshold for No Learning. This clearly demonstrates Live Learning's ability to enhance survivability across various numbers of CMs.

As with class Roundabout, class Swimming Pool suffers the same reduction in survivability with a lethality of 5%, as shown in Figures 4.16 and 4.17. The main observation is that survivability rates at the end of the mission are quite low. The same general rule applies that Live Learning is superior to No Learning for their respective optimal thresholds, however the gain is greatly reduced due to the increase in lethality. For example, the only mentionable improvement is for a threshold of 0.55 at 100k CMs and $\gamma = 1\%$ in Figure 4.17(b) to 11% in Figure 4.16(b).



(a) CMs = ∞

(b) CMs = 100,000

(c) CMs = 20,000

(d) CMs = 10,000

All graphs above assume a threat lethality of $\gamma = 0.1$. These results correspond to a No Learning mission of class Swimming Pool for the DOTA dataset.

Figure 4.19: Survivability with Limited CMs at 10% Lethality

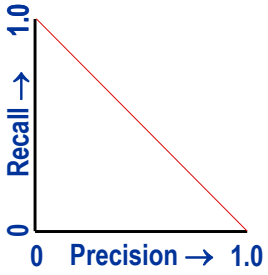
A lethality of 10% in Figures 4.18 and 4.19 simply reduces any such gains in survivability even further, with all missions ending with 0% survivability at all thresholds for the smaller number of available CMs. This is generally what we observed with class Roundabout as well. Overall, we observed that class Swimming Pool maintained a smaller standard deviation of ground truth positive (threat) sample scores as compared to Roundabout. This is because swimming pools are more unique in feature space across diverse environments, at least in the DOTA dataset. Conversely, roundabouts can look quite similar in features to roads, intersections, on-ramps and off-ramps, curvy roads, and so forth. This trait is what influences the differential behavior in some threshold curves for missions of class Swimming Pool.

4.7 Recall-biased Loss Functions

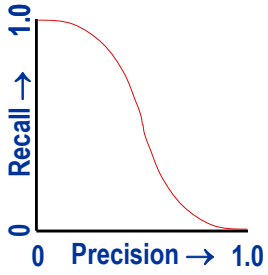
Our analysis and experimental results from the previous sections show that in SCML:

- Recall is always important.
- Precision may or may not be important, depending on the CMs available relative to mission duration and threat density. As long as a scout does not run out of CMs, precision can be traded off for improved recall.

These observations lead to the shape of the precision-recall curve playing an important role in SCML. In other words, it is not just the AUC of a model that matters, but also how that AUC is achieved. This is illustrated by Figure 4.20, which shows two hypothetical precision-recall curves that have an AUC of 0.5. Figure 4.20(a) trades off precision and recall evenly everywhere. In contrast, Figure 4.20(b) is biased in favor of recall at low precision. If one knows ahead of time that a mission can be executed at low precision, then Figure 4.20(b) is preferable to Figure 4.20(a). Live Learning can then be tuned to preferentially improve recall rather than precision.



(a) Unbiased AUC



(b) Biased AUC

Figure 4.20: AUC = 0.5

In the original Hawk implementation, FNs resulted in missed TPs. However, because discards are revisited later using improved models, "near misses" can be rediscovered and correctly interpreted as TPs. On the other hand, FPs always waste bandwidth. The incentives are different in SCML, where a single FN can be fatal. This suggests biasing learning differently from the original Hawk implementation. Tuning for bias can be achieved via the loss function used for training new models during Live Learning. Our experiments use a biased loss function that weights aggregate classification errors on positives (i.e., FNs) twice as heavily as errors on negatives. Hawk's original loss function weights aggregate error on positives and negatives equally.

For class Roundabout of dataset DOTA, Figure 4.21(a) compares CMs used and FNs for these two different loss functions. At all thresholds, Figure 4.21(a) shows fewer FNs at the price of higher CM usage. Figure 4.21(b) uses a different viewpoint to compare biased and unbiased loss functions. It asks "If the total FNs during a completed mission should lie

Threshold	Unbiased		Biased	
	CMs	FNs	CMs	FNs
0.10	185,313	6	251,255	0
0.25	62,155	25	223,433	3
0.40	28,290	33	126,078	12
0.55	15,102	51	49,378	26
0.70	7,904	74	21,587	45

Dataset: DOTA Class: Roundabout

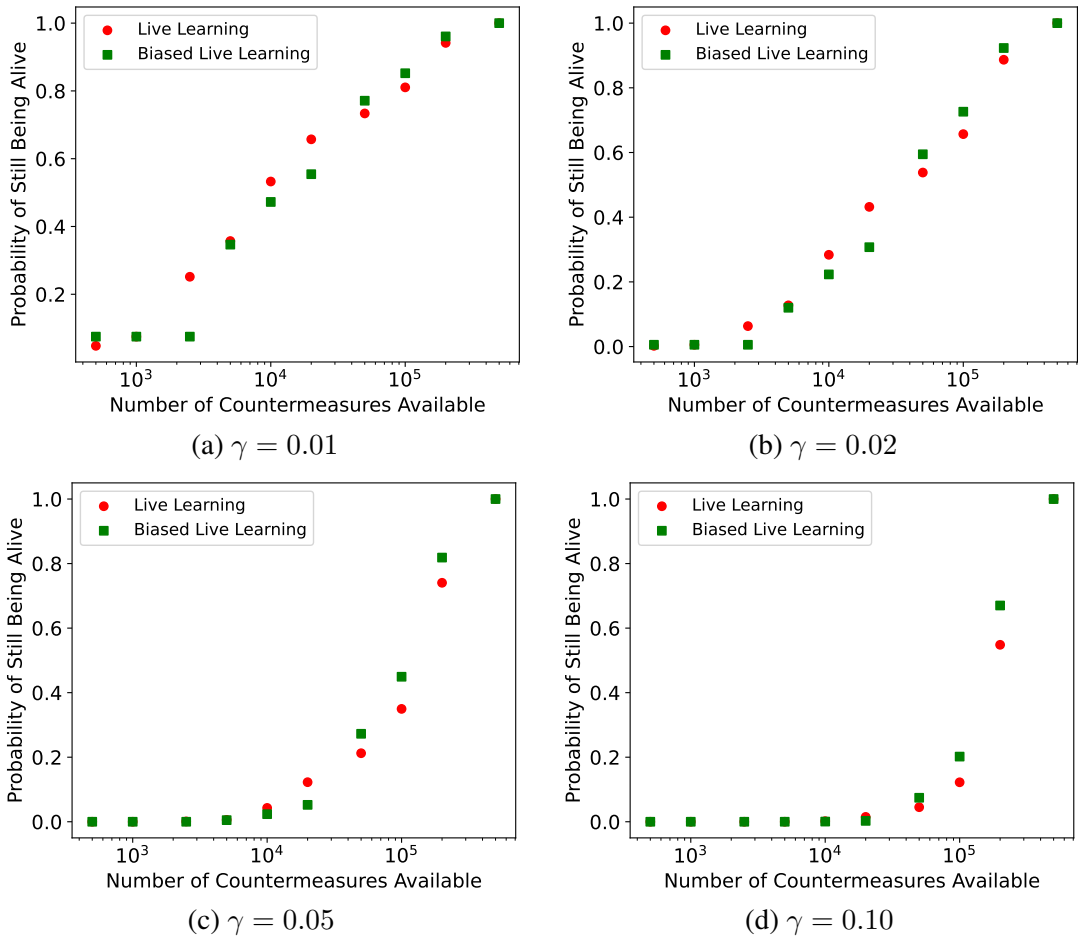
(a) Impact of Biased Loss Function

FNs	Unbiased		Biased	
	CMs	TH	CMs	TH
0-1	237,043	0.05	239,900	0.20
12-13	127,455	0.15	126,078	0.40
25-26	62,155	0.25	49,738	0.55
40-42	81,719	0.50	28,304	0.65
74-76	7,904	0.70	8,961	0.85

Dataset: DOTA Class: Roundabout

(b) Targeting a Specific FN Range

Figure 4.21: Targeted Analysis of Biased Loss Function



(a) $\gamma = 0.01$ (b) $\gamma = 0.02$

(c) $\gamma = 0.05$ (d) $\gamma = 0.10$

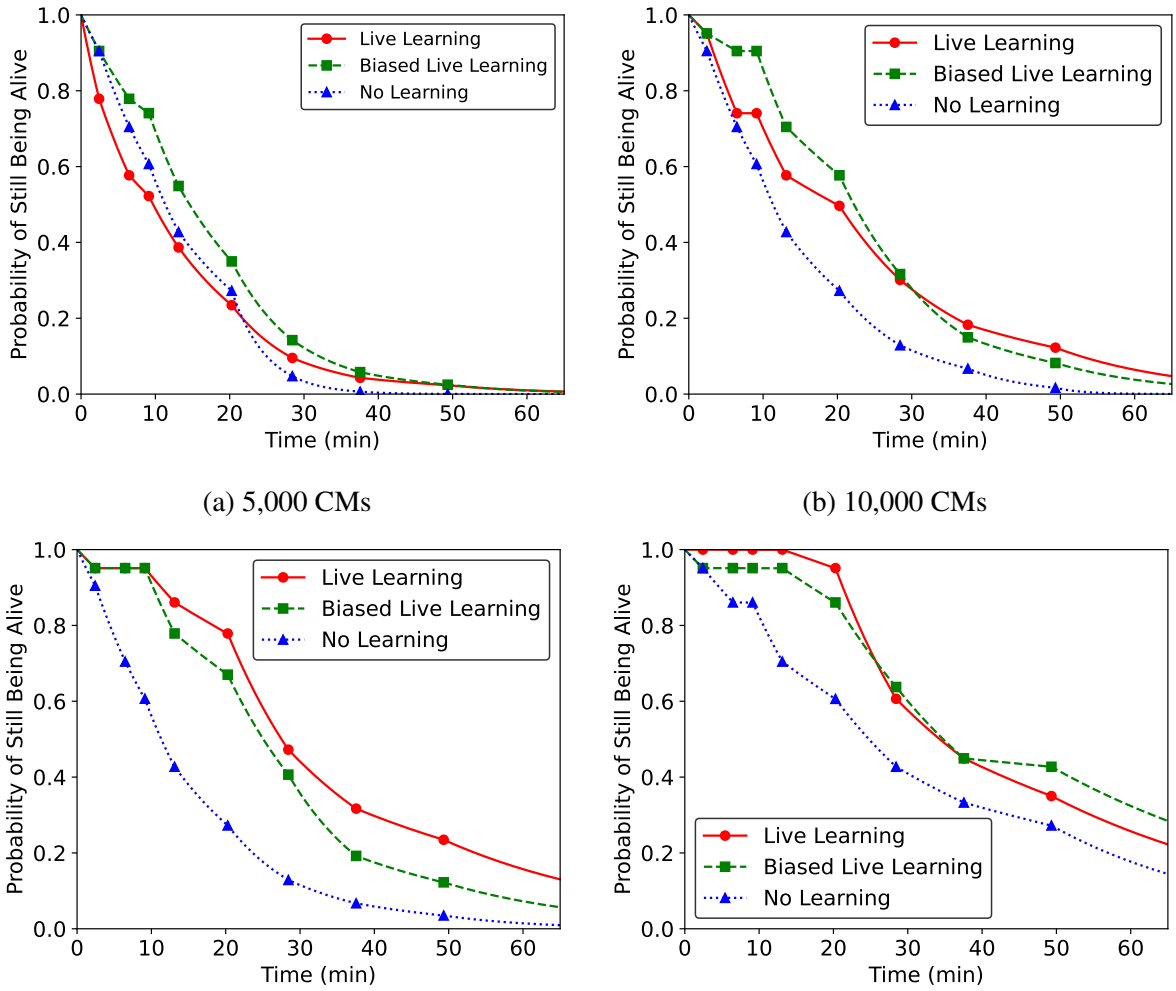
These results correspond to class Roundabout for dataset DOTA. For each data point, the survivability shown is for its optimal threshold (i.e., at which survivability is highest). Note that the X axis is in log scale.

Figure 4.22: Impact of Loss Function Bias on Survivability

within a targeted range, at what thresholds should the models operate? How many CMs do they use at those thresholds?" At very few FNs (0–1, 12–13), Figure 4.21(b) shows that both biased and unbiased models use roughly the same number of CMs; however, their optimal thresholds are quite different. For intermediate ranges of FNs (25–26 and 40-42), the biased models use significantly fewer CMs at optimal threshold. At the high end (74–76 FNs), the situation is reversed: the unbiased model uses noticeably fewer CMs. Thus, the choice of the loss function should be informed by the desired target FN range.

The impact on survivability of a biased loss function is shown by Figure 4.22. Note that the X axis is in log scale in these graphs. For each data point on these scatter plots, the survivability shown is for the optimal threshold. This is the threshold at which survivability is highest for the respective number of CMs and lethality. The benefit of bias is most apparent at a lethality of $\gamma = 0.10$ (Figure 4.22(d)). For all CM values, the survivability with a biased loss function is equal to or higher than that attained using an unbiased loss function.

At the lower lethality of $\gamma = 0.05$ (Figure 4.22(c)), the benefit of a biased loss function is still



(a) 5,000 CMs

(b) 10,000 CMs

(c) 20,000 CMs

(d) 50,000 CMs

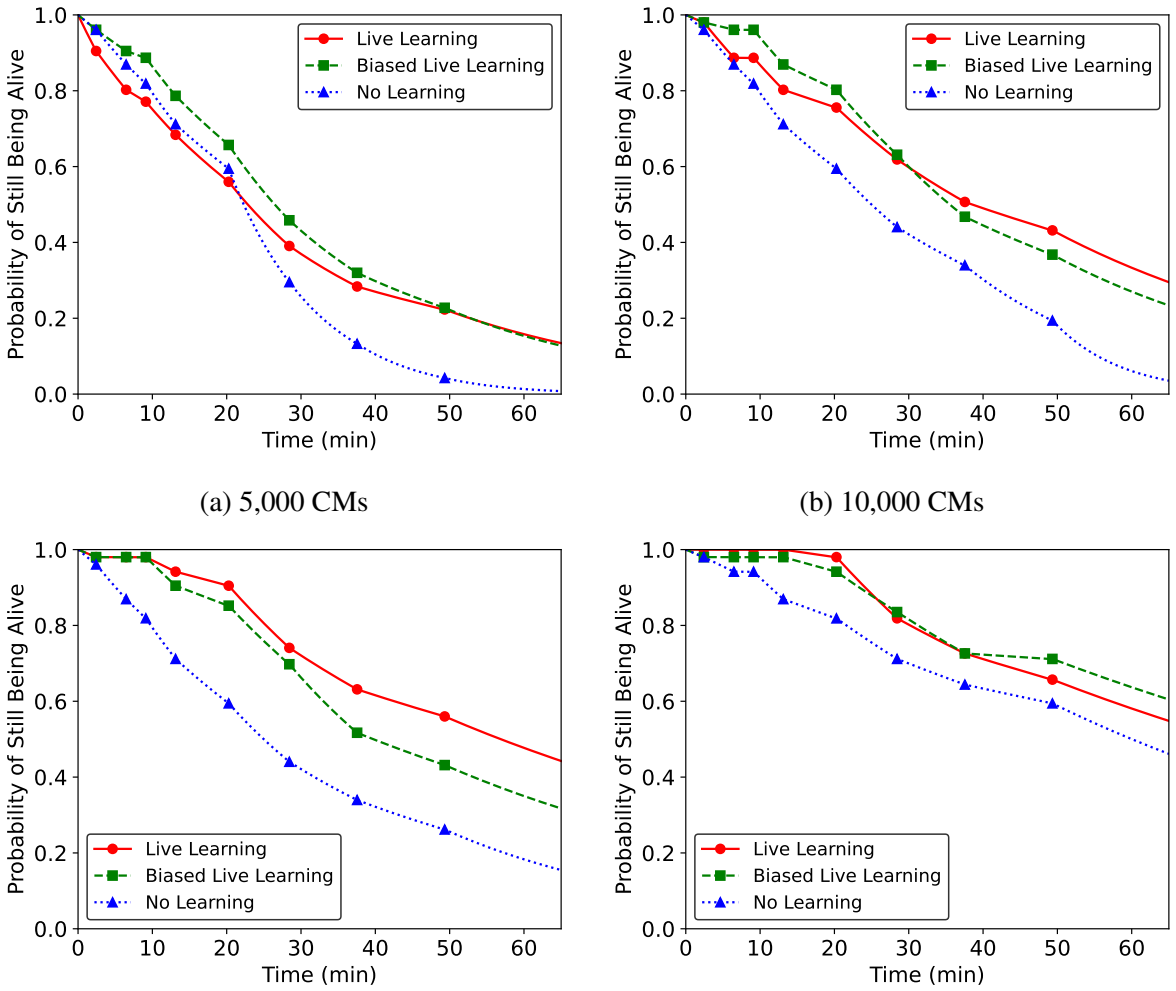
These results correspond to class Roundabout for dataset DOTA, and are obtained from the experiment for Figure 4.22(c).

Figure 4.23: Survivability at Optimal Threshold for $\gamma = 0.05$

visible. As before, the biased loss function does as well or better than the unbiased loss function. The sole exception is for 20,000 CMs.

When lethality is further reduced (Figures 4.22(b) and 4.22(a)), there is no longer a clear win from a biased loss function. At these lower lethalities, the penalty for an FN is small. The win from biasing for fewer FNs is much less now, and is swamped by other attributes of the loss function.

Figure 4.23 drills deeper into the data from Figure 4.22(c). For four different points on the X axis of Figure 4.22(c) (i.e., different CM values), Figure 4.23(a) through 4.23(d) show the full survivability curve for biased and unbiased loss functions. The heights of the points in Figure 4.22(c) correspond to the final Y values at the end of a mission for the curves in Figure 4.23. For calibration, the "No Learning" curve is also shown. This corresponds to the initial model remaining unchanged throughout a mission. The importance of Live Learning



(a) 5,000 CMs

(b) 10,000 CMs

(c) 20,000 CMs

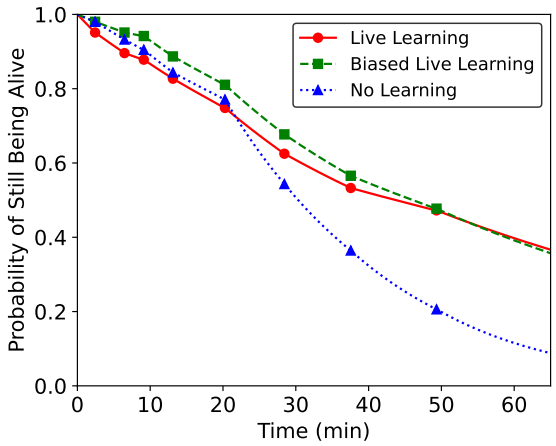
(d) 50,000 CMs

These results correspond to class Roundabout for dataset DOTA, and are obtained from the experiment for Figure 4.22(b).

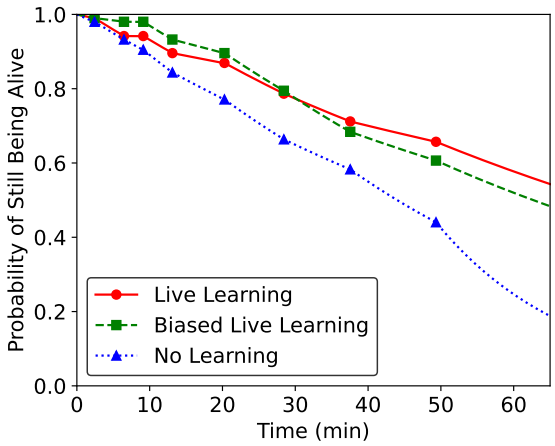
Figure 4.24: Survivability at Optimal Threshold for $\gamma = 0.02$

(biased or unbiased) relative to No Learning is clearly seen in Figure 4.23. The message from Figure 4.23 reinforces the message from Figure 4.22(c): at higher CM usage (50,000 CMs), the biased loss function improves survivability.

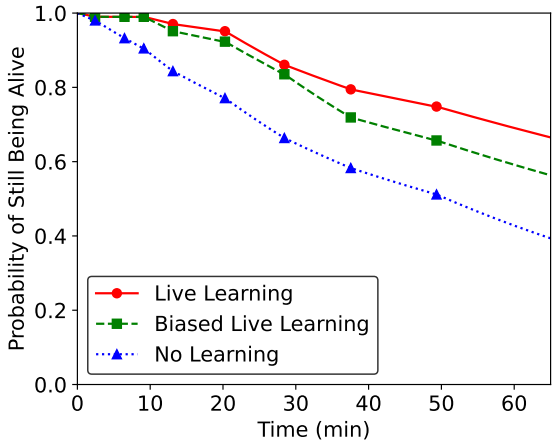
For lethalities of $\gamma = 0.02$ and $\gamma = 0.01$, Figures 4.24 and 4.25 enable in-depth comparison of biased, unbiased and No Learning cases. As expected, No Learning is always inferior to Live Learning. Between biased and unbiased flavors of Live Learning, there is not a clear winner at lower lethality. As mentioned previously (§4.6), score diffusion has an impact where the No Learning survivability is occasionally superior to that of Live Learning. This is not the case at our targeted mission duration, but early on when some of the outliers of ground truth positive threat samples score well below the respective threshold. However, because the mean of all threat samples increases as Live Learning continues its model improvements, its likelihood of mission success improves with respect to that of No Learning missions.



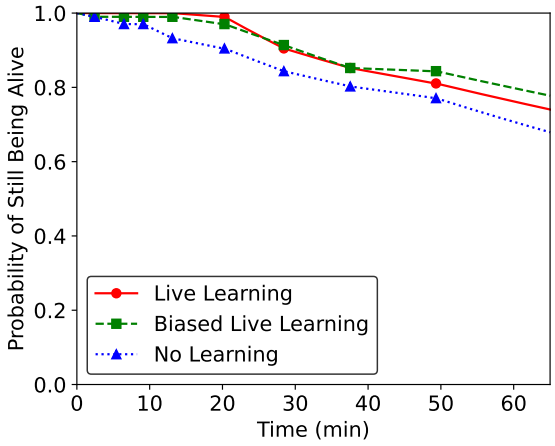
(a) 5,000 CMs



(b) 10,000 CMs



(c) 20,000 CMs



(d) 50,000 CMs

These results correspond to class Roundabout for dataset DOTA, and are obtained from the experiment for Figure 4.22(a).

Figure 4.25: Survivability at Optimal Threshold for $\gamma = 0.01$

4.8 Variable Mission Success Criteria - M of N Scouts

4.8.1 Analytical Description

The analytical model introduced in Chapter 3 defined the probability of mission success of a given duration to be the probability that all individual scouts, regardless of the total number, survived the mission. In other words, the question we were trying to answer was: If the required mission duration is T, what is the probability that *all* N scouts did not encounter a lethal threat until at least time T (all scouts survived)? Under this definition of survival, the average threat arrival rate was aggregated across all scouts where each scout inferenced approximately the same number of potentially lethal threat samples. We defined this rate as α.

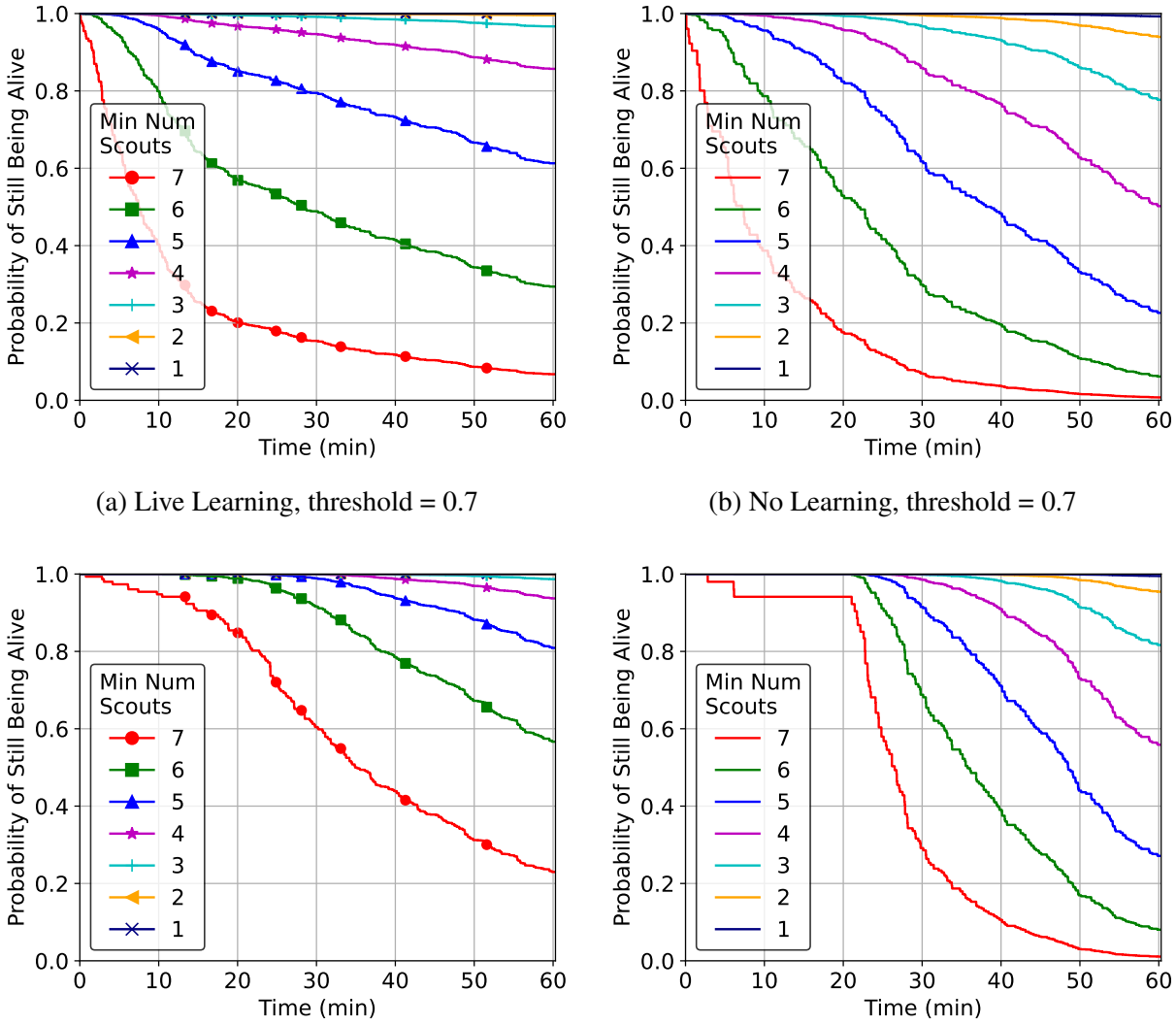
For more realistic survivability analysis, we must treat each scout as an individual SCML system. Thus, rather than simply modeling the probability that a single lethal threat destroys the overall system (meaning all scouts), we can now model the probability that a single lethal threat only destroys a single scout. By extension, we can then derive the probability of a given number of scouts are still alive at any instant in time during the mission. For a mission comprised of N scouts, the new threat arrival rate for each individual scout is α/N. In order to appropriately modify Equation 3.12 from Section §3.1.4, $\lambda_i = \alpha(1-\beta_i)\gamma$ simply becomes $\lambda_i = (\alpha/N)(1-\beta_i)\gamma$. Our goal here is to analyze and evaluate the impact on survivability of a new definition of mission success, ***M of N Survival Criterion***: Of N deployed scouts, what is the probability that at least M scouts survive at least until the mission duration (time T)?

The probability of survival of an individual scout at any instant in time is unique to that of any other scout in experimental scenarios. Therefore we model this new survival criterion with the Poisson Binomial Distribution [81] in equations 4.1 and 4.2. This distribution is a generalized version of the standard binomial distribution, which requires that probabilities of success for all independent Bernoulli trials are equal, e.g. a sequence of coin flips. Here we treat each individual scout's probability of survival as a single Bernoulli trial. For example, scout 1 after 10 minutes may have a survival probability of 0.7 while scout 2 may have a survival probability of 0.5. These values are derived from parameters pertinent to SCML operation (CM deployment threshold, sequence of sample scores, etc.) on each scout where one scout may encounter more threats than the other in a given time interval. Each scout will ultimately have a unique survivability curve in such experimental trials due to the randomness of threat sample interarrival times. Equation 4.1 defines the probability that *any* M of the N total scouts are still alive at time T (the end of the mission). M is any number in the range $[1, N]$.

$$P_T(s = M) = \sum_{I \in C(N,M)} \prod_{i \in I} p_T^i \prod_{j \notin I} (1 - p_T^j) \tag{4.1}$$

$$P_T(s \geq M) = \sum_{k=M}^{N} \sum_{I \in C(N,k)} \prod_{i \in I} p_T^i \prod_{j \notin I} (1 - p_T^j) \tag{4.2}$$

The case where $M = N$ simply means that all scouts must survive the entire mission for it to be considered a success. In this equation, $C(N, M)$ represents the set of all combinations of choosing M scouts from N total scouts. p_T^i represents the probability that scout i (which is one of the chosen scouts in the combination I) is alive at time T and (1 - p_T^j) represents the



(a) Live Learning, threshold = 0.7

(b) No Learning, threshold = 0.7

(c) Live Learning, threshold = 0.55

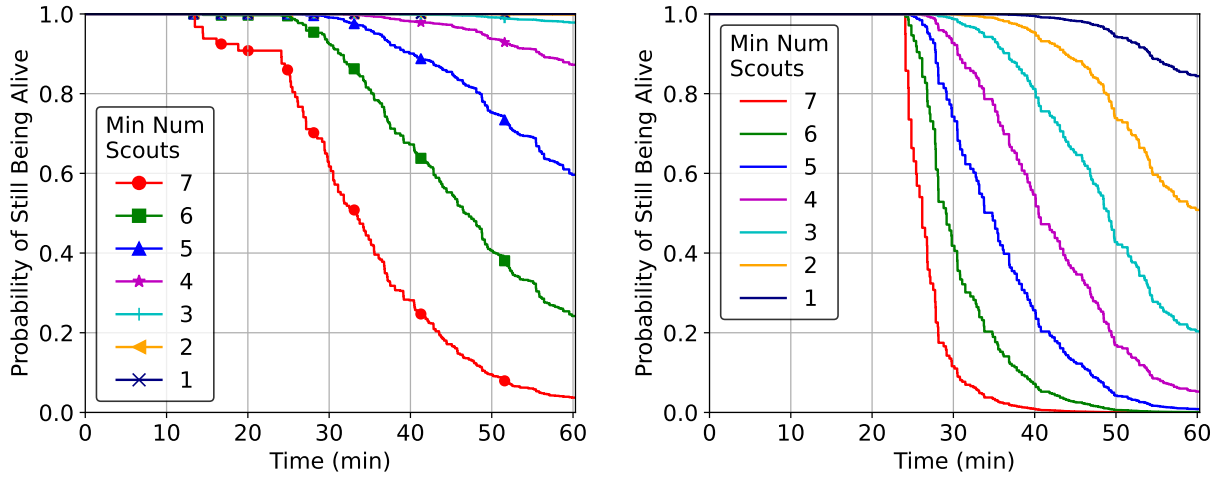
(d) No Learning, threshold = 0.55

All graphs above assume a threat lethality of $\gamma = 0.02$ and 50k CMs. These results correspond to Live Learning and No Learning missions of class Swimming Pool for the DOTA dataset.

Figure 4.26: Value of Live Learning for Survivability, $\gamma = 2\%$.

probability that scout j (which is one of the scouts not chosen in the combination I) is not alive at time T. The final result is the aggregate probability, across all possible combinations of M scouts, that M scouts are alive at time T. Because this equation provides us such a result for all values of M, we can extend this result to *at least M of N*. Suppose no less than M out of N scouts must reach the destination for a mission of duration T to be successful. By combining multiple instances of Equation 4.1, we can derive Equation 4.2, which gives the probability of mission success that at least M out of N scouts are alive at time T.

We assess the impact of the relaxation of our survival criterion by running a Live Learning mission and, instead of computing the survivability curves across all scouts in aggregate, we



(a) Live Learning

(b) No Learning

All graphs above assume a threat lethality of $\gamma = 0.05$ and 100k CMs with a threshold of 0.4. These results correspond to Live Learning and No Learning missions of class Swimming Pool for the DOTA dataset.

Figure 4.27: Value of Live Learning for Survivability, $\gamma = 5\%$

compute the survivability for each individual scout. This allows us to compute the probability of survival for exactly M scouts across all possible combinations of M scouts. We can then derive the final result: the probability that at least M out of N scouts survive until time T.

4.8.2 Experimental Results: Survivability as a Function of M Scouts

We have described in this section how to evaluate SCML system performance when the survival criterion is relaxed and flexible. Initially, the simplest case was that all individual scouts must survive until the end of the mission for it be considered successful. In the real world, especially in hostile environments, a task or mission can be completed even if 100% of all assets or agents don't survive. This concept is known as redundancy. A key overarching theme in the following results is: *If we need to guarantee a probability that M scouts survive an entire mission, we must send N scouts.* Figure 4.26 shows the difference in value that Live Learning provides for survivability for two different thresholds: 0.7 and 0.55, with 50k available countermeasures. Each individual plot shows the probability that at least M scouts are alive at time t. Figure 4.26(a) shows the Live Learning survivability gains over No Learning in Figure 4.26(b). If we have strict survivability requirements ($M = 7$), the probability of mission success is increased from nearly 0% to about 8%. If we relax the requirement to $M = 6$ scouts, the probability of mission success increases from approximately 8% to 30%. This trend follows as the requirements are relaxed, but the gains decrease as Live Learning provides less value when $M < 4$. This is because when M is small, there are many more possible survival combinations (groups of scouts) than when M is closer to N. The 0.7 threshold also ensures that the scouts never deplete their countermeasures as there is no obvious countermeasure depletion cliff.

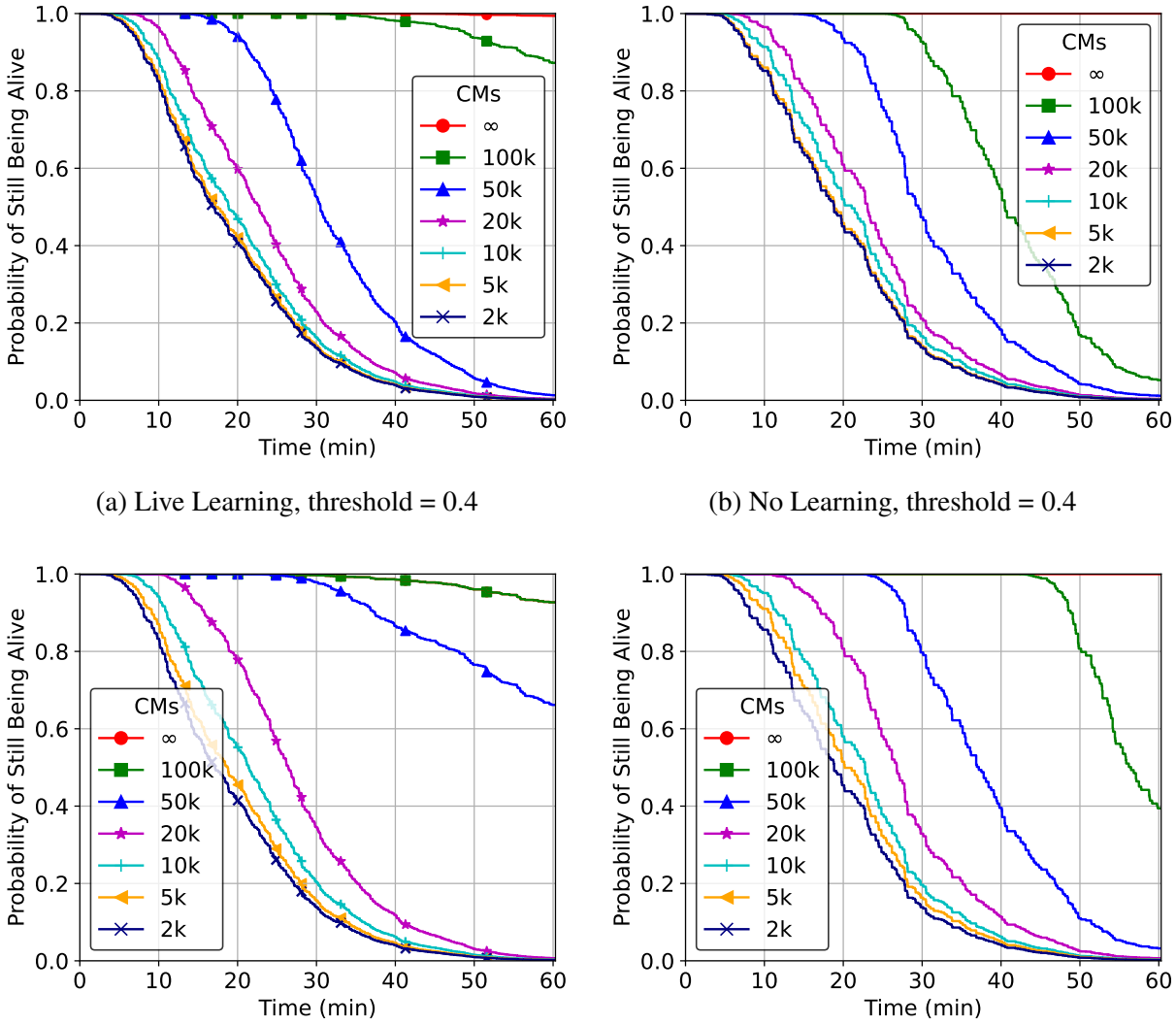
In Figures 4.26(c) and 4.26(d), we reduce the threshold to 0.55. In both plots, but especially in that of No Learning, there is a clear CM depletion cliff at about 20 minutes into the mission. This causes the threat arrival rate to increase as both FNs and TPs are now potentially lethal. Live Learning mitigates the effect of this cliff by its ability to reduce inference scores of ground truth negatives over time. We clearly observe the improvment in survivability in the Live Learning curves over No Learning. For $M = 7$, we see an increase from 1% to 24% and for $M = 6$, survivability increases from 8% to 57%. As with a threshold of 0.7, further reducing M limits the value of Live Learning over No Learning as the survival criterion is less challenging. Comparing the top two and bottom two pairs of figures, we notice that a lower threshold of 0.55 allows for Live Learning to improve the probability of mission success by reducing ground truth negative inference scores, enabling greater efficiency with countermeasure deployment.

Increasing the number of available countermeasures but reducing threshold enables Live Learning to have an even more pronounced effect on survivability. Curves for $M = 4, 5,$ and 6 in Figure 4.27(b) have a probability of mission success close to 0%. However, the equivalent curves in Figure 4.27(a) have a probability of success of approximately 87%, 59%, and 24%, respectively. This is because the lower threshold of 0.4 reduces FNs overall, yet depletes CMs at about 24 minutes into the mission. In other words, this combination of parameters reduces potentially lethal threats early in the mission, but renders the scouts more vulnerable later in the mission, where the No Learning curves decay rapidly.

4.8.3 Experimental Results: Survivability as a Function of Countermeasures

Relaxed survival criterion of $M \geq 4$ and $\gamma = 5\%$ is shown in Figure 4.28. Figures 4.28(a) and 4.28(b) on top show the impact of available countermeasures with a threshold of 0.4. Clearly, at 50k CMs and below, there is virtually no improvement by Live Learning. This is because the curves with 50k CMs and below all deplete their CMs prior to 20 minutes and some even between 5 and 10 minutes into the mission. However, if scouts have double the CMs to survive past 20 minutes (green curve), they improve their survivability from about 5% to 87%, purely by the fact that Live Learning is able to reduce the score of ground truth negatives and improve its CM efficiency.

Increasing the threshold to 0.55 in Figures 4.28(c) and 4.28(d) noticeably improve the survivability of both the 50k and 100k curves. The 50k curve is able to reduce its CM deployment rate due to the higher threshold and thus can improve the probability of mission success from 4% to 67% with Live Learning. Reducing the survival criterion even further to $M \geq 2$ in Figure 4.29 yields interesting results, where the threshold is set to 0.7. We notice that most of the high countermeasure curves are not visible. This is because they are all drawn underneath the 20k and 10k curves, which implies that large number of CMs are not valuable at a higher threshold. All of the No Learning curves in Figure 4.29(b) are within 15-40% survivability. Live Learning in Figure 4.29(a) is able to increase the probability of mission success significantly for all but the smallest number of CMs (2k). The key takeaway here is that if the survival criterion is sufficiently relaxed, there is a higher likelihood of mission success at lower numbers of available CMs.



(a) Live Learning, threshold = 0.4

(b) No Learning, threshold = 0.4

(c) Live Learning, threshold = 0.55

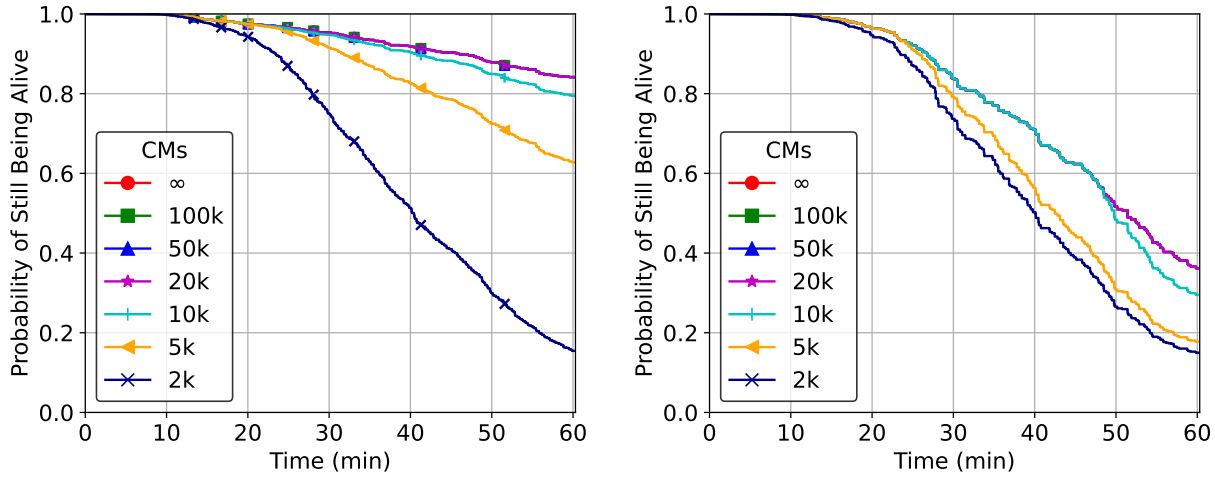
(d) No Learning, threshold = 0.55

All graphs above assume a threat lethality of $\gamma = 0.05$ and $M \geq 4$ scouts. These results correspond to Live Learning and No Learning missions of class Swimming Pool for the DOTA dataset.

Figure 4.28: Value of Live Learning over No Learning for Survivability, $\gamma = 5\%$.

4.8.4 Experimental Results: Survivability as a Function of Threshold

At the most restrictive survival criteria ($M \geq 6, 7$), Figure 4.30 depicts the benefits of Live Learning at low lethality (1%) and plentiful CMs (100k). We observe in Figure 4.30(a) that the two lowest thresholds (green and red curves) begin to rapidly decay around 30 minutes, due to CM depletion. This also holds true for the same curves in Figure 4.30(b), however, they are not visible because the blue curve is drawn over the top of them. Thresholds 0.4 and 0.55 both significantly improve their probabilities of success with Live Learning, due to decreased ground truth negative scores, resulting in efficient CM deployment. The 0.7 improves its performance in



(a) Live Learning

(b) No Learning

All graphs above assume a threat lethality of $\gamma = 0.05$, $M \geq 2$ and a threshold of 0.7. These results correspond to Live Learning and No Learning missions of class Swimming Pool for the DOTA dataset.

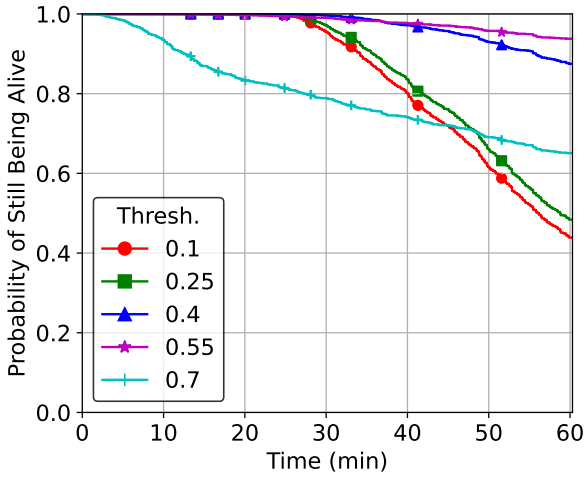
Figure 4.29: Effect of Live Learning for Further Relaxed Survival Criterion ($M \geq 2$), $\gamma = 5\%$

Figure 4.30(a) only due to increased scores of potentially lethal threats, reducing the frequency of FNs.

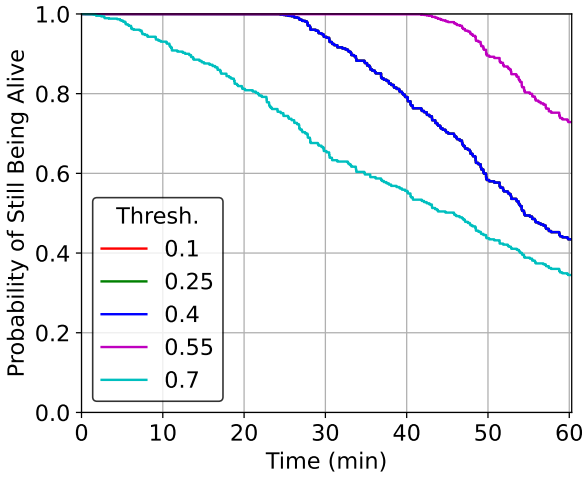
When $M \geq 7$, missions across all thresholds have reductions in probabilities of success. As with $M \geq 6$, the bottom three thresholds have identical survivability curves with No Learning in Figure 4.30(d). With Live Learning in Figure 4.30(c), the 0.1 and 0.25 threshold curves remain largely unchanged but the 0.4 threshold curve increases from about 13% to 49%. This is because Live Learning is able to pull many ground truth negative scores down below 0.4 but not quite below 0.25 and 0.1. The No Learn figure depicts a higher probability of success than Live Learning until about 45 minutes for threshold 0.55. This is because in No Learning ground truth positive scores still remain high, resulting in minimal FNs, yet it has a steep CM depletion cliff at 42 minutes. Conversely, some Live Learning ground truth positive scores may decrease with Live Learning early in the mission, but the positive effect is the significant reduction in CM deployment, allowing for an increased probability of survival from 33% to 64%.

We now compare survivability across the middle range of M values. Figure 4.31 compares probabilities of success for $M = 3, 4, 5, 6$ with 50k CMs and $\gamma = 5\%$. In all four plots, the red and green curves (thresholds of 0.1 and 0.25) are hidden under the blue curves (threshold of 0.4) as very few, if any, ground truth positives score below 0.4, leading to little to no FNs. It is possible that two scouts, for example, experienced an FN early in the mission, however, but $M \geq 3$ means that five or more scouts would have to encounter a potentially lethal threat within a certain time interval for even a single drop in survivability to be noticeable in the figure.

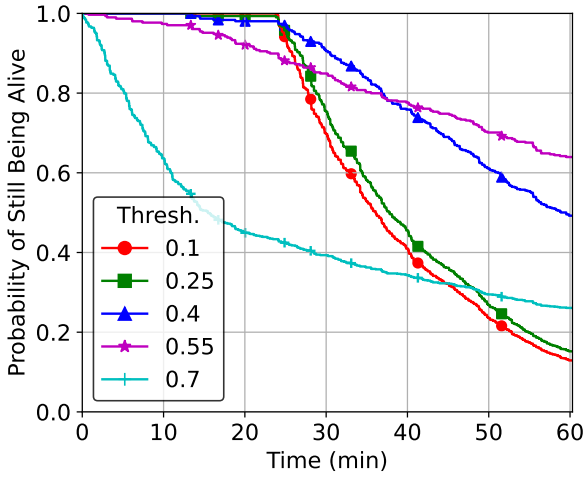
These four figures demonstrate the impact of threshold selection on optimal survival criterion at a given number of available CMs. Starting with $M \geq 3$ in Figure 4.31(a), the most relaxed criterion, we observe that the 0.4 and below thresholds deplete their CMs at approximately 20



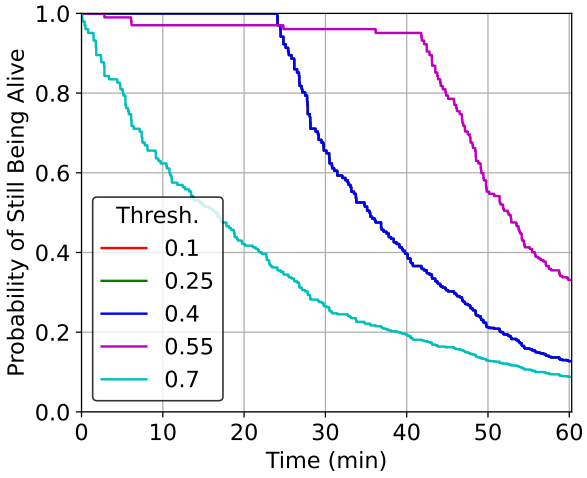
(a) Live Learning, $M \geq 6$



(b) No Learning, $M \geq 6$



(c) Live Learning, $M \geq 7$

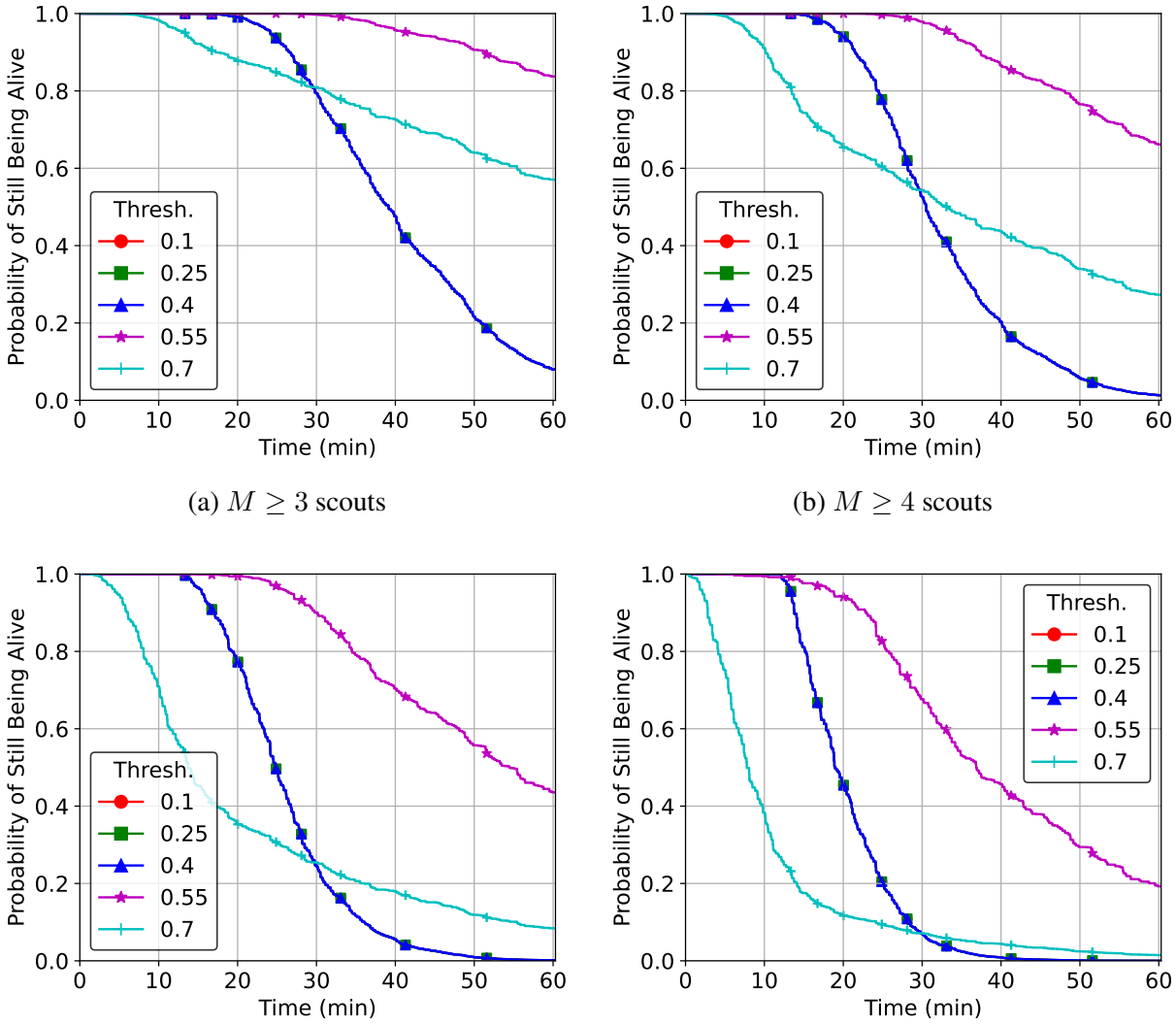


(d) No Learning, $M \geq 7$

All graphs above assume a threat lethality of $\gamma = 0.01$, 100k CMs, $M \geq 6$ and $M \geq 7$ scouts. These results correspond to Live Learning and No Learning missions of class Swimming Pool for the DOTA dataset.

Figure 4.30: Value of Live Learning over No Learning w/ Strict Survival Criterion, $\gamma = 1\%$

minutes into the mission and the 0.55 threshold curve at about 30 minutes. The 0.7 threshold never depletes its CMs as all of its potentially lethal threats come as a result of FNs. With $M \geq 4$ scouts in Figure 4.31(b), probabilities of mission success reduce the 0.4 threshold and below curves from about 8% down to 1%. The 0.55 curve's likelihood of success was reduced from 84% to 67%, and the 0.7 curve from 57% to 27%. Similarly for $M \geq 5$ in Figure 4.31(c), the 0.55 threshold curve has a lower chance of success at 44% and the 0.7 curve an 8% chance. Finally, for $M \geq 6$ in Figure 4.31(d), the 0.55 curve now has only a 20% chance of success while the 0.7 curve has a 2% chance of success. The main theme here is that the lowest overall



(a) $M \geq 3$ scouts

(b) $M \geq 4$ scouts

(c) $M \geq 5$ scouts

(d) $M \geq 6$ scouts

All graphs above assume a threat lethality of $\gamma = 0.05$ and 50k CMs. These results correspond to Live Learning missions of class Swimming Pool for the DOTA dataset.

Figure 4.31: Survivability by Threshold and M Scouts Required for Mission Success

performing thresholds (0.1, 0.25, and 0.4) and next lowest performing threshold (0.7) have their greatest reductions in performance from $M \geq 3 \rightarrow M \geq 4$. The highest performing threshold (0.55) has the greatest drop in performance from $M \geq 5 \rightarrow M \geq 6$.

For the given number of available CMs, there is clearly an optimal threshold for a Live Learning mission as only the more restrictive survival criterion cause greatest degradations in survivability. The survivability of suboptimal thresholds in this scenario decrease the greatest under the more relaxed survival criterion. Intuitively, different thresholds may be optimal for various scenarios, but this analysis allows us to determine how robust a given threshold is to increasingly restrictive survival criterion. Even if a given threshold performs optimally when

$M \geq 1$, it may not necessarily be robust to increasingly restrictive criterion like $M \geq 6$ or $M = 7$. Therefore, if such criterion needs to change dynamically during the mission it would be wise to select the threshold that maintains a reasonable (or required) probability of mission success as M increases.

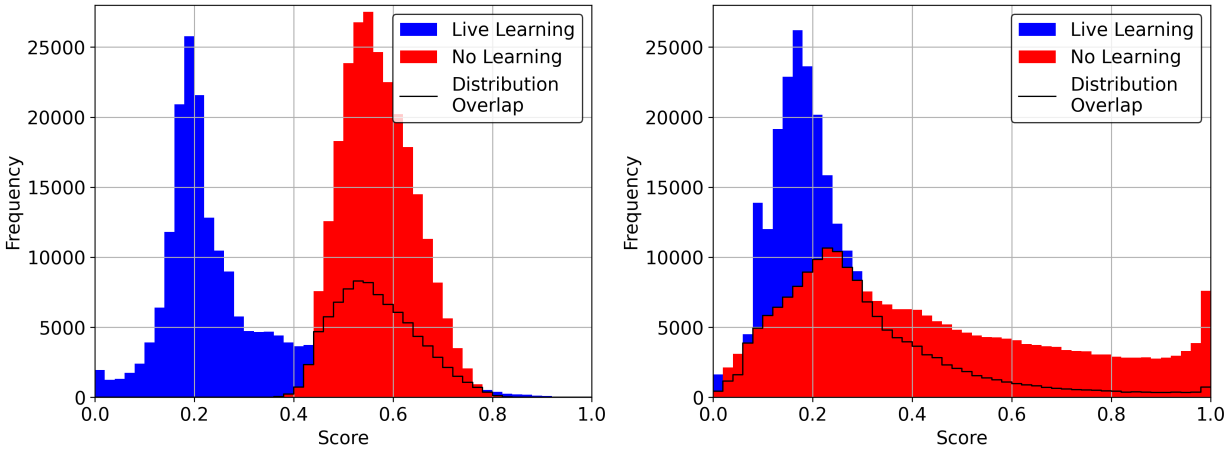
4.9 Divergent Objectives: Live Learning vs. SCML Metrics

Although SCML and Live Learning complement each other, they can operate as completely independent systems. The objective of an SCML system is to maximize the probability of mission success by minimizing the probability of succumbing to lethal threats. Live Learning's objective is to find the maximum number of rare target (threat) samples from an extremely imbalanced high data rate stream under limited network bandwidth and labeling budgets. Their Key Performance Indicators (KPIs), the metrics by which their performance is measured, diverge slightly such that the value Live Learning provides does not necessarily translate completely to SCML. In short, Live Learning enhances performance by improving the relative score distribution between positive and negative samples. Conversely, SCML performance is determined by a sequence of absolute sample scores in their comparisons to a specific threshold value (whether constant or variable). This section provides detailed analysis and results on the effects of this nuanced difference.

Live Learning with Hawk was originally designed and validated to iteratively improve its ability to collect samples of a rare novel target class from streaming, real-time data. In order to accomplish this, models inference incoming samples and prioritize them in a queue on the scout according to the likelihood of belonging to the class of interest. At some rate dependent on network and human labeling throughput, the highest scoring samples are sent to home for labeling, which are then used for future retraining. At any point in the mission, the task of the current model is to assign each sample an inference score that dictates its place in the priority queue and its chance of being transmitted for labeling. Effectively, the model simply "sorts" the inbound sample stream as accurately as possible. Samples at the head of the queue are selected for transmission when bandwidth permits. Therefore, as long as positive samples are close to or at the head of queue, they will be sent to home.

In Hawk, collecting the greatest number of positive samples is enabled by high positive scores *relative* to negative scores while the reverse impedes it. For instance, if most negative sample scores are below 0.25 and several positive sample scores are around 0.3, the positive samples are guaranteed to be transmitted. However, if many negative samples are currently in the queue with scores around 0.9, positives will need to have scores above that level to have any chance of transmission. Simply put, the absolute scores of the positives are not meaningful in Hawk, rather it is their relative position in the queue compared to negative samples.

SCML also benefits, though indirectly, from generally increased positive sample scores and generally decreased negative scores. The former reduces FNs while the latter conserves CMs by minimizing FPs. In our experimental model (Chapter 4), we described how the inference score of each sample is compared directly to the current countermeasure deployment threshold to determine the outcome. Whether the threshold remains constant or is adaptive during a mission, this is an *absolute* comparison. This simple logical operation determines whether a threat goes



(a) Dataset: DOTA, Class: Swimming Pool

(b) Dataset: DOTA, Class: Roundabout

All graphs above represent results corresponding to Live Learning and No Learning missions of classes Swimming Pool and Roundabout for the DOTA dataset.

Figure 4.32: Shift in Negative Score Distribution from No Learning to Live Learning

undetected (an FN whose score is below the threshold) and can be potentially lethal, or if it must be neutralized with a countermeasure (an FP or TP that scores above the threshold). The decision happens at the moment right after inference to minimize delay. The score of any particular sample is never compared to scores of other samples for the immediate SCML decision, and therefore the mere improvement in the positive to negative score distribution is not completely indicative of improvements in SCML performance.

We have seen in previous results (§4.6) that in specific intervals during the mission, Live Learning is actually counterproductive for survivability compared to No Learning. This is because of the iterative and stochastic nature of model retraining where parameter updates from the current to the new model may not always result in increased scores of ground truth positive samples. It is entirely possible that in a mission with a configured threshold of 0.4, Model 0 (in a No Learning mission) inferences a ground truth positive sample with a score of 0.45, resulting in a TP and neutralized with a CM. Similarly, Model 1 (in a Live Learning mission) may yield an inference score of 0.35 (for the same sample), resulting in an FN. While this does not happen frequently, it occurs often enough that it is noticeable in certain circumstances. Previous sections in this dissertation that provide standard SCML results in the form of survivability curves comparing Live Learning performance with that of No Learning have occasionally exhibited this behavior. This section provides deeper statistical analysis of this phenomenon, which should further clarify the previous explanation of the difference between default SCML and Live Learning metrics: the significance of absolute versus relative score distribution.

We analyze existing log data from missions performing both No Learning and Live Learning for classes Swimming Pool and Roundabout from the DOTA dataset. Figure 4.32 depicts the shift of the distribution scores from negative samples in both types of missions. We observe in Figure 4.32(a) that the No Learning mission for class Swimming Pool yields a mean inference score of 0.57, as shown in Figure 4.33. With Live Learning, the mean is reduced to 0.34 with

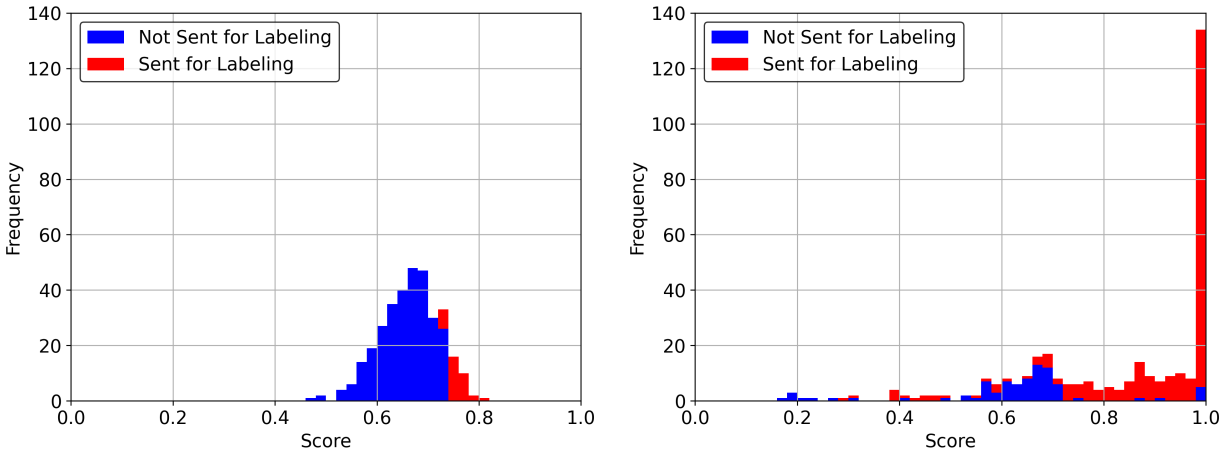
Class	Swimming Pool				Roundabout			
	Negatives		Positives		Negatives		Positives	
Mean/St Dev	μ	σ	μ	σ	μ	σ	μ	σ
No Learning	0.57	0.07	0.66	0.06	0.44	0.27	0.91	0.17
Live Learning	0.34	0.19	0.83	0.20	0.24	0.16	0.82	0.27
	Raw #	%	Raw #	%	Raw #	%	Raw #	%
Decreased Scores	206841	82	45	13	184309	73	64	21
Increased Scores	45055	18	290	87	67583	27	245	79

Figure 4.33: Relevant Statistics for Data Visualized in Figures 4.32 and 4.34.

very few samples scoring above 0.8, as can be seen in the bottom right corner of the figure. Live Learning was able to reduce the inference scores of 82% of negative samples from No Learning. We also notice that Live Learning produces a bimodal distribution of negative samples, with one mode remaining in close proximity to the original mean of No Learning and one centered around 0.2. This is a class-specific phenomenon in that Live Learning is able to reduce the mean of the primary mode to about 0.2 but the secondary mode has a mean of around 0.55. To translate into Live Learning terms, Hawk is able to able to recognize and differentiate easy negatives (negatives with lower scores) from hard negatives (negatives with higher scores), at least for class Swimming Pool. Figure 4.32(b) demonstrates a different effect where the negative samples in the Roundabout mission represent a single mode both in No Learning and Live Learning. No Learning has a much wider distribution (with a mean of 0.44 and standard deviation of 0.27) whereas Live Learning shifts the mean to the left (0.24) and reduces the standard deviation to 0.16. Live Learning is also able to reduce the number of samples scoring above 0.8 to a minimum, enabling many more positive samples to be sent to home for labeling. As will be reinforced in the next figures, Hawk's ability to find as many rare positive samples in a sea of data in part relies upon its ability to reduce the inference scores of the greatest number of negative samples to the greatest degree possible. For class Roundabout, Live Learning was able to reduce the inferences scores of 73% of all negative samples from No Learning, as shown in Figure 4.33.

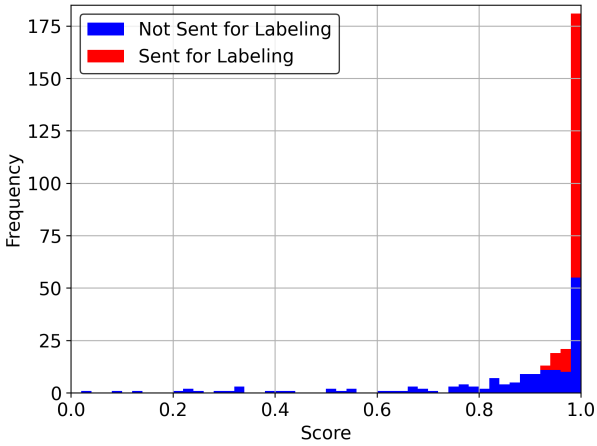
The other core task of Live Learning with Hawk is to increase the inference scores of as many positive samples to the greatest possible degree. The combination of this effect and the reduction in negative inference scores results in more efficient transmission where rare positives are more likely to be transmitted for labeing than negatives. Figure 4.34 depicts the distribution shift for positives from No Learning to Live Learning for the corresponding classes of dataset DOTA. Figure 4.34(a) shows the positive sample score distribution for No Learning of class Swimming Pool, with a mean of 0.66 (from Figure 4.33). We assigned a color to each sample according to whether or not they are ultimately sent to home. With No Learning, when the model never changes, the positives samples sent to home are on the right edge of the distribution closest to one.

Figure 4.34(b) represents the positive score distribution for Live Learning, with a mean of 0.83. It is clear that many sample scores are increased as close to one as possible (with 87% of all positive samples yielding increased inference scores from No Learning). On the other hand,

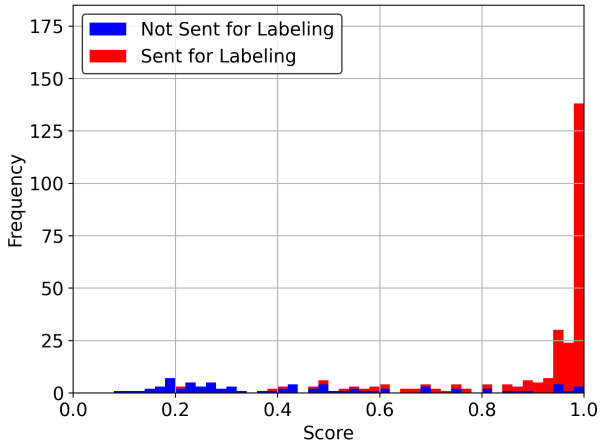


(a) No Learning, Class: Swimming Pool

(b) Live Learning, Class: Swimming Pool



(c) No Learning, Class: Roundabout



(d) Live Learning, Class: Roundabout

Figure 4.34: Positive Score Distribution for Classes Roundabout and Pool of the Dataset DOTA.

a small fraction of positives have reduced inference scores (13%). This is due to the underlying retraining process of Live Learning. Generally, many more negative samples are available for retraining than positives, even though positives are shared across all scouts after being labeled at home. Athough the mean increased from 0.66 to 0.83, any one of the 13% of all positive samples that resulted in reduced inference scores could increase the number of FNs in an SCML mission, depending on the threshold.

Positives of the Roundabout class in a No Learning mission, shown in Figure 4.34(c), depict the vast majority of scores between 0.8 and 1.0, many of which are never sent to the labeler, with an overall mean of 0.91, shown in Figure 4.33. This complements the effect in Figure 4.32(b) where the No Learning distribution contained a relatively large number of negative samples also with scores between 0.8 and 1.0 as compared to Live Learning. The practical effect is that high-scoring negative samples consume more network and labeling bandwidth than do similarly scoring positives sample due to extreme class imbalance.

Live Learning for the Roundabout class in Figure 4.34(d) shows that high-scoring positive

samples sent for labeling are largely unimpeded from being sent to home for labeling, with an overall mean of 0.82. The vast majority of samples that are not sent to home are below 0.8 and even less than 0.2 in a few extreme cases. Clearly, some "hard positives" exist within the overall set of positives containing features for which the model has difficulty learning. As previously stated, because Live Learning periodically trains new models, one of the side effects includes score diffusion of a portion of the positive samples, resulting in a subset of samples being reduced in score. Live Learning caused a reduction in positive sample mean as the No Learning mean was already rather close to one and because of the hard positive effect, where some positive scores were reduced significantly, having a disproportionate affect on mean reduction. Ultimately, even though the mean dropped by 0.09, Live Learning was still able to increase 79% of all positive sample scores. In our experiments, Hawk is only able to collect approximately between 200-300 positive samples for retraining during a typical mission, which are far outnumbered by the number of available negative samples for training. During the mission, earlier training iterations may only have access to anywhere between 20 and 100 positive samples for training, limiting feature diversity trained into such early, weaker models.

The analysis in this section demonstrates that Live Learning may not be able to perfectly decrease inference scores of all negative samples or increase that of all positive samples. The degree to which it can perform either depends to some extent on the nature of the class itself. The implications for SCML are twofold. First, the increase in positive scores generally decreases the number of FNs during an SCML mission. However, this is only true if the threshold is at a level that would actually make a difference in changing the inference result. For instance, if the threshold was 0.7 and the score for a sample was 0.4 in a No Learning mission then increased to 0.6 in a Live Learning mission, Live Learning would make no impact on converting an FN to a TP. If the threshold were 0.5, Live Learning would have converted this inference result from an FN to a TP, thereby increasing survivability (requiring a CM to neutralize and under the asumption CMs have not yet been depleted). Second, the decrease in negative scores decreases the number of FPs during an SCML mission. Again, the current threshold has a substantial impact on the degree to which FPs are reduced. If the threshold is 0.3, for example, and a negative sample inference score is reduced from 0.9 to 0.5, then the FP is not converted to a TN and a CM will still be wasted on it. If the threshold were 0.6, then the FP would be converted to a TN by Live Learning and the CM would not be wasted. The aggregate number of score reductions and resulting conversions from FPs to TNs greatly increases the amount of time until CMs are depleted. The resulting effect is that the amount of time during a mission that TPs (in addition to FNs) are potentially lethal threats is significantly reduced, ultimately increasing the likelihood of mission success.

For these particular classes, we observed that the Pool class performed better in increasing positive scores (87% of total) compared to the Roundabout class (79% of total), the latter of which had a reduction in mean positive score. This means that Live Learning for the Pool class would likely have a much greater effect on decreasing the total number of FNs during an SCML mission. Live Learning for the Pool class only reduced 13% of all positive scores compared to the Roundabout class which reduced 21% of all positive scores.

We also saw that Live Learning for the Roundabout class was able to reduce the inference scores of 73% of all negative samples with 82% for the Pool class. However, the Roundabout class had a reduced mean of 0.24 while Pool had a reduced mean of 0.34. Therefore, it is likely

that Live Learning for the Roundabout class will perform better at reducing the total number of FPs during an SCML mission. As stated previously, reducing the rate of FPs delays CM depletion, which ultimately limits the total number of potentially lethal threats encountered by the scouts.

To restate, the Live Learning/SCML duality discussed here involves the nuanced difference in the technical metrics by which both systems are evaluated. The relative score distribution in Live Learning with Hawk dictates the rate and quality of improved model generation, indirectly affecting survivability. The absolute score distribution in any generic SCML system directly dictates the individual classification result for each encountered sample. This results in a code-pendent relationship in which individual processes, Live Learning and SCML, each with their own unique metrics, affect the performance of the other and thus survivability in general.

4.10 Adaptive Thresholding

Our experimental results (§4.6) showed that the CM deployment threshold has a significant impact on system survivability. The threshold, along with the individual inference score, dictate the classification result: FP, TP, FN, or TN. If the threshold is too high, the frequency of FNs increases and the system will fail to detect potentially lethal threats, resulting in poor recall. The advantage of a higher threshold is that CMs are deployed more conservatively, increasing the probability that the system will not exhaust them prior to the end of the mission, leading to greater long-term survivability. On the other hand, a lower threshold reduces the number of FNs, yet is more likely to exhaust all CMs prior to the end of the mission. We also saw that CM exhaustion leads to a sudden increase in the number of potential threats that include TPs in addition to FNs. Considering all of these tradeoffs, we determined that there is a need for a resource-driven adaptive thresholding algorithm to improve the performance of SCML systems. In other words, we need an intelligent method for threshold adjustment as a function of CM depletion over time. This section introduces a resource-driven method to adaptive thresholding that dynamically reacts to the state of the scout's finite resources (CMs).

Resource-focused system optimization and adaptation has been studied prior to the surge in autonomous systems research. Flinn and Satyanarayanan investigated how the combination of energy-aware operating systems and applications can prolong battery life while maintaining application quality [82]. More recently, researchers have recognized the need for energy-efficient autonomous system operation to maximize general endurance [83, 84]. Path planning and its many adjacent areas have been robust topics of research in robotics. For instance, energy-aware path planning for swarms has received more recent attention [85, 86] as well as that for aerial drones more broadly [87, 88]. Still most research around energy-conscious path planning focuses on simple ground operation [89–92]. These studies, among others, generally have the same broad objective as that in our work, which is to be as efficient and intelligent with the use of finite resources as possible, in the pursuit of some further goal (survivability in our case). We introduce our unique approach that applies to the SCML use case in §4.10.1.

When defining any approach to resource-driven adaptive thresholding, we had to consider three key characteristics of a Live Learning-enabled SCML mission. First, although we expect in most cases the threat interarrival time to be exponentially distributed, it can be bursty, as with

network traffic. Moreover, we assume the system has no a priori knowledge of the sample score distribution. This means the system must have the ability to surge CM deployment when required and conserve CMs when able. Second, with Live Learning we expect model improvement in the form of generally increasing ground truth positive threat scores and decreasing ground truth negative scores. Therefore, we expect the SCML system will need to deploy CMs at a greater rate early in the mission with weaker inferencing models but at a lower rate with more accurate models later in the mission. Third, we recognize the need to limit the ability to support bursty CM deployment later in the mission. If an SCML system needs to deploy many CMs early in the mission, it has the luxury of time to balance it with a compensatory lower CM deployment rate later in the mission. However, if CM deployment is allowed to spike late in the mission to a similar degree as early in the mission, it will likely lead to premature CM exhaustion, negatively affecting survivability. As a result, we must ensure that there is a reduced chance of excessive CM deployment toward the end of the mission than at the start.

4.10.1 Convergent Proportionate Setpoint Control for Countermeasure Deployment

In this section, we describe the details of our approach to dynamically adapt the CM deployment threshold with the purpose of controlling the CM deployment rate. The CM deployment rate at sample i, R_i, is defined in Equation 4.3:

$$R_i = \frac{c_i/C}{i/|\mathcal{Z}|} \tag{4.3}$$

where c_i/C is the percent of total CMs used and $i/|\mathcal{Z}|$ is the percent of samples inferenced thus far in the mission. By controlling this rate, for example by remaining close to $R = 100\%$, we ensure that the SCML system is neither too frugal nor too wasteful in CM deployment, ideally depleting them at the end of the mission. This not only allows CMs to remain available for the whole mission, preventing the system from becoming vulnerable to TPs, but also to prevent mission completion with a large number of excess CMs. The goal is to keep the threshold sufficiently high such that CMs are not depleted early, yet sufficiently low to ensure all available CMs are deployed. In order to accomplish this, the system must monitor the rate at which CMs are deployed and intelligently adjust the threshold accordingly. If CMs are deployed too rapidly, the threshold must increase at the expense of potential additional FNs. Likewise, if the CM deployment rate is on track to complete the mission with a large excess of CMs, the threshold should decrease in order to increase the CM deployment rate and reduce the probability of FNs.

To implement adaptive thresholding, we use a combination of multiple methods, resulting in a hybrid method of Convergent Proportional Setpoint Control with Saturation. The purpose of this approach is to minimize wide swings in threshold but also allow for it to react to increasing or decreasing trends in the CM deployment rate with minimal delay. Setpoint control [93] is a widely used technique in control theory and systems. It sets a target for a process value or essential variable [94] in various control systems. The common example is a heating and air conditioning system that has a set desired temperature. In this example, it is undesirable for

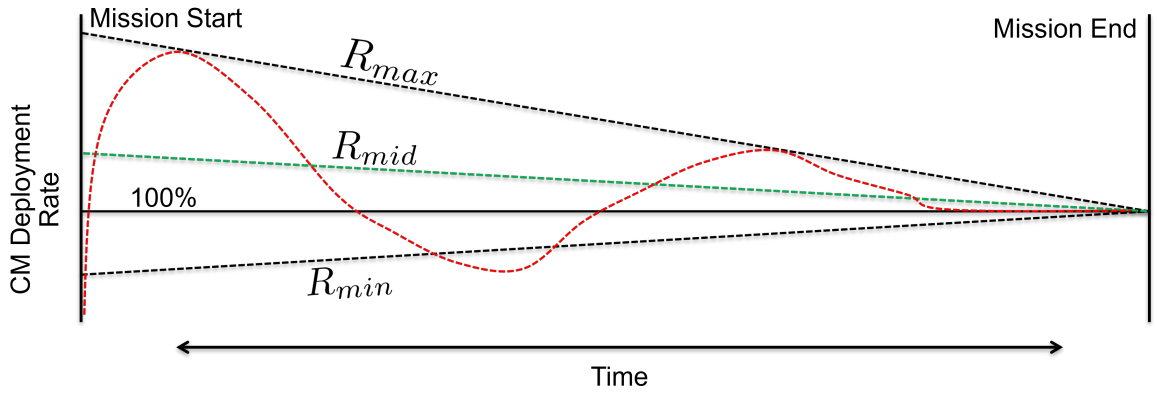


Figure 4.35: Example Convergent CM Deployment Rate

the heater to activate every time the temperature drops an insignificant amount. Therefore, a deadband or hysteresis approach may be appropriate to prevent a physical system from having to constantly activate and deactivate. Proportional Integral Derivative (PID) Controllers [95] perform similar functions in common applications like vehicle cruise control. Other work incorporates setpoint control for energy management in electric vehicle [96]. Other researchers have leveraged various forms of time-varying setpoints in their specific domain studies [97, 98].

In SCML, where the CM deployment threshold is simply adjusted in software, there is no concern about unstable system cycling. Thus, setpoint control is the optimal approach for adaptive thresholding. Proportional setpoint control refers to the linear relationship between the distance between the setpoint and the process value (current CM deployment rate) and the corrective adjustment made to the threshold. In our approach, the current CM deployment rate is the process value which may need to be adjusted at every time step (sample inference). The setpoint in an SCML system is the target CM deployment rate, for example 95%, 100%, 105%, etc. In Figure 4.35, the setpoint is represented by R_{mid}, which is simply the mean of R_{max} and R_{min}. R_i is the process value, the current CM deployment rate at sample i. Saturation refers to a hard upper limit (R_{max}) to the CM deployment rate, which, when reached the SCML system can no longer deploy CMs. Once the CM deployment rate drops below this limit, the system can begin deploying CMs again. The lower limit, R_{min}, is meant to act as a hard lower limit (if needed), but for our specific implementation it is only used to compute the setpoint, R_{mid}. This method is convergent in that the initial upper and lower limits are configured such that that setpoint converges to a CM deployment rate of 100% at the end of the mission. Converging the setpoint to 100% in this manner ensures that the threshold adjustments are dampened toward the end of the mission to avoid early CM exhaustion or hoarding. The novelty in this overall approach is the combination of saturation, convergence, and proportionality for setpoint control.

Consider an example mission with adaptive thresholding in Figure 4.35 where R_{max} starts well above 100% and over time converges to 100% by the end of the mission. Similarly, R_{min} starts below 100% and slowly converges to 100% by the end of the mission. From these two boundaries, we can compute R_{mid}, which is simply the mean of the two boundaries. The current value of R_{mid} (the setpoint) in part determines whether the threshold is increased or decreased after each inference. The red dotted line represents a toy example of the iteratively computed CM deployment rate (from Equation 4.3). If the computed deployment rate is above the setpoint,

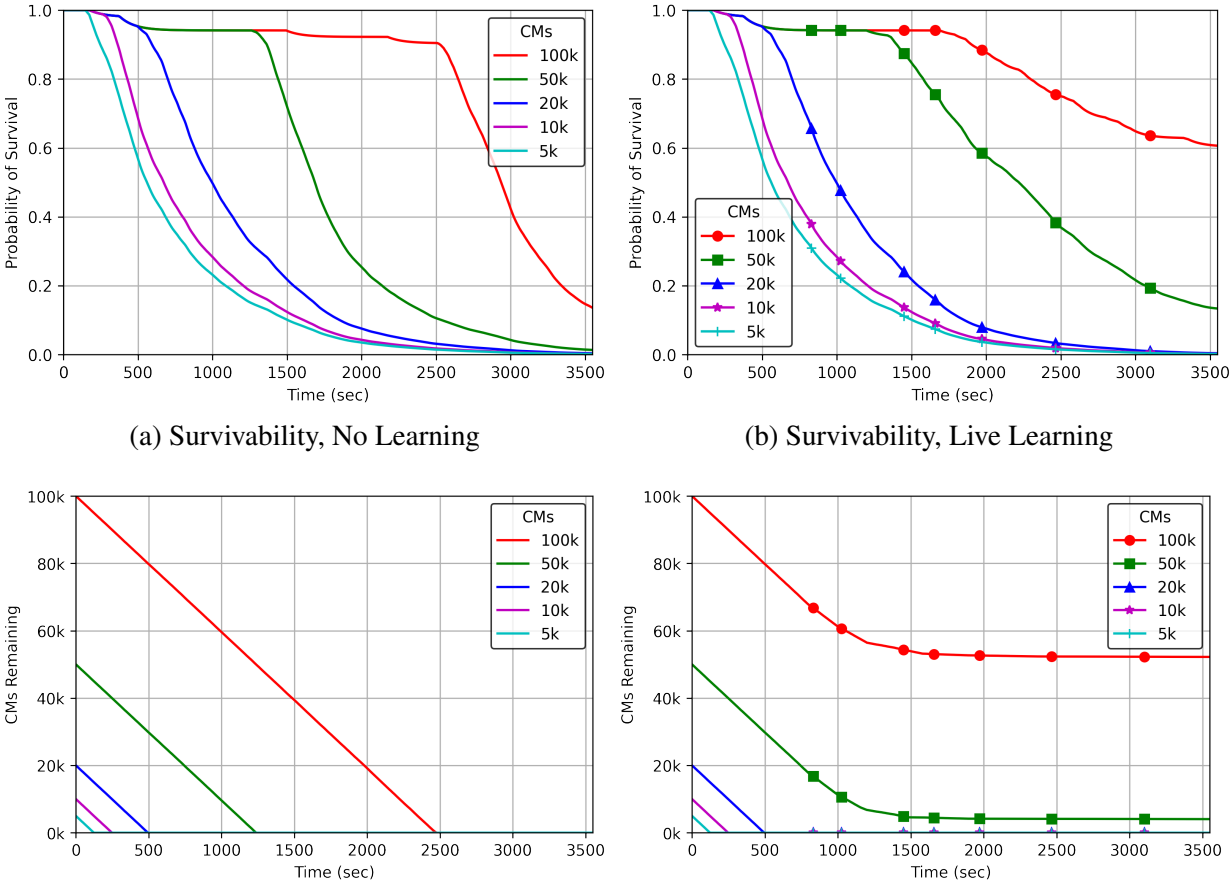
Algorithm 1 Convergent Proportional Setpoint CM Deployment Control w/ Saturation

Require: $\mathcal{Z}$: set of all samples of incoming stream, $R_{max,1}$: CM deployment rate initial upper bound, $R_{min,1}$: CM deployment rate initial lower bound, α: weighting factor, C: total CMs available

1: $R_{mid,1} \leftarrow \frac{R_{max,1}+R_{min,1}}{2}$ // Setpoint CM depl. rate prior to inferencing sample 1
2: $c_1 \leftarrow 0$ // Number of total CMs deployed prior to inferencing sample 1
3: $R_1 \leftarrow 0$ // CM deployment rate prior to inferencing sample 1
4: **for** $i \leftarrow 1$ **to** $|\mathcal{Z}|$ **do**
5: **if** $z_i \geq \theta_i$ **then**
6: **if** $c_i < C$ **then**
7: **if** $R_i \leq R_{max}$ **then**
8: $c_{i+1} = c_i + 1$ // Deploy CM
9: **end if**
10: **end if**
11: **end if**
12: $R_{i+1} = \frac{c_{i+1}/C}{i/|\mathcal{Z}|}$ // Recompute CM utilization
13: $R_{mid,i+1} \leftarrow \frac{R_{max,i+1}+R_{min,i+1}}{2}$ // Setpoint CM depl. rate prior to inferencing sample i
14: $\Delta_{i+1} \leftarrow R_{i+1} - R_{mid,i}$ // Optimal CM deployment rate gap
15: $\theta_{i+1} = (1-\alpha)(\theta_i) + \alpha(\Delta_{i+1})$ // Compute threshold for next sample
16: **end for**

then the threshold will be increased to reduce the deployment rate. If the computed deployment rate is below the target rate, the threshold will be decreased to increase the deployment rate. One can carefully select the initial R_{max} and R_{min} values to influence the R_{mid} (setpoint) values. A wider boundary interval (greater R_{max} and smaller R_{min}) results in greater CM deployment rate flexibility early in the mission while boundaries just above and below 100% will lead to a more strict target deployment rate throughout the mission. Recall that because R_{max} is a hard upper bound (the saturation point), the system has an incentive to not reduce the threshold to such an extent where CM deployment is repeatedly throttled (as it increases the chances TPs will not be neutralized even though CMs are still available).

The process by which threshold computation occurs is defined in Algorithm 1. The inputs are the total number of available CMs, the maximum CM deployment rate, the minimum rate, the exponentially weighted moving average factor, and an incoming stream of data samples (the scout receives one sample at a time and does not have access to them a priori). First, the required components are initialized prior to inferencing the first sample. For each inference sample, simple logic determines whether the score is at least the current threshold on line 5. On line 6, it then checks whether there is at least one CM available and on line 7 it checks whether the current deployment rate is not greater than the maximum rate. If it is greater, no CM is deployed due to throttling. If it is not greater, a CM is deployed and the CM deployment counter is incremented on line 8. Once the CM deployment decision has been made, the CM deployment rate for the next inference is recomputed on line 12. On line 13 the setpoint rate is recomputed from the next maximum and minimum rates. The threshold change is computed as the simple difference



(a) Survivability, No Learning

(b) Survivability, Live Learning

(c) Remaining CMs, No Learning

(d) Remaining CMs, Live Learning

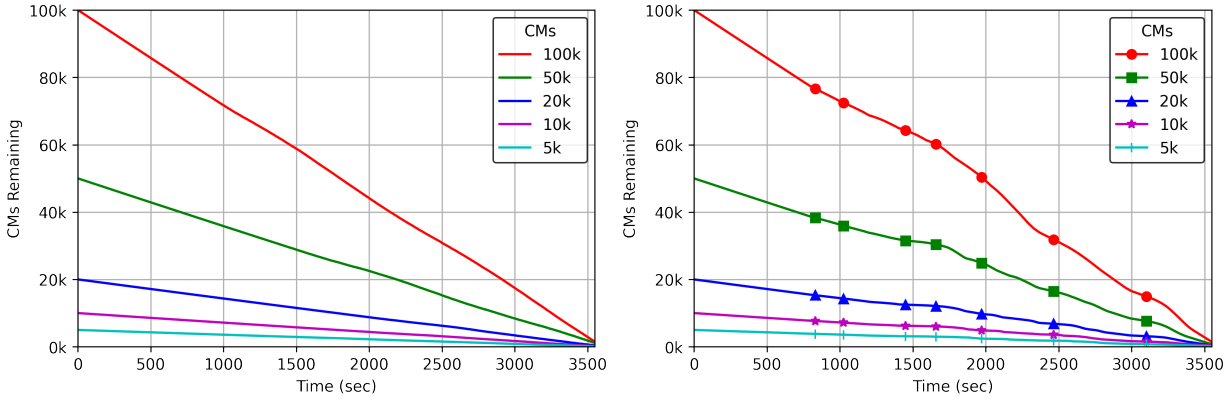
These results correspond to a mission of class Swimming Pool for the DOTA dataset, with γ = 0.02 and constant threshold = 0.55.

Figure 4.36: Survivability and Remaining CMs with Constant Threshold

between the new deployment rate and the next setpoint rate on line 14. On line 15, the new threshold is computed using an exponentially weighted moving average of the current threshold and the threshold change. The primary takeaways are: 1) Whether the threshold is increased or decreased depends on whether the recomputed CM deployment rate is less than or greater than the setpoint rate (which can change over time according to Figure 4.35), and 2) CMs deployment is throttled if the current deployment rate exceeds the maximum rate, potentially causing unneutralized TPs.

4.10.2 Experimental Results

In our initial experimental results (§4.6) we demonstrated the effect of CM deployment threshold on survivability. In this section, we first analyze the rate of CM depletion for a constant threshold, and then compare it to results with our adaptive thresholding algorithm. Figure 4.36 shows curves for survivability and remaining CMs for both No Learning and Live Learning missions.



(a) Remaining CMs, No Learning

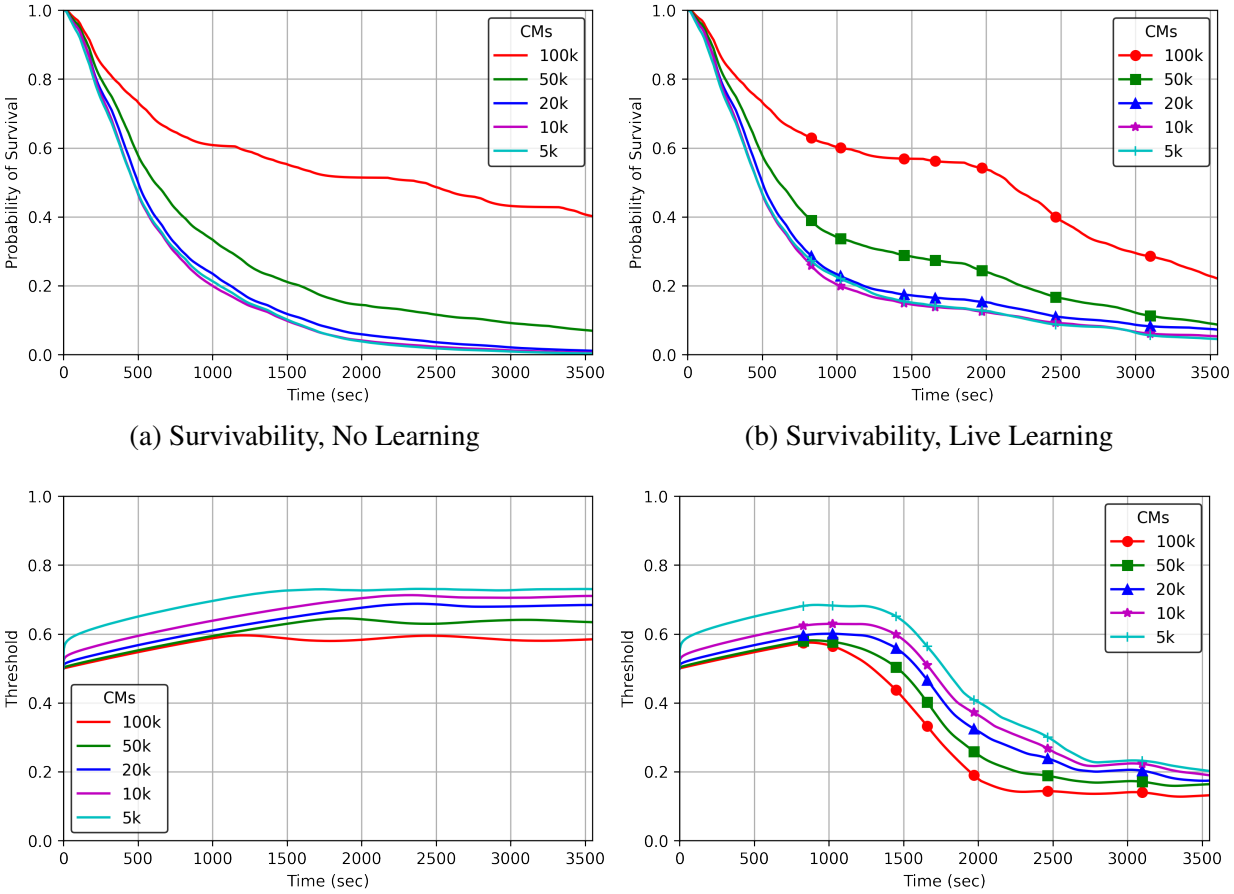
(b) Remaining CMs, Live Learning

These results correspond to a mission of class Swimming Pool for the DOTA dataset, with γ = 0.02 and a variable threshold with $R_{max} = 103\%$ and $R_{min} = 98\%$.

Figure 4.37: Remaining CMs with Variable Threshold $(103/98)$

Figure 4.36(b) shows the survivability improvement of Live Learning over No Learning (Figure 4.36(a)). At a threshold of 0.55 and with minimal CMs available, Live Learning fails to provide any value with such severe resource constraints. However, for 50k and 100k CMs, there are clear gains in survivability. We observe in Figure 4.36(c) that 100k CMs are depleted by approximately 2500 seconds into the mission. This CM depletion cliff is clearly visible in Figure 4.36(a), where the survivability curve decreases sharply. Similarly, 50k CMs are depleted by roughly 1200 seconds, which also corresponds to the visible CM depletion cliff at the same time in the respective survivability curve. The effect of Live Learning on CM deployment is clear in Figure 4.36(d). Although CMs of 20k, 10k, and 5k are completely exhausted prior to the first available improved model, 100k and 50k CMs provide sufficient time to reach the first retraining. Once the fourth new model has been retrained, almost no CMs are deployed for the rest of the mission. This is because the Live Learning process has adequately retrained each successive model such that the fourth and all subsequent models score the vast majority of all ground truth negative samples below the constant threshold of 0.55, preventing CMs being wasted on FPs. Even if most ground truth positives are scored above 0.55 as TPs, due to extreme class imbalance only a tiny number of CMs are deployed to neutralize them. Visualizing the remaining CMs as a function of time, Figure 4.36(d) clearly depicts the value of Live Learning in preventing CM exhaustion (at some levels of initial CMs) by sufficiently reducing ground truth negative inference scores.

The primary practical effect of our adaptive thresholding algorithm is the relatively stable depletion of CMs over time during the mission, shown in Figure 4.37. A No Learning mission uses a single model, and therefore the distribution of negative scores does not noticeably shift over time, as shown in Figure 4.37(a). Additionally, given a relatively narrow initial CM deployment rate interval between 98% and 103%, this results in remaining CMs having a smooth linear dependence on mission time. With Live Learning in Figure 4.37(b), each new model clearly perturbes the rate at which CMs are deployed. Regardless of the number of total CMs, each curve still shows that CMs are exhausted right at the end of the mission. Contrary to Figure 4.36(d) with a constant threshold, with adaptive thresholding and Live Learning, CMs are deployed at



(a) Survivability, No Learning

(b) Survivability, Live Learning

(c) Threshold, No Learning

(d) Threshold, Live Learning

These results correspond to a mission of class Swimming Pool for the DOTA dataset, with γ = 0.02 and a variable threshold with $R_{max} = 103\%$ and $R_{min} = 98\%$.

Figure 4.38: Survivability and Variable Threshold $(103/98)$

a generally stable rate across the entire mission and thus are neither exhausted prematurely nor naively hoarded.

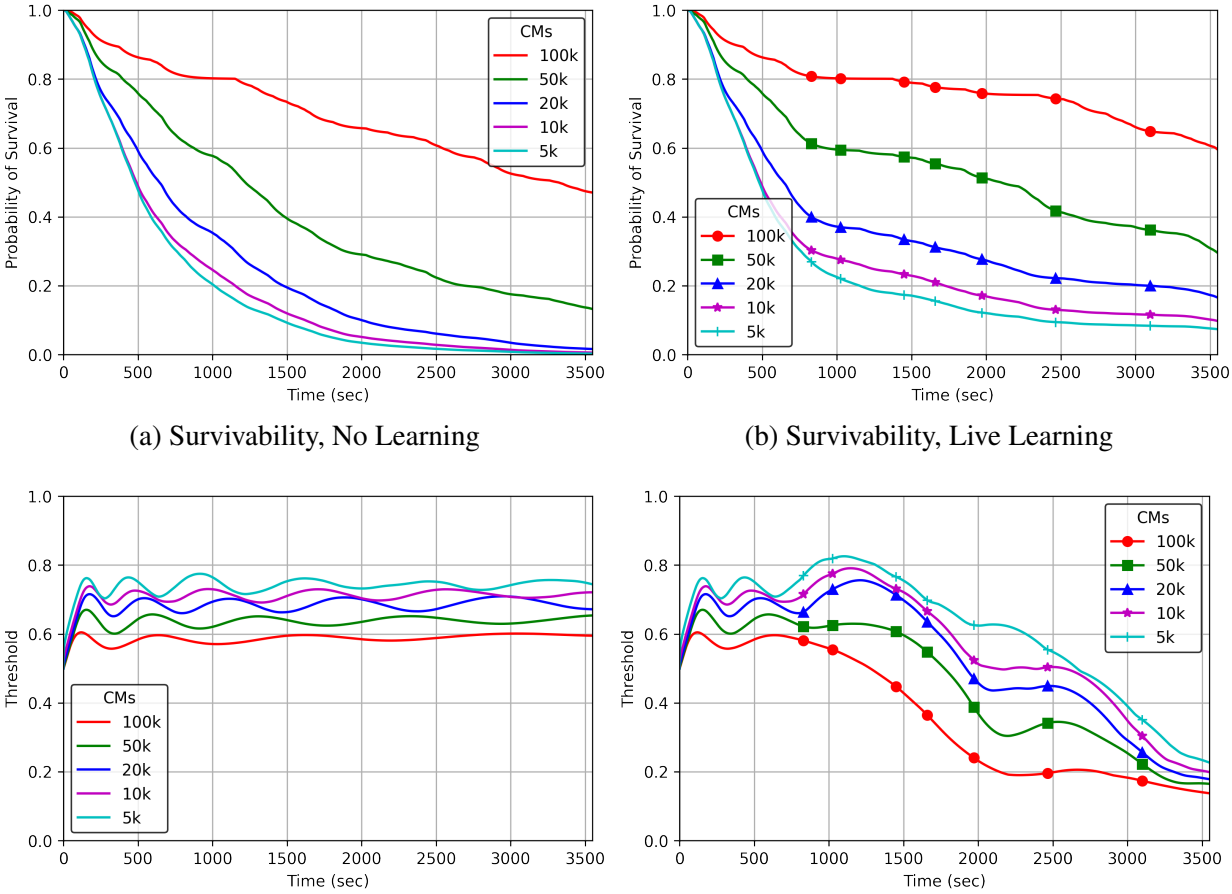
We previously discussed the impact of the tunable initial maximum and minimum CM deployment rates. Rates close to 100% allow for minimal flexibility in reacting to random bursts of FPs but result in more stable threshold adjustments. Additionally, if the CM deployment rate reaches the hard maximum limit, CM deployment is throttled, which can result in unneutralized TPs – even when CMs have not been exhausted. Figure 4.38 shows the effect of adaptive thresholding with strict deadband limits of $R_{max} = 103\%$ and $R_{min} = 98\%$. With scarce available CMs, Live Learning (Figure 4.38(b)) clearly improves survivability over No Learning (Figure 4.38(a)). Although the improvement is not large in absolute terms, an increase from near 0% to around 10% is significant. However, for 100k CMs, Live Learning survivability is reduced by almost 50%.

We observe in Figure 4.38(d) that all thresholds trend downward in the middle third of the mission, where the 100k CM line reaches 0.2 at about 2000 seconds and about 0.16 at 2500

seconds. It is clear that the 100k threshold line is significantly lower in Live Learning (Figure 4.38(d)) than in No Learning (Figure 4.38(c)) in the last two thirds of the mission. This means that the number of FNs experienced with Live Learning is generally lower than that for No Learning. Thus, Live Learning experiences more TPs than No Learning, which is ideal in scenarios where no CM throttling occurs. Recall that R_{max} converges to 100% toward the end of the mission and R_{mid} is simply the mean of R_{max} and R_{min}. Under an initially constrained R_{max} of 103%, toward the end of the mission it will be close to 100% and R_{mid} will be even closer to 100%. This results in constant CM throttling in which many TPs go unneutralized. Thus, because Live Learning encounters a much greater number of TPs compared to FNs, many of them go unneutralized because of throttling due to the strict R_{max} in this scenario. No Learning did not suffer from this effect as much because although it never trained new models, the loss in survivability from more FNs due to high threshold was far less than the loss in survivability due to CM throttling as $R_{max} \approx 100\%$ toward the end of the mission. This affected Live Learning to a much greater degree as its threshold reached 0.16. Also recall that the threshold is adjusted based on its distance from R_{mid}, which is virtually 100% toward the end of the mission, therefore it is unlikely it will change at all. The reason this only affects scenarios with large numbers of available CMs is because the adaptive thresholding algorithm ensures such plentiful CMs can be used to neutralize a greater number of TPs by minimizing threshold and by extension potentially lethal FNs. Scenarios with small number of CMs generally suffer relatively more FNs as compared to TPs as their thresholds must be kept higher to ensure scarce CMs are not exhausted prematurely, which ultimately minimizes the potential asymmetric throttling effect between No Learning (stable threshold) and Live Learning (decreasing threshold).

Figure 4.39 depicts improved survivability for Live Learning where $R_{max} = 150\%$. The results in these plots demonstrate the clear impact of a much greater maximum CM deployment rate along with a much lower minimum rate. As we saw previously in Figure 4.38, a strict upper limit on CM deployment results in frequent CM throttling, which can have a negative impact on survivability, even when samples are correctly classified as TPs. A greater upper bound allows for much wider swings in the threshold such that CM throttling is quite rare, and therefore unneutralized TPs are rare. Due to this effect, with any number of CMs, Live Learning improves survivability in Figure 4.39(b) over that for No Learning in Figure 4.39(a). The greatest percent increases in survivability are at minimal numbers of CMs (by approximately a factor of five). The curve with 50k CMs roughly doubles its survivability and 100k CMs improves from about 48% to 60%. This trend reinforces the value that resource-oriented adaptive thresholding provides: intelligent and efficient CM deployment for enhanced survivability, especially under extreme resource constraints. We also observe that there is no noticeable asymmetric throttling effect from the reduced threshold in Live Learning. This results in better improvements in survivability with Live Learning over No Learning with greater numbers of CMs.

Figure 4.39(c) depicts a greater variability in threshold early in the mission. This is due to the wider deployment rate boundaries of 150% and 75%, which allow the threshold to drift farther from the midpoint. Recall that we converge the R_{max} and R_{min} values to 100% by the end of the mission, resulting in reduced threshold variability over time. With Live Learning and wider maximum and minimum deployment rates (Figure 4.39(d)), similar threshold behavior occurs at the beginning of the mission. However, compared to the tightly bound threshold curves in Figure 4.38(d), there is wider variation in threshold across all CM levels as it decreases. Although



(a) Survivability, No Learning

(b) Survivability, Live Learning

(c) Threshold, No Learning

(d) Threshold, Live Learning

These results correspond to a mission of class Swimming Pool for the DOTA dataset, with γ = 0.02 and a variable threshold with $R_{max} = 150\%$ and $R_{min} = 75\%$.

Figure 4.39: Survivability and Variable Threshold $(150/75)$

both figures begin with different deployment rate boundaries, $(98, 103)$ compared to $(75, 150)$, thresholds in each figure converge to similar values. This is also true for both threshold figures with No Learning (Figures 4.38(c) and 4.39(c)).

4.10.3 Summary of Adaptive Thresholding

In summary, as we have compared results in Figures 4.38 and 4.39, a wider CM deployment rate boundary interval accomplishes multiple goals. First, it allows the system to deploy CMs at a rate much greater than the baseline (100%) rate with minimal throttling when only weak models are available. This approach enables the system to neutralize as many TPs as possible at the expense of wasting many CMs on excessive FPs. Second, although we start the mission with a wide boundary interval, this interval width decays down to zero over time to prevent premature CM exhaustion or hoarding. We allow the interval width to decay to zero because we expect the sample score distribution to significantly improve with each freshly trained model in Live

Learning, meaning that the threshold toward the end of the mission should exhibit less variance. Third, the wider interval allows the system to react to potentially bursty threat arrivals. A cluster of samples, whether ground truth threats or not, with high inference scores will require a burst of CMs. Thus, a strict upper bound is in most cases unable to support such a burst, resulting in potentially unneutralized TPs due to throttling, which could be lethal. We saw that this wider CM deployment rate interval has the greatest effect on scenarios with the largest numbers of CMs. They are most affected by CM throttling in Live Learning toward the end of the mission when the maximum deployment rate approaches 100%.

Our approach to adaptive thresholding involved a decaying deadband mechanism that controlled the CM deployment rate as a function of time. We demonstrated the nuanced impacts of both strict and lax initial deadband boundaries on the fluctuation of the actual CM deployment rate and potential throttling. Overall, we showed that strict boundaries still allowed Live Learning to improve survivability with tight constraints on the number of CMs, but decreased it with plentiful CMs. Conversely, a wide initial boundary interval reduced the risk of CM throttling, allowing for surge CM deployment when necessary, ultimately improving survivability at all numbers of CMs.

4.11 Strategic Scout Deployment

When defining the initial analytical model (§3), one of the subtle, yet impactful, design decisions we made was that all scouts deployed at the start of the mission. In other words, they begin to sense and inference samples from the environment at $t = 0$. In many scenarios this may be the optimal approach given a particular objective. In other such cases (and often in many real world scenarios), it is frequently prudent to hold a percentage of one's operational assets in reserve. In benign environments where the only threat to a scout not completing its mission is internal malfunction or exhaustion of finite energy or other resources, there is no need to anticipate external threats and accordingly be both proactive and reactive to such threats. However, in the SCML context the adversarial agent is most likely to be the cause of mission failure, and therefore the manager of the team of SCML systems must plan and execute an appropriate and effective deployment configuration ideally tailored for the unique mission at hand. We model *Strategic Scout Deployment*, which is the ability to dynamically deploy and redeploy individual scouts to achieve specific mission success criterion, through several mechanisms.

4.11.1 SCML Scout Mode

First, we must be able to control in a centralized manner when any given scout actively inferences (or does not actively inference) samples from its environment. The purpose of this capability is to enable more granular control over resources (the total number of actively inferencing scouts) based on mission requirements (duration, expected threat density, expected threat lethality, etc.). As a result, we can control exactly how many scouts contribute samples to the Live Learning process at any instant in time. We define the three comprehensive scout operating modes below:

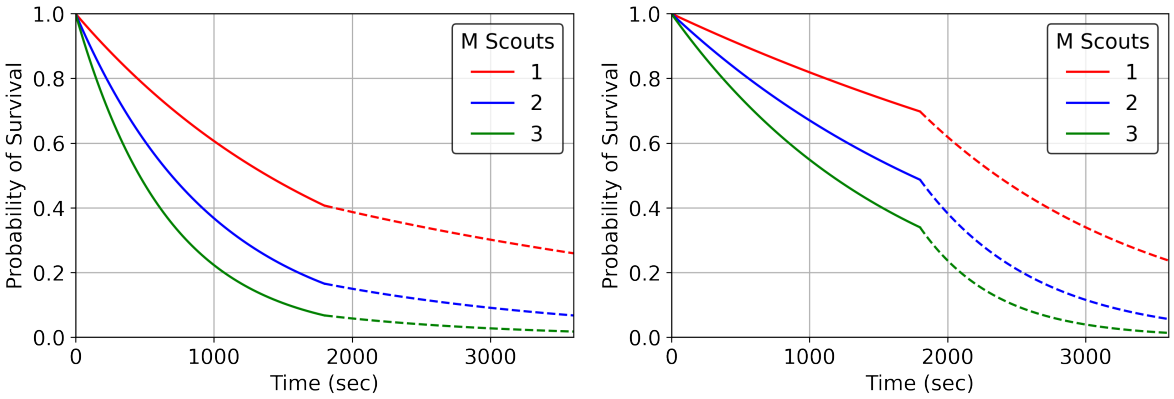
- *Active*: The scout inferences samples from the environment at a given rate (this is the default behavior). It performs all other Live Learning functions as previously defined:

prioritized transmission to home and retraining after receiving sufficient labeled TPs.

- *Idle*: The scout does not actively inference samples from the environment. By extension, it does not transmit any samples to home for labeling. We assume each scout has high bandwidth links to other scouts such that it still receives labeled samples from currently active scouts. This results in the idle scout's ability to retrain new models in parallel with the other scouts, albeit in a potentially purely supervised manner (as it may not have any stored unlabeled easy negatives). We also assume another scenario where the scout is physically located at home, and thus co-located with the human labeler. In this case, the idle scout would have access to all labeled samples to retrain its local models. In practice, an idle scout can be physically located at home or some other location, but in either case we make the same assumptions about available data (labeled samples) and its ability to train new models as with active scouts.
- *Destroyed*: A scout is destroyed when it encounters a lethal threat. This occurs when an actual threat (any FN, or a TP when countermeasures have been depleted) is lethal with probability γ. Once destroyed in the course of a single mission, a scout cannot return to either the *idle* or *active* states.

In §4.8, we defined more complex mission success criterion as at least M scouts must survive the mission out of N total scouts. In this section, both M and N scouts refers to the number of active scouts. This means that even if the number of active scouts varies during the mission, ultimately the criterion for mission success is simply whether at least M active scouts survive. More specifically, at least M active scouts must be alive *at all times* during the mission for it to be considered successful. The last point is critical in that we could simply deploy many additional scouts toward the end of the mission to achieve close to 100% likelihood that $M \geq 2$ scouts survive, for example. If three scouts are activated at the start of a mission (as shown in Figure 4.40(a)) that lasts 3600 seconds and the success criterion is $M \geq 2$ scouts, then there is some probability that $M \geq 2$ scouts survive the first half of the mission (roughly 17% after 1800 seconds according to the end of the solid blue curve). At this point in time, four additional scouts are activated. The dashed blue curve represents the survivability of $M \geq 2$ scouts for the remainder of the mission after the additional four scouts have been activated. Once activated at 1800 seconds, each of the four individual scouts have an initial 100% probability of being alive. However, the probability that $M \geq 2$ scouts are still alive beyond 1800 seconds is dependent on the same probability at 1800 seconds when only the three original scouts were active. This results in the expected contiguous survivability curves, where the solid and dashed lines connect at 1800 seconds. Intuitively, the curves decay quickly when only three scouts are active, but decay slowly when four additional scouts are activated. With seven total scouts active in the second half of the mission, there is a much higher probability that any two or more of them are alive (given that at least two survived the first 1800 seconds). However, this effect only slows the decay in survivability relative to the low probability of survival (about 17%) in the first half of the mission.

Figure 4.40(b) suffers from the opposite problem. Seven scouts are activated at the beginning of the mission as in the default configuration, which clearly improves the likelihood of survival of at least $M \geq 1, 2,$ and 3 scouts. In this scenario, we redeploy four scouts half way through the mission for any number of reasons, and the probability of success thus decreases significantly



(a) 3 Active Scouts → 7 Active Scouts

(b) 7 Active Scouts → 3 Active Scouts

These figures are not intended to be quantitatively accurate. They are examples of the effect on survivability of varying the number of active scouts during an SCML mission.

Figure 4.40: Toy Example of Delayed Deployment and Early Redeployment of Scouts

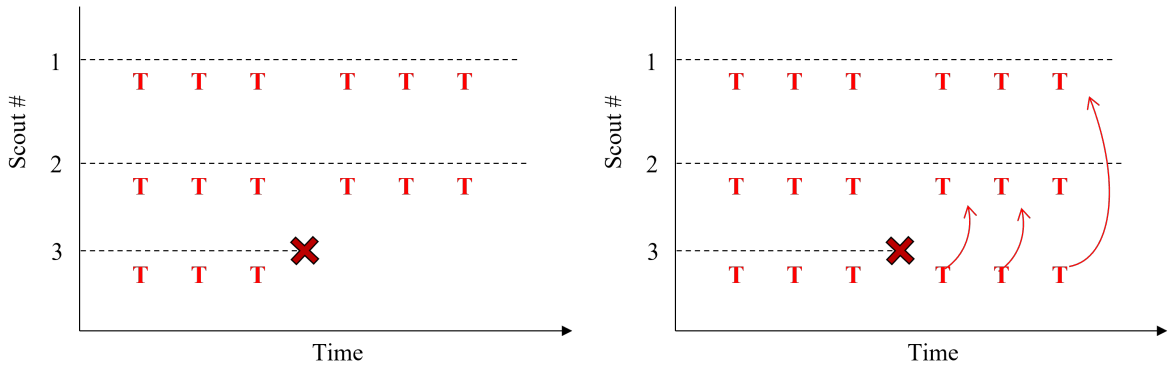
as only three scouts remain. Survivability decays more rapidly in the second half of the mission when fewer scouts are active, but the probability of success is much higher at the 1800 second mark given seven scouts were active in the first half.

The aforementioned figures are meant to depict the implications for survivability of various scout deployment configurations. Delaying the deployment of some scouts has a different impact on survivability than redeploying some scouts during the mission. Live Learning is also affected as there will be fewer samples inferenced from the environment with only three active scouts. Thus, SCML staggered deployment configurations have unique direct impacts on survivability as well as on the system's ability to leverage the power of Live Learning, which has a mutual dependence on survivability itself.

4.11.2 Threat Distribution & Scout Inference Rate

In our initial analytical model (§3), the average threat arrival rate, α, represented the average aggregate number of threat samples encountered per unit time across the total number of deployed scouts. By defining the arrival rate in this manner, we treated the team of scouts as a single consolidated SCML system. As a result, a single lethal threat caused the mission to fail. In practice, real SCML missions are likely to leverage many scouts across a wide geographic area, and therefore a single lethal threat is unlikely to damage or destroy more than a single scout. As in the previous section, we treat each individual scout as a separate SCML system and any subset of scouts will be active at any one time. Therefore, we must determine how to properly track the sample arrival rate (and by extension the threat arrival rate) of each scout and the aggregate rate across all scouts. We define two approaches below that are the simplest from an operational perspective. Of course there may be other, more complex approaches, but for our purposes of survivability analysis, the initial two are sufficient:

1. *Locally Constant*: This is the simplest approach in that the average threat arrival rate on



(a) Locally Constant

(b) Globally Constant

Locally vs. Globally constant threat arrival rates define whether the arrival rate remains constant for a single scout when it is active (local), or the arrival rate across all scouts remains constant, regardless of how many scouts are active.

Figure 4.41: Toy Example of Threat Redistribution when Fewer Scouts are Active

an individual scout is the overall rate divided by the total number of scouts N. Therefore, all scouts have the same local arrival rate, regardless of how many are currently active. Consider an example scenario to illustrate this approach in Figure 4.41(a). If $N = 3$ scouts are deployed at the beginning of a mission and after some period of time (marked by the red X), one scout becomes idle, then $N = 2$ scouts are now active. The threat arrival rate on each of these two remaining scouts does not change. Operationally, this makes two critical assumptions: 1) threats can only threaten a single scout, likely due to geographic distribution (only a certain number of threats can threaten a specific scout per unit time, regardless of what happens to other scouts), and 2) compared to a standard mission where all seven scouts are active the entire mission, fewer total samples will be inferenced (during this time interval), which ultimately hinders the Live Learning process while simultaneously reducing the number of encountered potentially lethal threats.

2. *Globally Constant*: This approach assumes that the aggregate arrival rate at any time across all active scouts remains constant during the mission. The corollary assumption here is that the adversarial agents can adjust their resources from idle scouts to active scouts. Figure 4.41(b) shows the same example mission where three scouts are initially active and one becomes idle. The adversarial agents thus shift their resources to the two active scouts, resulting in a greater threat arrival rate for the two remaining active scouts. In practice, this would occur when the scouts are in relatively close proximity to each other or it is feasible logistically for the threats to target a different scout. Holding the aggregate arrival rate across all active scouts constant for the entire mission naturally causes the arrival rate per scout to fluctuate according to the total current number of active scouts. If the number of active scouts increases, the arrival rate per scout decreases and vice versa. Although the same total number of samples are inferenced across all scouts with this approach, Live Learning will still be affected as fewer scouts will process more samples individually, potentially causing priority queue bottlenecks of true positives. Survivability performance

is also negatively affected as each active scout will likely encounter more threat samples per unit time (as compared to when the full team of scouts is active).

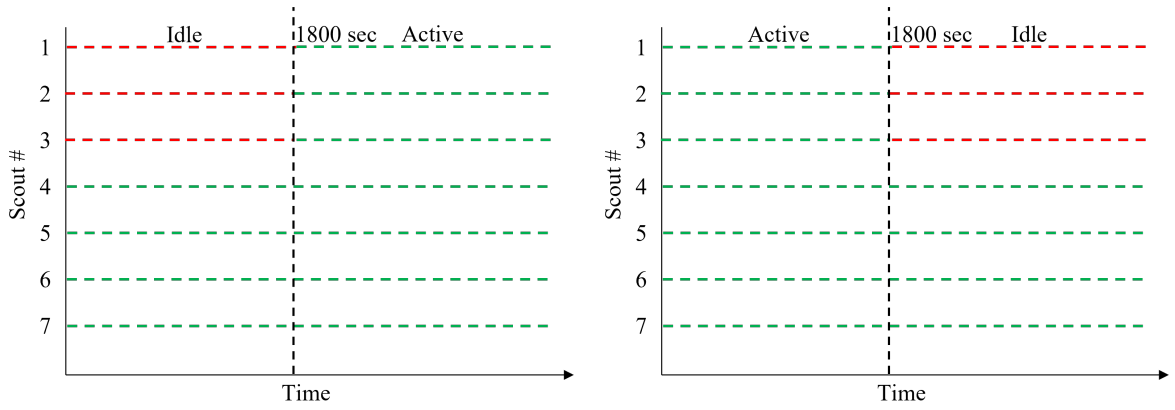
By explicitly controlling which exact scouts are active at any instant in time and the rate at which they inference samples from the environment, we can influence the behavior of Live Learning and the resulting SCML performance. Live Learning depends on a minimal number of rare positive samples to improve its models, and therefore the number of active scouts and the rate at which they inference have a significant impact. One scout collecting samples may not be able to provide sufficient TPs in a reasonable amount of time to trigger a retraining iteration, slowing learning. To mitigate this risk, we can reduce the threshold of newly discovered TPs for required retraining, but this risks producing an only marginally improved model with limited increase in performance. SCML mission success criterion may not accommodate this approach. If the requirement is to achieve a high probability that at least three scouts successfully complete the mission, then we will need to have at minimum three, but more likely four or five scouts deployed and active at any one time. In a sense, this co-dependent relationship is such that Live Learning is held hostage to SCML operational requirements and constraints, yet SCML systems are dependent on Live Learning for their enhanced survivability across time.

To implement strategic scout deployment, we modified Live Learning to account for dynamic scout behavior. We identify the Live Learning functions that must be modified for strategic scout deployment and those that remain unchanged below:

- Synchronized Retraining: By default, scouts launch each retraining process when a threshold number of positive samples have been labeled across all scouts. We assume idle scouts are able to continue to receive labeled samples via two possible methods: 1) over the network from other scouts as normal, and 2) if idle scouts are temporarily colocated at home with the labeler, relevant samples are simply transferred from home to the idle scouts.
- Sharing Negatives: By default, all scouts share only confirmed positive samples with their sister scouts for future retraining. Idle scouts will not receive labeled negative samples back from home as they are not actively inferencing. Thus, if scouts are idle at the beginning of a mission, they will not have any additional unlabeled negative samples (besides a small number of bootstrap negatives) for use in future retraining. With staggered deployment, labeled negatives, in addition to positives, are shared with sister scouts.
- Integration of Negatives: By default, each active scout only trains on labeled negatives inferenced locally, and therefore trains on an increasing number of total negatives each iteration (which includes a random sampling of local unlabeled easy negatives). Each idle scout, because it receives negatives (in addition to positives) from all active scouts with staggered deployment, randomly samples from the negative samples it receives from active scouts to ensure it does not cause an excessively imbalanced set of negatives for future retraining (several labeled negatives for each unlabeled negative).

4.11.3 Experimental Results: Strategic Scout Deployment

To demonstrate the impacts on survivability of staggered deployment scenarios, we performed a number of experiments with different configurations. Figure 4.42 depicts one such configuration. Figure 4.42(a) shows three idle scouts at the beginning of the mission, which go active with the



(a) 4 Active Scouts → 7 Active Scouts

(b) 7 Active Scouts → 4 Active Scouts

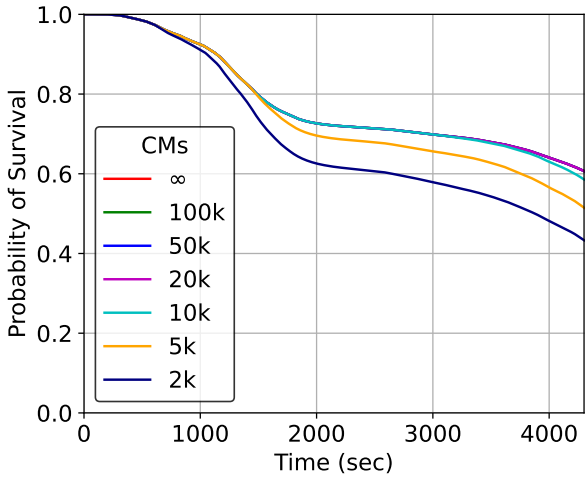
These figures demonstrate the experimental configuration for the following results where the number of active scouts changes dynamically during the mission.

Figure 4.42: Number of Scouts Active as a Function of Time

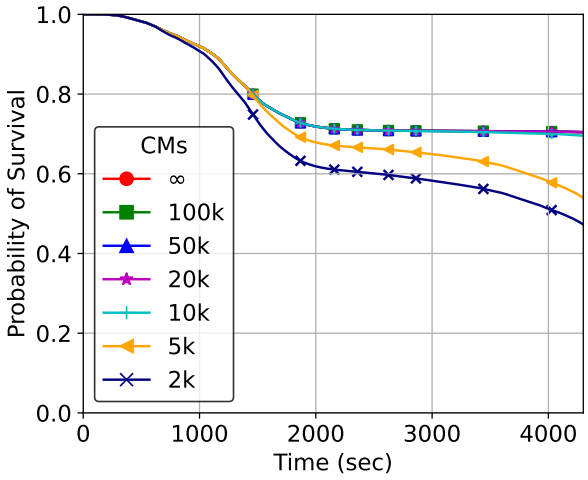
other four existing active scouts at 1800 seconds into the mission. Figure 4.42(b) shows the opposite scenario where all seven scouts are active in the first part of the mission, then three are redeployed or put into idle mode at 1800 seconds. These two staggered deployment scenarios are represented in the following experimental results.

With the given deployment scenario, we can observe the various tradeoffs in the number of active scouts, thresholds, CMs, survival criterion, locally vs. globally constant threat distribution, and so forth. Figure 4.43 depicts the value of Live Learning for locally constant threat distributions. In these results, *delayed deployment* represents the scenario depicted in Figure 4.42(a) where four scouts are deployed at the beginning of the mission and the remaining three are activated after 1800 seconds. Similarly, *redeployment* represents the scenario depicted in Figure 4.42(b) where all seven scouts are deployed at the beginning of the mission and three are redeployed (transitioned to idle mode) after 1800 seconds. First we observe with delayed deployment in Figures 4.43(a) and 4.43(b) the expected rapid decay in survivability early in the mission when only four scouts are active. At 1800 seconds when the additional three scouts are activated, survivability decays at a much slower rate. Conversely, redeployment scenarios (Figures 4.43(c) and 4.43(d)) exhibit minimal decays in survivability early in the mission when all scouts are active, and then rapidly decays after three scouts are redeployed or inactivated after 1800 seconds.

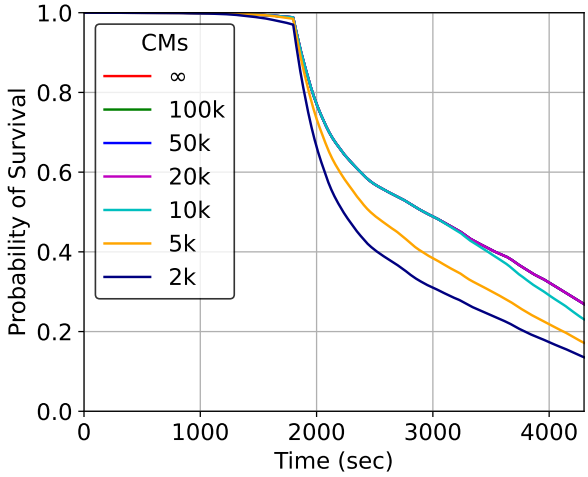
With delayed deployment, Live Learning provides only modest improvements in survivability in Figure 4.43(b) over No Learning in Figure 4.43(a), as $M \geq 3$, which is a rather relaxed survival criterion. The activation of three scouts at 1800 seconds slows the rate of decay in survivability to such a degree that Live Learning provides only minor additional improvements. However, in redeployment scenarios Live Learning provides significant increases in the probability of mission success with CMs of 10k and above, as shown in Figure 4.43(d). As all scouts are active at the beginning of the mission, survivability at all levels of CMs is virtually 100%. If CMs are not severely constrained, Live Learning is able to greatly limit the rapid rate of decay in survivability



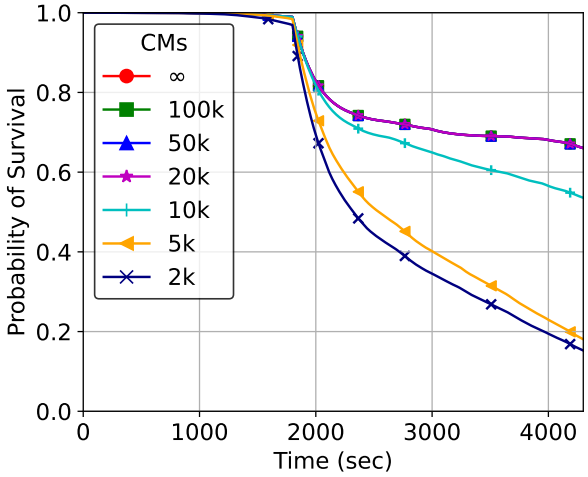
(a) No Learning, Delayed Deployment



(b) Live Learning, Delayed Deployment



(c) No Learning, Redeployment



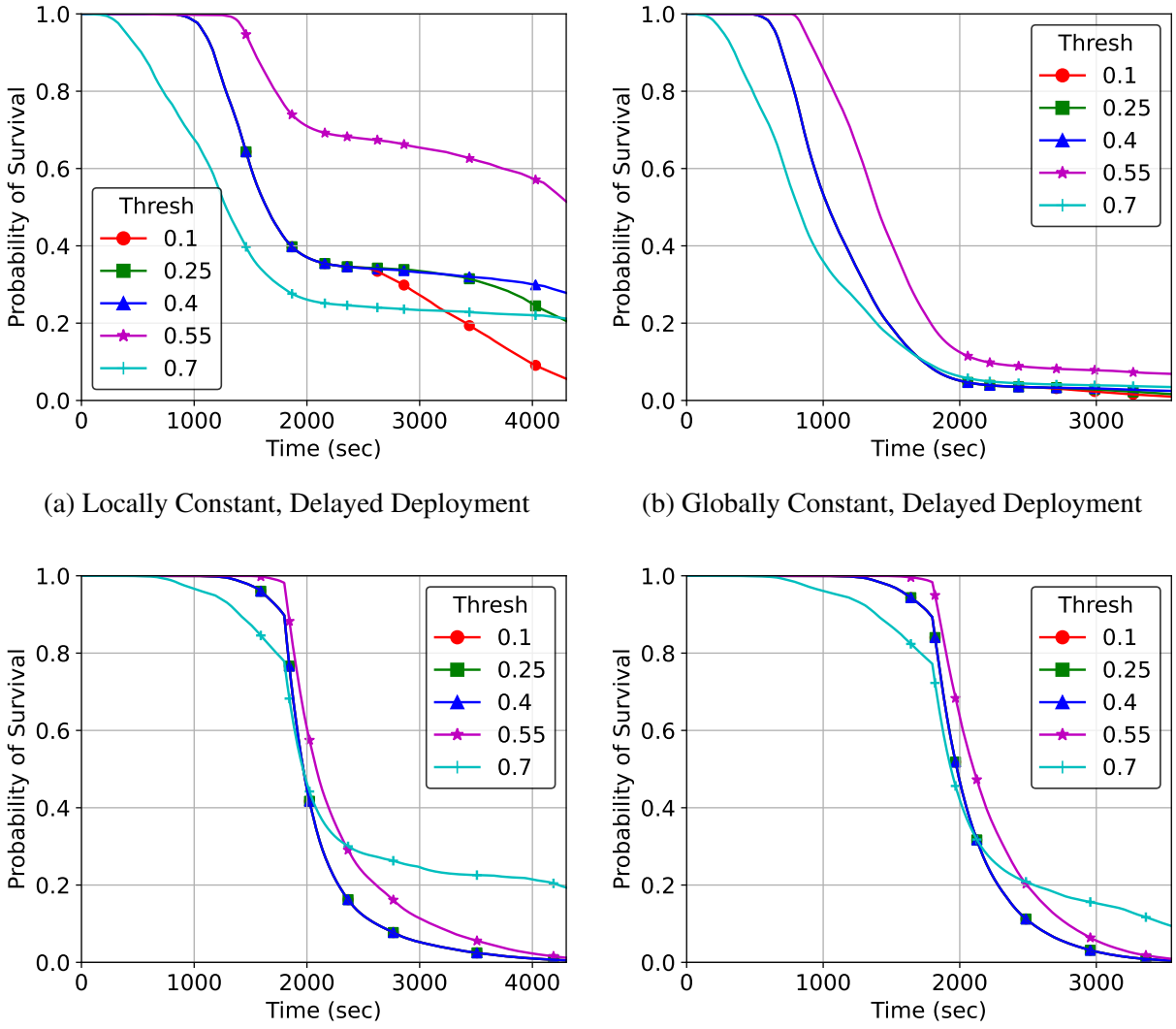
(d) Live Learning, Redeployment

All graphs above assume a threat lethality of $\gamma = 0.02$, $M \geq 3$, and a threshold of 0.7. These results correspond to SCML missions of class Swimming Pool of the DOTA dataset.

Figure 4.43: Value of Live Learning for Survivability with Staggered Deployment by CMs

once three scouts are deactivated at 1800 seconds. The key insight is that Live Learning is far more valuable to enhanced survivability when it must compensate for the reduced number of active scouts contributing to the mission success criterion. In these results, $M \geq 3$ of $N = 7$ total scouts, therefore the relative survival criterion is not strict. As a result, because most of the retraining iterations occur after 1800 seconds, Live Learning adds maximum value in redeployment scenarios when only four active scouts are active and can contribute to the $M \geq 3$ required scouts.

It is also important to understand the tradeoffs between locally and globally constant threat arrival rates with respect to both delayed deployment and redeployment scenarios as previously



(a) Locally Constant, Delayed Deployment

(b) Globally Constant, Delayed Deployment

(c) Locally Constant, Redeployment

(d) Globally Constant, Redeployment

All graphs above assume a threat lethality of $\gamma = 0.05$, $M \geq 3$, and 50k CMs. These results correspond to Live Learning SCML missions of class Swimming Pool of the DOTA dataset.

Figure 4.44: Tradeoffs in Survivability for Staggered Deployment Strategies by Threshold

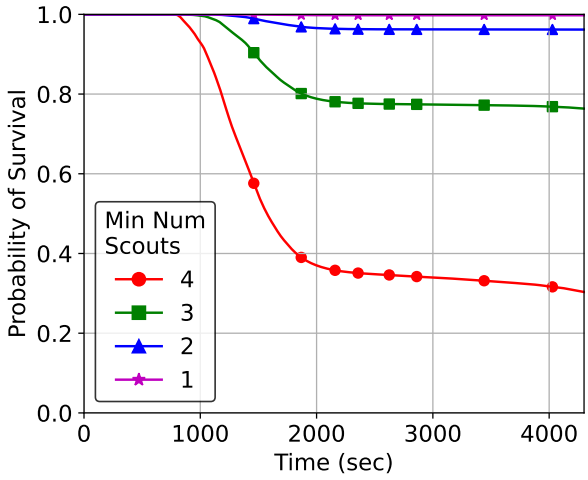
defined. For previous experimental results, we defined the mission duration as the time required to inference all samples in the stream. Therefore, previous results in Figure 4.43 show the mission ending at roughly 4200 seconds. This is because for part of the mission only four scouts inferenced, extending the amount of time required to inference all samples, and as a result, the mission duration. In the following context, when we compare results with different aggregate inference rates (locally constant versus globally constant), we must define mission duration independent of such rates. Therefore, we set 3600 seconds as the mission duration as this is the time required to inference all samples when the aggregate inference rate across all scouts remains constant throughout the mission. This time represents the duration which the scouts must

perform a reconnaissance mission in a given area, for example. As an intuitive result, locally constant scenarios are those where threats can only target SCML systems in geographic proximity. Once more systems are activated, the aggregate threat arrival rate will increase as threats can target more systems per unit time. For locally constant inference rate missions, when fewer than the total set of seven scouts are active, the aggregate inference rate is reduced accordingly, requiring an additional 600 seconds to inference the remaining samples. In this manner, we can compare the probability of survival between locally constant and globally constant missions at 3600 seconds, even though locally constant missions have yet to complete inferencing all samples at that point in time. The expected result is that locally constant missions will experience much less decay in survivability during the time interval when a subset of scouts are active.

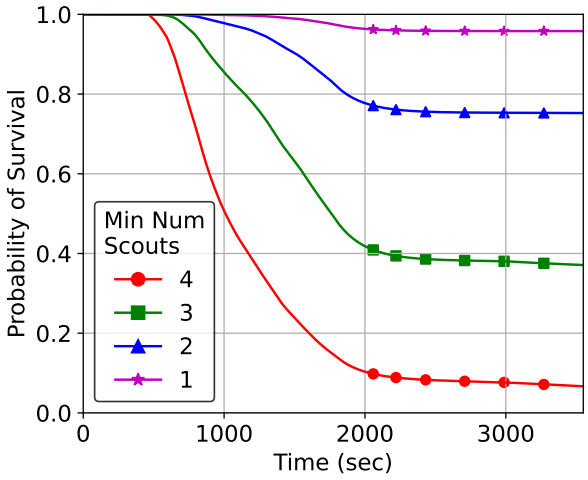
Figure 4.44 illustrates the tradeoffs between locally constant and globally constant missions. Locally constant missions (Figures 4.44(a) and 4.44(c)) have an extended X-axis to visualize when all samples have been inferenced, but still have a mission duration of 3600 seconds to compare to the other two figures. Figure 4.44(b) depicts the challenges with a globally constant sample arrival rate with only four active scouts early in the mission. The result of intersecting these two mission properties is the severe decay in survivability from four scouts inferencing seven scouts' worth of samples and a significant challenge in collecting sufficient positive samples at home to perform retraining. With a weak bootstrap model at the beginning of the mission and only four active scouts, a greater number of positive threat samples compete with many more negative samples in scout priority queues, delaying the first retraining until after 2000 seconds into the mission. Figure 4.44(a) shows the same delayed deployment scenario, but with locally constant arrival rates where the first retraining occurs closer to 1400 seconds, which significantly slows the decay in survivability not only from a decreased threat arrival rate but also from the learning itself. In this figure, we also notice clear differences in the CM depletion cliff between thresholds of 0.1 ($\sim$2600 seconds) and 0.25 ($\sim$3400 seconds), which cause accelerations in survivability decay late in the mission.

Figures 4.44(c) and 4.44(d) demonstrate the negligible impact of threat distribution on survivability when seven scouts are active and mission success criterion is only $M \geq 3$. Survivability remains close to 100% for the first 1800 seconds, in spite of differential overall threat density between locally and globally constant missions, as seven active scouts easily maximize the probability that at least three are still alive. However, once three scouts are converted to idle mode (deactivated), a globally constant arrival rate cuases survivability to decay faster than a locally constant rate. At the end of the mission (3600 seconds), all thresholds have a greater likelihood of mission success with a locally constant rate over a globally constant rate. These figures as a whole show that a globally constant arrival rate in which threats can be redistributed to other scouts when fewer are active impacts survivability more with a delayed deployment (in which additional scouts are activated later in the mission) rather than a redeployment Live Learning mission. This is due to a greater threat density when scouts generally inference with weaker models (leading to more potentially lethal threats) and an associated delay in generating improved models from having to compete with many more negative samples across fewer scout priority queues.

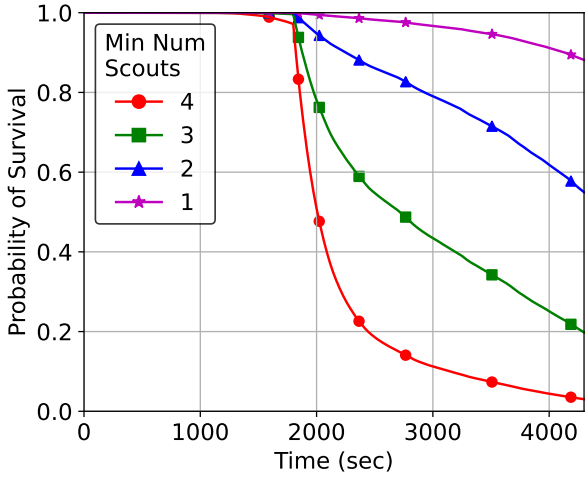
We also show how the mission success criterion affects survivability under various deployment scenarios (Figure 4.45). We observe that the percentage in survivability gains increase with the strictness in survival criterion with a locally constant arrival rate in Figure 4.45(a) over Figure



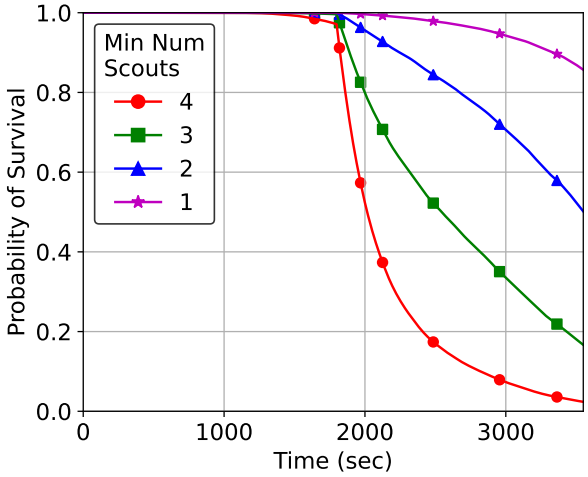
(a) Locally Constant, Delayed Deployment



(b) Globally Constant, Delayed Deployment



(c) Locally Constant, Redeployment



(d) Globally Constant, Redeployment

All graphs above assume a threat lethality of $\gamma = 0.02$, 50k CMs, and a threshold of 0.4. These results correspond to Live Learning SCML missions of class Swimming Pool of the DOTA dataset.

Figure 4.45: Tradeoffs in Survivability for Staggered Deployment Strategies by M Scouts

4.45(b). For example, $M \geq 1$ with locally constant improves to 100% from 95% with globally constant while $M \geq 4$ improves from about 7% to 33% (almost 5x) (at 3600 seconds) from globally to locally constant.

We observe similar gains in survivability in redeployment scenarios in Figures 4.45(c) and 4.45(d) by the end of the mission (3600 seconds). These figures demonstrate the wide distribution in the probability of mission success as a function of at least M scouts required to survive. We also showed that a locally constant threat arrival rate improves performance significantly over a globally constant rate in a delayed deployment scenario as compared to a redeployment

scenario, which produces similar survivability outcomes for the different threat arrival rates. This is due to the differential arrival rate in potentially lethal threats early in the mission. It is also because of the challenges with early model retraining when fewer scouts are available for sample prioritization and transmission to home for labeling.

4.11.4 Summary

We have described in this section the performance tradeoffs when deploying scouts in dynamic scenarios, where scouts may transition from idle to active or in reverse. As expected, such transitions affect the rate at which samples are inferenced both at a local (per scout) level and at a global (across all scouts) level. Whether a scout is active or idle determines whether it can contribute samples for model retraining, whether it can share its labeled samples with other scouts, and ultimately whether it can be included in the statistical estimation as to whether the mission success criterion is being met.

Our experimental results demonstrated that differential threat densities early in the mission (delayed deployment missions) have significant impacts on survivability along with the effects on the underlying Live Learning processes that enable model retraining to occur. Weaker models generally imply lower scoring positive samples having to compete with higher scoring negative samples, In scenarios with extreme class imbalance, there are far more of the latter than the former. The combination of these effects result in the poorest performance in globally constant, delayed deployment scenarios. We also showed that Live Learning adds the greatest value in compensating for reduced overall survivability when scouts are converted from active to idle mode, by slowing the overall rate of decay. Overall, we analyzed how mission success criterion, the threat arrival mode, a delayed deployment or redeployment scenario, and the added value of Live Learning all make unique impacts on the probability of mission success when employing staggered scout deployment. The implications of this analysis are: 1) the explicit understanding of the mutual dependence of the operational aspects of SCML (how many scouts to activate/deactivate and when, success criterion, etc.) and the underlying ML functions such as Live Learning, and 2) an insightful tool to optimally deploy SCML systems as a function of the high-level mission objectives constrained by finite resources, expected adversarial threat density, and expected ML performance.

4.12 Novel Class Discovery and Adaptation

We described how the nature of adversarial environments motivated the need for SCML in §1. Threats evolve and morph while novel (unpredictable) threats may be discovered during a mission. Therefore, SCML systems must be able to react to such unpredictable classes of threats to maintain a robust probability of mission success in highly dynamic scenarios. Live Learning can be modified to provide this capability to SCML systems. We discussed in §1.3 how Live Learning is analogous to the strengthening of a biological entity's immune system over its lifetime. If an animal is exposed to many similar instances of the same virus, its immune system will adapt and develop the ability to fight and neutralize exposure to such instances in the future. However, if an animal is exposed to a completely different virus, perhaps one that attacks a completely dif-

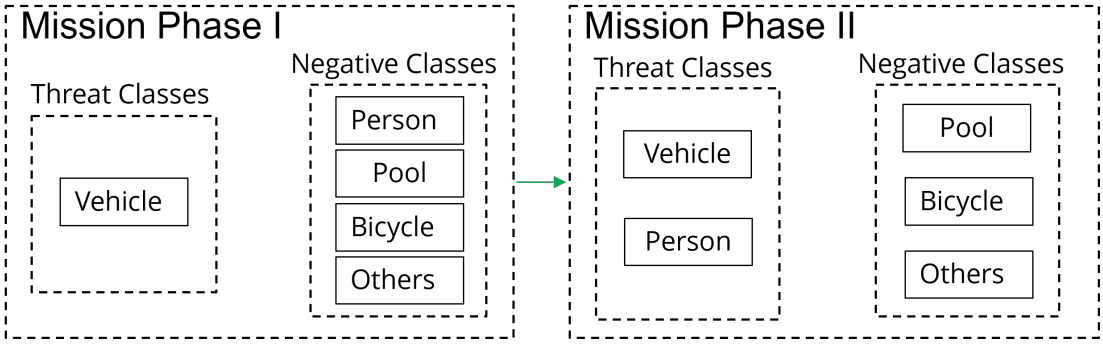


Figure 4.46: Example SCML Mission Phase Transition

ferent internal system or uses a novel attack vector, the immune system must adapt to it *without forgetting* how to react to the original viral threat. As a result, the immune system develops additional capabilities to neutralize many *novel* threats over its lifetime for which it was not originally prepared. In contemporary ML research, the fields of Open Set Recognition (OSR) [99–102] and Open World Recognition (OWR) [103–106] are the fields that most accurately represent this capability. OSR typically involves determining whether a sample is *in-distribution* and the model classifies it as one of the classes represented in the training data or *out-of-distribution* (OOD), in which the model determines it does not belong to one of classes represented in the training data. In the SCML context, we assume extreme class imbalance, where the vast majority of inferenced samples belong to the negative or background class, which may contain many different potential classes of interest that are currently out of distribution in that they have not been explicitly trained into the model. OWR extends OSR by recognizing OOD samples and training their features into the model as a new class. In the context of Live Learning and SCML, we call this *Novel Class Discovery and Adaptation*. In adversarial environments where exhaustive a priori knowledge of all possible threats is impossible, it is critical for SCML systems to be able to detect threats that are unknown prior to the mission.

Live Learning with Hawk assumes a small number of positive samples representative of the threat class (or classes) are available for training a bootstrap model prior to mission start. During the mission, additional samples are encountered, inferenced, prioritized for transmission from scout to home, and labeled by the human expert for future model improvement. While this is a tightly coupled, pipelined approach to continuously improve survivability, it does not account for the recognition of *novel threats*. We define a novel threat as a class of objects that neither the human expert nor the scouts currently recognize as one of the current threat classes. In other words, samples containing an instance of a novel threat are either labeled as negatives when presented to the expert or are never even transmitted from scout to home. SCML systems that perform Live Learning must be able to adapt to such a novel threat during a mission with no previous knowledge of its existence. Novel class discovery occurs via two different methods, which are described in detail in the following subsections:

1. Passive Novel Class Discovery
2. Online Semi-Supervised Clustering

Regardless of the specific method, a new class is ultimately created by the expert at home (to

ensure such an important action aligns with human-dictated operational goals) and labels for its constituent samples are sent back to the scouts (discovery). To leverage labels of a new class, the scouts must modify their model architectures such that they generate probability scores that include the novel class(es) (adaptation). As a result, new instances of the novel class(es) will be prioritized for transmission from scout to home as with the existing threat classes. Conceptually, this capability allows Hawk to expand its ability to perform Live Learning for C threat classes to $C + N$ threat classes dynamically during a single mission, where C is the number of original threat classes at the start of the mission and N is the number of novel threat classes discovered during the mission. A conceptual diagram of this state change with a specific class example is depicted in Figure 4.46. In this example, phase I represents the time interval when class vehicle is the only threat class. Samples considered negatives in this example may contain people, swimming pools, bicycles, and many other classes. At some point during the mission, the domain expert determines that person should be considered a threat class in addition to vehicle, transitioning the mission from phase I to phase II. The detailed processes by which new classes are recognized or detected, either at home or on the scouts, are documented in the respective subsections.

Hawk was originally designed and optimized for binary classification: a single positive class and negative class. To support dynamic novel class discovery during the mission, the system has to support multi-class classification. With respect to the model, this is trivial. What is not as trivial is ensuring that scouts and home are synchronized as to the current number of classes, their class names and labels, respective data management, and statistics management. In a typical Live Learning mission, the models on the scouts are attempting to discover C positive classes of threat objects. Therefore, when performing pure classification, the models generate an output vector of $C + 1$ class probabilities to account for the negative class. The expert decides that one or more samples present in the labeling interface meet a subjective threshold (threat assessment) that warrants creating a new class and labels the sample(s) as such. The new label(s) of class $C + 2$ and beyond are transmitted back to the scouts. At the next retraining, the model is modified such that it outputs a vector of length $C + 2$. Figure 4.47 depicts the model transition from an output of $C + 1 = 2$ nodes to $C + 2 = 3$ nodes. In this case, the number of original threat classes is $C = 1$ and one novel class is added. The node labels 1 and 2 represent threat class 1 and threat class 2 while node N represents the negative class. Threat samples inferenced with this adapted model are transmitted to home with a score vector containing both the existing and new class(es). At this point, Live Learning functions as normal except that now the human expert monitors samples for one additional threat class.

4.12.1 Passive Novel Class Discovery

Conceptual Description

The simplest approach to perform novel class discovery is to exclusively leverage human domain expertise. When inspecting and labeling samples received at home, the domain expert recognizes one or more samples containing a suspicious or interesting object that should be considered a potential threat, yet it is not deemed a member of one of the current threat classes. The expert creates a new class, labels the sample(s) as members of the new class and sends the labels

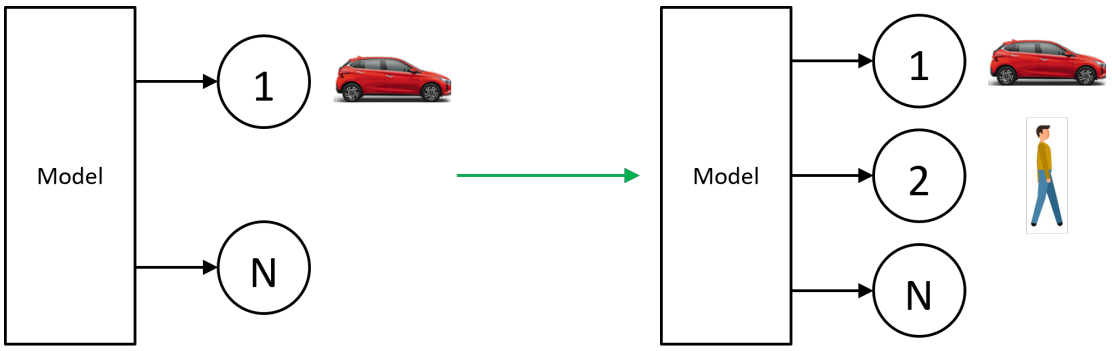
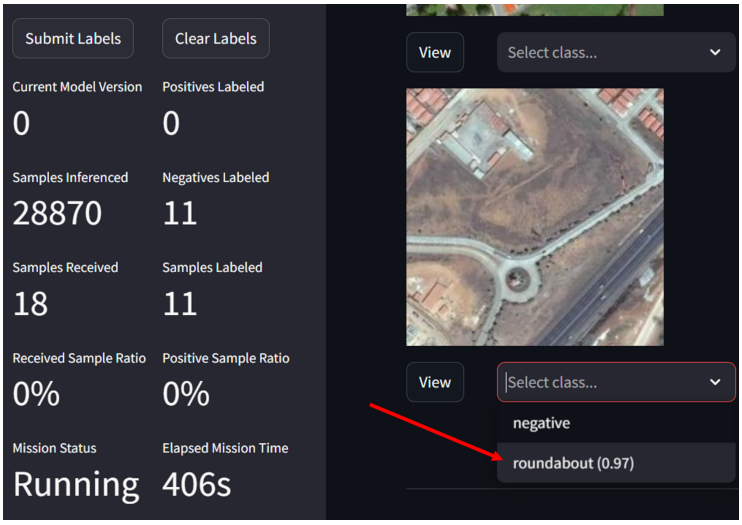


Figure 4.47: Expanding the Output Vector of a Simple Classification Model

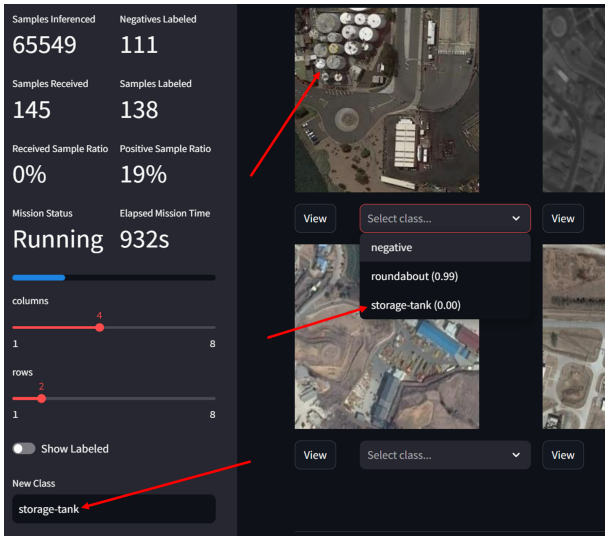
back to the scout. Once retraining conditions are met, the classification head of the model is replaced to accommodate the output score(s) for the new class(es). This process is referred to as "passive" as no additional processes run on the scouts or at home to perform this task. Therefore, the only new tasks performed are that of the domain expert observing a sample, deeming it valuable, creating a new class, and labeling the sample. The model adaptation occurs once retraining conditions are reached, but is transparent to the human. Figure 4.48 depicts the core steps of this process from the Hawk web interface, each of which are described next.

1. Samples are inspected and labeled by the expert appropriately for the current threat class(es) and negatives (Figure 4.48(a)).
2. The expert determines that a sample should be treated as a novel threat class based on domain knowledge and operational context (Figure 4.48(b)).
3. The expert creates a new class in the user interface and labels the appropriate samples (Figure 4.48(b)).
4. The labels for new and existing threat classes are sent back to the the scouts.
5. At the next retraining iteration, the model's classification layer is modified such that its output vector contains scores for the negative class, existing threat classes, and all new classes (Figure 4.48(c)).
6. The mission continues as normal, except now the expert will be tracking one or more additional threat classes for labeling.

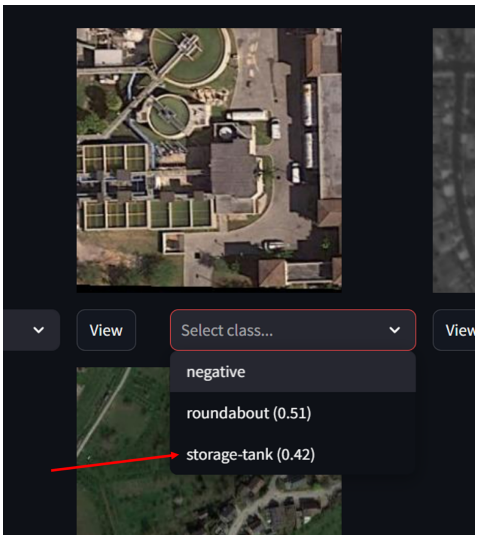
This is a passive approach, and therefore the implicit, but necessary assumption is that any novel class discovered by the expert is similar in feature space to samples of existing threat classes. The scouts prioritize samples for transmission based on the likelihood that they belong to one of the threat classes (by summing the scores of all threat classes). If the current model has not been trained on some yet to be discovered novel threat class, the only chance these samples have of being transmitted to home for discovery is if they sufficiently resemble one or more of the threat classes. This type of data is referred to as *Near Out-of-Distribution* or Near OOD. Near OOD data is that which the model has not been explicitly trained on, but is similar to such data. In the Hawk context, Near OOD data is part of the set of background samples inferenced by the scouts. It is similar in feature space to an existing threat class, but cannot quite be considered a



(a) Labeling a Sample of the Current Threat Class



(b) Identifying and Creating a Novel Threat Class



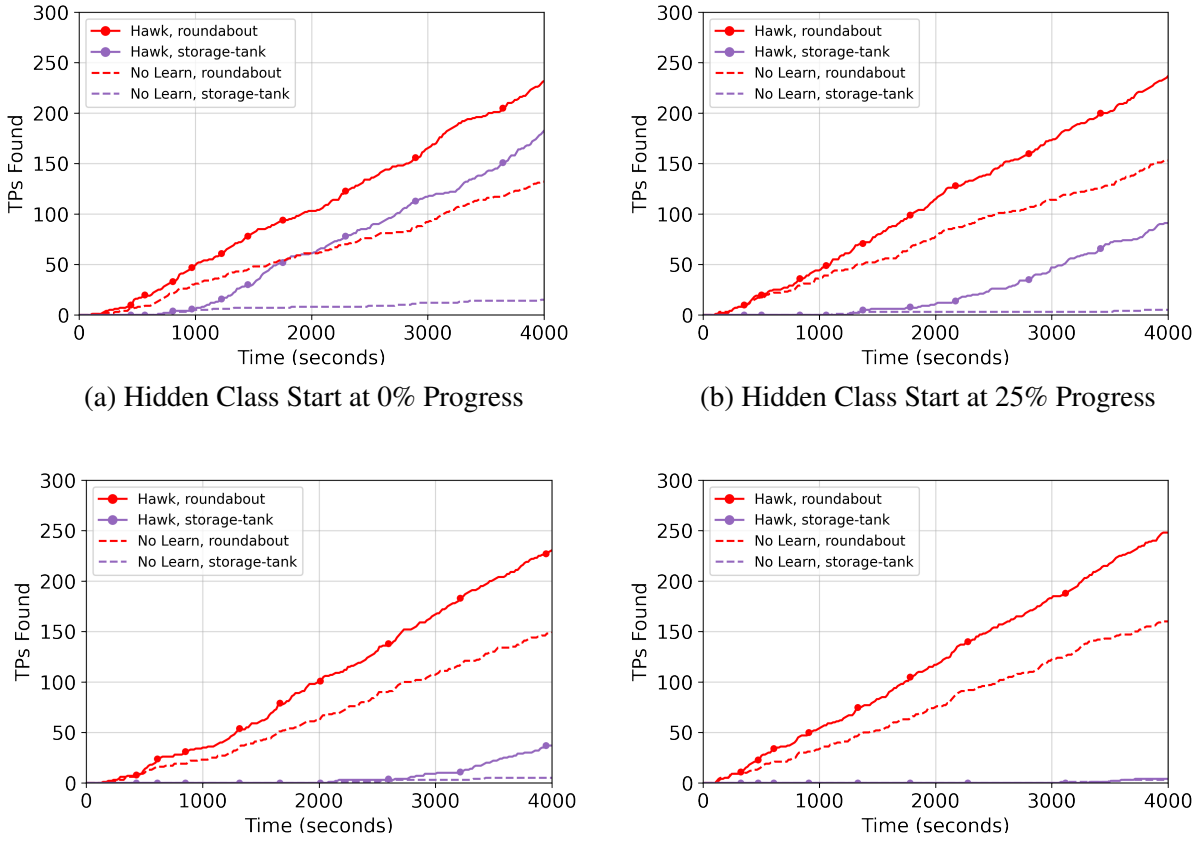
(c) Score for New Threat Class

These three images represent the process by which a human domain expert labels samples for the existing threat class, identifies a novel threat class, creates it and labels samples accordingly, and finally verifies that the new class receives its unique classifcation score from the adapted models on the scouts.

Figure 4.48: Adding a Novel Threat Class in the Hawk GUI

member of any of the existing threat classes.

Discovering Far OOD data is also possible, however, it is the practical equivalent of treating a dog as the threat class and potentially discovering a large tractor. They may both be interesting objects in a given scenario, but are so different in feature space that there are likely several other classes of objects in the environment much more similar to dog. Thus Near OOD is a more tractable problem, especially in a Live Learning setting where the time horizon for finding novel



(a) Hidden Class Start at 0% Progress

(b) Hidden Class Start at 25% Progress

(c) Hidden Class Start at 50% Progress

(d) Hidden Class Start at 75% Progress

These results correspond to Live Learning and No Learning missions of class Roundabout that performs Passive Novel Class Discovery for class Storage-Tank, both of the DOTA dataset.

Figure 4.49: Positive Samples Discovered of Primary Class Roundabout and Hidden Class Storage-Tank for Dataset DOTA using Passive Discovery.

classes is short using a passive approach.

Experimental Results

To experimentally validate passive novel class discovery, we created a typical Live Learning scenario with approximately 250,000 samples distributed across seven scouts, 300 of which contained at least one instance of class roundabout of the DOTA dataset. Also included were 300 samples containing at least one instance of class storage-tank. The initial bootstrap model was only trained on 20 roundabout examples but no storage-tank examples, thus the model had no initial knowledge of storage-tanks. In this scenario, we call the storage-tank class the "hidden" class as it was not explicitly trained into the bootstrap model but it was present in the set of streaming samples inferenced by the scouts.

We introduced the hidden samples at four different points in time during the mission, as depicted in Figure 4.49. Each of the four figures represents two experiments, one employing Live Learning with Hawk and the other using No Learning. For each of these experiments, we

collected the same statistics as previous results: the cumulative number of positive samples transmitted to home as a function of time. Figure 4.49(a) represents experiments in which samples of the hidden class were introduced in the stream at the start of the mission (the same time as those of class roundabout). As expected, once the first new model is installed, the Hawk curve for roundabout begins to increase faster than the No Learning curve, resulting in 232 total positive roundabout samples found compared to 132 by the end of the mission. Although hidden samples of class storage-tank start to be inferenced by scouts at the beginning of the mission as with samples containing roundabouts, no samples are transmitted to home until about 750 seconds into the mission. This is because the probability that any sample containing a storage-tank is transmitted to home (based on their classification score and position in the priority queue) is purely based on its similarity in feature space with the roundabout class.

We also notice in Figure 4.49(a) that Hawk is able to find 186 storage-tank samples from passive discovery. Around 750 seconds into the mission, the first storage-tank is transmitted to home by one of the scouts, labeled, and returned to the scouts. Now that at least one labeled sample of the novel class is available on the scouts, when the next retraining is triggered, the model output layer is modified. This allows the newly retrained model to output scores for classes roundabout, storage-tank, and negative. The first retraining with at least one storage-tank sample occurs at roughly 800 seconds, which is when the model's output vector expands to support the new class. It also embeds initial feature data into the model from a small number of samples of class storage-tank. After the next few new models are trained, the rate at which storage-tank samples are found is approximately equal to the rate at which roundabout samples are found. Without the capability to learn, in the dashed purple curve, the number of storage-tank samples transmitted to home is greatly reduced from 186 to 15. Such a significant difference in performance demonstrates how critical the adaptation and learning process is to collect the greatest number of samples of a novel class created during the mission.

For the three other scenarios in which hidden samples are delayed, the sample average arrival rate to the scouts remains the same as if they began arriving at mission start. For example, in Figure 4.49(b) there are 225 total hidden samples across 3000 seconds (starting at 1000 seconds into the mission), 150 samples in Figure 4.49(c) across 2000 seconds (starting at 2000 seconds into the mission), and 75 samples in Figure 4.49(d) across 1000 seconds (starting at 3000 seconds into the mission). In Figure 4.49(b), only after the first three retraining iterations (that have included samples of both roundabout and storage-tank) does the rate of storage-tank samples found begin to approach the rate of roundabout samples found in the respective Hawk curve. We also observe in this figure that both the Hawk and No Learning curves for class roundabout achieve slightly higher totals of collected positives (237 and 154, respectively). As fewer total storage-tank samples (225) are distributed in the stream beginning at 1000 seconds, they consume fewer precious transmission slots at the head of the priority queue on each scout. The effect is more pronounced in Figure 4.49(d), where 248 and 160 roundabouts are found for Hawk and No Learning, respectively, as only 75 total storage-tank samples are in the stream, beginning at 3000 seconds.

Clearly, in all four figures the difference between the number of positives found between Live Learning with Hawk and No Learning is far greater for a novel class discovered during the mission (storage-tank) than with a class for which a boostrap model was trained prior to mission start (roundabout). We also observe that as hidden samples are first introduced at later

and later times, the ability to learn is greatly reduced as more epochs of training are executed on data of the original threat class (roundabout), making it less likely that storage-tank samples are assigned a sufficiently high score to be prioritized in the queue for transmission. In other words, repeatedly training on roundabout samples as the only positive class biases the model more toward identifying unique features of roundabouts, and therefore reduces the likelihood of sufficiently high scores for storage-tank samples to be discovered.

Summary

Passive novel class discovery is the simplest method within the context of Live Learning to dynamically discover and adapt to a new class of interest according to mission requirements, dependent entirely on its similarity to the existing positive class. It avoids any meaningful computational requirements at home or on the scouts. The only modification to Live Learning on the scouts is the modification of the output classification layer of the models once samples of a new class have been labeled. The only additional task for the domain expert at home is to choose to create a new class and assign observed samples to it, if deemed appropriate. Our results showed that the capabilities of Live Learning provide far greater value to novel class discovery and model adaptation than it does for building a training set for a class for which we provide a bootstrap model, relatively speaking. This is because finding any reasonable number of samples of a novel class is extremely challenging without the capability to learn from even one newly labeled sample of a novel class. We also described how avoiding extreme model bias toward the initial threat class can be achieved by introducing samples of a potential novel class earlier in the mission.

4.12.2 Semi-Supervised Online Clustering

While the discovery of near OOD data is useful when the goal is to find novel classes of interest similar to existing classes, in other scenarios it may also be valuable to discover instances of far OOD data. Far OOD data represents that which is radically different than data the current models have been trained on. For example, suppose the original threat class was car. A near OOD class would be truck. A far OOD class could be an airplane: far larger, much different general shape, and long wings protruding from the fuselage. Passive discovery with a model trained on cars may have difficulty finding samples containing only an airplane. SCML systems ought to have some mechanism to discover potential threat classes that largely do not resemble current classes.

4.12.3 Conceptual Description

Our approach to enabling discovery of both near and far OOD classes involves more complex and computationally intensive tasks than passive discovery. The previous method used inference scores as the exclusive variable in determining which samples are transmitted to home for labeling. It relied upon the similarity of a potentially novel threat class to the existing threat class. This is under the assumption that the scores of the former would place them toward the head of the priority queue, as with the latter. In contrast, this method does not exclusively rely upon inference scores. As a result, it gives much lower scoring samples a chance to be selected by the scout, transmitted to home, and ultimately confirmed or discovered by the human expert. The

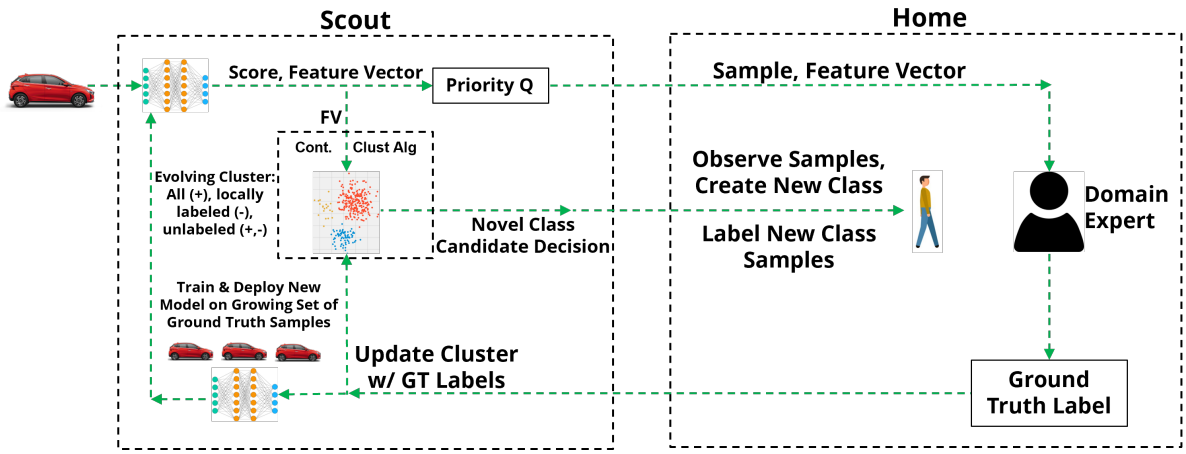


Figure 4.50: Semi-Supervised Online Clustering within Live Learning

details by which samples are selected for transmission to home are described in detail in this section.

Passive discovery leverages the existing selection process, which simply transmits the sample at the head of the priority queue from scout to home. The semi-supervised online clustering approach seeks to make such a decision based on the operational preferences of the domain expert. We characterize it as online as we cluster on new unlabeled samples that have been recently inferenced by the scouts. Figure 4.50 depicts a system-level diagram that integrates the clustering process within Live Learning. During inference, the model extracts a feature vector in addition to the default inference score for each sample. The feature vector of every sample is sent to a clustering queue in parallel with the default priority queue. The clustering process is triggered periodically based on time or number of inferenced samples. Once the process has completed, a number of samples of novel class candidates are selected for transmission to home. These samples are displayed in a separate browser page from the main Hawk labeling interface. Once the samples are visible in the browser, the domain expert performs the same set of steps as described in §4.12.1 to create a new class, assign a sample to that class, and send the labels back to the scouts. As labels for the new and existing classes continue to be sent to the scouts, they are associated with the relevant feature vectors in order to perform future iterations of semi-supervised clustering.

We now describe the internal steps of the actual clustering algorithm in detail. Figure 4.51 shows a conceptual example of semi-supervised clustering in the Live Learning context with class imbalance. As mentioned, each clustering iteration is triggered on some time interval or a threshold of inferenced samples. Labels and features vectors are also available for all samples transmitted to home and labeled thus far in the mission. Overall, the figure depicts a general feature space containing labeled and unlabeled samples. Labeled samples are members of one of two positive classes and the negative or background class. There are also a large number of unlabeled samples that could belong to one of the existing positive classes or the negative class (from which we could derive a novel class). We break the process into three primary steps, each of which are configurable according to the operational goals of the respective mission:

1. Select which unlabeled samples to label. A feature vector for each inferenced sample is

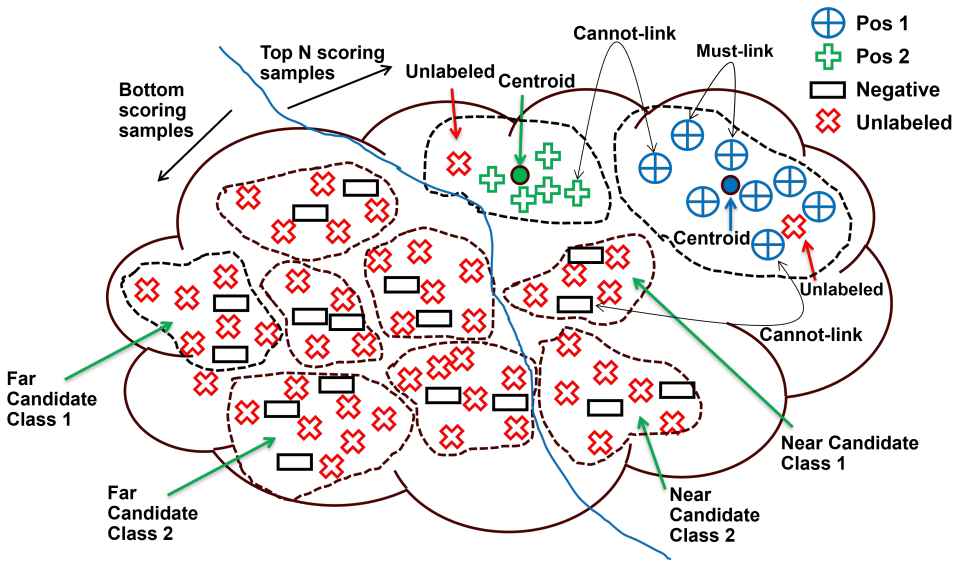


Figure 4.51: Semi-Supervised Clustering for Novel Class Discovery

placed in the clustering queue. Once the current iteration is triggered, a set of unlabeled samples currently in the queue are selected for clustering. The selection policy can take three forms. One policy selects the top N scoring samples, as shown in Figure 4.51. Samples located to the right of the blue line that loosely bisects the feature space would be selected under this policy, which are most similar to the existing threat class(es) (optimal for near OOD discovery). Similarly, the bottom N samples could be selected from the queue for clustering. These are samples likely to be the most different from the existing threat class(es) (optimal for far OOD discovery). A random sampling across all unlabeled samples in the queue can be used to select a diverse set of samples that are both similar and different to the existing threat class(es).

2. Determine which type of clustering to perform. Multiple options exist here as well. By default, we employ simple K-means clustering where K is configurable. We also must choose whether to cluster only unlabeled samples (noted by the red markers in Figure 4.51) or both labeled and unlabeled samples for semi-supervised clustering. With exclusively unlabeled samples, we have no knowledge of existing classes except what is encoded in each sample's feature vector. With a semi-supervised approach, we add all labeled samples and enforce two constraints on them: must-link and cannot-link. In Figure 4.51, the must-link constraint shows how two labeled samples of the same positive class must be assigned to the same cluster. The cannot-link constraint shows how two samples with labels of a different positive class or a sample from a positive class and a sample with a negative label cannot be assigned to the same cluster. As shown, unlabeled samples of course have no restrictions on which clusters they are assigned to. As new classes are created during the mission, the clustering process will create separate clusters for them for each iteration thereafter.
3. Once the cluster assignments have been made, the final step is to select which samples should be transmitted to home for inspection by the domain expert. This step has several

options based on mission requirements. If the objective is to find samples of a potential near OOD novel class, then N samples would be selected from that cluster for transmission. Another approach available is to select two random samples from each cluster. We can also select N samples each from the top M clusters whose centroids are closest to the existing positive class(es). This would prioritize near OOD samples but select from across several clusters to allow for sample diversity.

The costs of edge-based clustering for OOD discovery in a Live Learning context are the computational resources on compute-limited scouts and additional network bandwidth to transmit samples of potential novel classes from scout to home for inspection by the domain expert. The computational load on the scout to run a parallel clustering process is dependent on 1) the number of total samples given as input to the clustering algorithm, 2) the clustering algorithm itself, and 3) how frequently the algorithm is executed. The expert must take these into consideration when choosing how to configure the mission. For example, the clustering algorithm should not execute so frequently that only a small number of samples have been added to the queue since the pervious iteration. This is also dependent on the inference rate on the scouts, which is a separate configuration according to the sensor, GPU available, data type, etc. It may also be impractical to cluster all new samples arrived since the previous iteration if the inference rate is too high. Ultimately, the expert should configure the clustering process such that the algorithm runs frequently enough to provide reasonably fresh samples to home, but not too frequently that each iteration starts immediately after the previous one. There must be sufficient time between each iteration to repopulate the queue with a certain number of samples, ideally more than the configured number of samples clustered for the next iteration. Thus, there are many variables and configurations to optimally balance based on user preferences as well as hardware and system constraints. Additionally, once a set of samples are selected to be sent to home, they do not consume the standard transmission slots from the default selection process. Rather, they are sent in parallel, consuming additional bandwidth above and beyond the current selection process. Therefore, if clustering is selected as the method for novel class discovery, either the transmission rate for the standard Live Learning process must be reduced to accommodate it or additional bandwidth must be allocated (if available) to the scout to support the transmission of additional samples selected by the clustering process to home.

4.12.4 Experimental Examples

The semi-supervised clustering approach provides the domain expert maximum flexibility in defining mission objectives. If it is expected that any novel class will be similar to existing threat classes, then focusing on near OOD cluster samples is ideal. In contrast, if both near and far OOD potential novel classes are expected, one can select samples to cluster from the entire input queue and select samples for transmission from all clusters regardless of their centroids' distance from the existing positive classes.

One example mission using our clustering approach used class roundabout of dataset DOTA as the existing threat class. Figure 4.52 depicts example samples that were selected upon completion of a clustering iteration and transmitted to home for inspection. Figures 4.52(a) and 4.52(b) as examples from a near OOD cluster. When roundabout is the existing threat class, samples containing intersections are frequently transmitted to home as they significantly resemble round-

(a) Intersection

(b) Storage Tank

(c) Boat

(d) Airplane

Intersections and storage tanks are considered near OOD data while boats and airplanes are considered far OOD data, compared to class roundabout of dataset DOTA.

Figure 4.52: Examples of Near and Far OOD Samples Selected by the Clustering Process

abouts. The same can be said about samples containing storage tanks based on their round shape and various sizes. Boats and airplanes in Figures 4.52(c) and 4.52(d) are examples of samples from far OOD clusters, much less similar to roundabouts as they are unique shapes and found in different background environments (water/airports).

The primary advantage of the clustering approach is that the scouts perform additional computational work determined by the domain expert at the beginning of the mission, yet are transparent to him during the mission. The expert only interacts with the samples selected from the clustering process based on the policy selected at the beginning of the mission. The standard Live Learning processes function as normal and the samples derived from the clustering process can be completely ignored by the expert. This prevents any meaningful increase in cognitive load on the expert. If finding novel classes is a core objective, by default he can perform quick visual inspections of many samples without having to label any of them. However, if one requires deeper

inspection and an actual label in pursuit of creating a new class, then slight additional cognitive load is required. Ultimately, it allows the expert to focus on the critical mission at hand: labeling as many roundabout samples as possible to generate improved models, while also providing additional unique samples both similar and different to roundabouts that can be inspected as a potentially novel threat class.

4.12.5 Summary

We introduced Novel Class Discovery for SCML systems that perform Live Learning. The ability to adapt to previous unknown threats present in adversarial environments is critical to the survivability of SCML systems. We discussed how such unknown threats can be both similar to and different from existing threat classes, representing near and far OOD data, respectively. Although it requires no additional computation on scouts, passive discovery is limited to near OOD data in that it relies exclusively on the likelihood of a sample being sufficiently similar to one of the existing threat classes through its prioritization in the transmission queue. Conversely, online semi-supervised clustering can be configured to select samples from clusters that are either near or far from the existing threat class clusters. It leverages a continuously growing set of labels of the current threat classes to anchor their respective cluster groups in feature space. With semi-supervised clustering, we enforce must-link and cannot-link constraints such that any samples labeled as the same class must be assigned to the same cluster and those labeled as a different class may not be assigned to the same cluster. Our approach allows for flexible configuration options based on operational requirements of the domain expert.

We demonstrated with experimental results how passive discovery is able to suitably adapt successive scout models to collect additional training samples of a novel class, dynamically discovered during the mission. The degree to which this approach is effective is dependent on the similarity between a potential novel class and the existing threat class(es). With online clustering, both near and far OOD samples can be sent to home for inspection and labeling. Many optional configurations can influence the nature of such samples depending on operational needs, such as clustering only near OOD samples or clustering a wide distribution of near and far OOD samples to generate diverse distribution of samples of potential novel classes.

Other methods can be explored and integrated into the Live Learning architecture to further enhance survivability of autonomous SCML systems in adversarial environments. For threats known a priori and those that are dynamically discovered during execution, Live Learning is clearly an invaluable enabler of SCML system survival due to its adaptability.

Chapter 5

Extending Live Learning for Radar Applications

5.1 Introduction/Background

Live Learning with Hawk was originally designed and evaluated exclusively for visual data in the form of RGB images in [18]. We deemed it necessary to extend its support to domains of data other than visual to demonstrate its extensibility. Validating the core concurrent functions like inferencing, transmission, labeling, and training on data from other parts of the electromagnetic spectrum can demonstrate the versatility of Live Learning and Hawk as the basis for SCML via diverse sensor modalities. The primary modifications to Hawk come in the traditional ML training and inferencing processes as they are the two primary components that interact with the internal representation of the data itself. As with visual spectrum images, non-visual data can still ultimately be represented as multiple channels of two-dimensional arrays of pixels. However, the difference in what features the pixels represent provide sufficient value in validating Hawk and Live Learning on such unique (radar) data. Radar data provides some location information as in a standard image but with lower resolution because it also depicts an object's velocity, discussed below in more detail.

The vast majority of academic literature in computer vision has focused on developing models capable of understanding data represented in the visual spectrum. Naturally, visual red-green-blue (RGB) images are snapshots of the environment in which humans perceive the visual world. Therefore, they represent the most commonly used data modality in the development and deployment of contemporary ML products and systems. Human eyes have evolved to perceive peak electromagnetic power at the red, green, and blue wavelengths within the visual spectrum. This band, however, is relatively narrow compared to that of the overall electromagnetic spectrum, as depicted in Figure 5.1 below. On the longer end of the wavelength spectrum, radio waves were first used by humans at the turn of the 20th century primarily for communications. As time has progressed, an explosion of devices and applications have been developed based on the unique properties of electromagnetic waves across many different bands. Legacy Amplitude Modulation (AM) and Frequency Modulation (FM) radio signals have been in use for over a century and other frequencies within various bands carry analog and digital television signals.

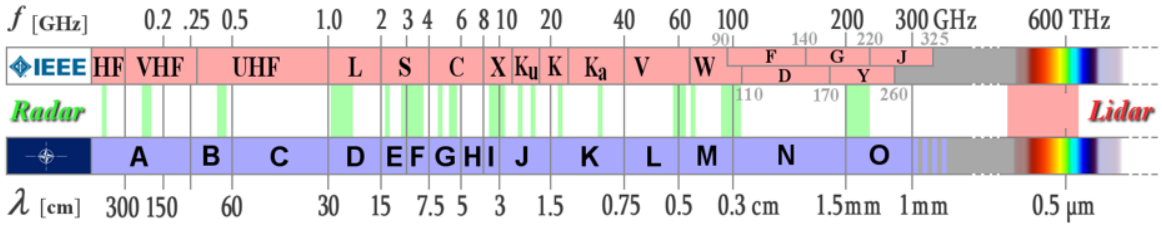
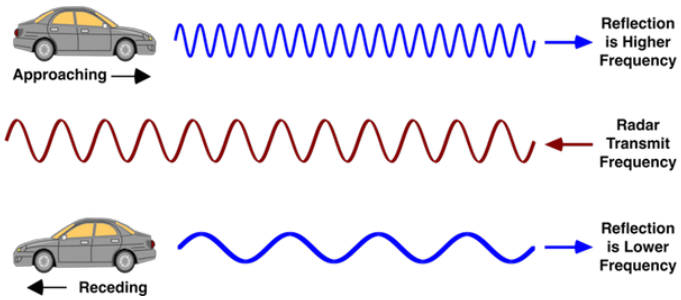


Figure 5.1: Electromagnetic Spectrum with Associated Radar-Specific Bands (Source: [107])

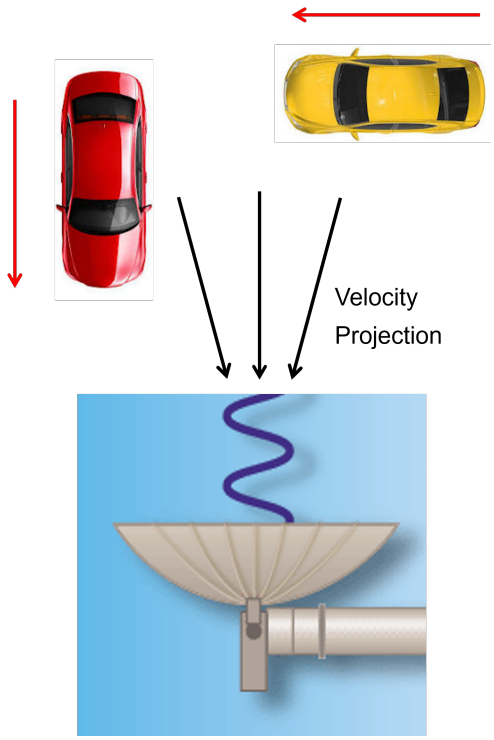
Exploration into other, smaller bands has enabled the widespread application of new systems. Radar is a primary example, with many different bands generally ranging from 5 MHz to 130 GHz in frequency, or approximately 60 meters to 2 μm in wavelength.

Radio Detection and Ranging (RADAR) was first used during World War II to detect the presence of aircraft at a range of approximately 80 miles. Its most common use in the contemporary ML context is with sensors for self-driving cars. Many more applications are enabled by radar technologies at a wide array of frequency bands, as shown by the green regions in the middle row of Figure 5.1. Energy in the visual band is passive in that it is simply ambient light reflected from objects perceived by human eyes. In contrast, radar is active in that a source emitter must transmit energy in a given direction and analyze the energy of the returning reflections to extract information from the environment. Unlike with visible light, this information contained within the reflected waves is not represented as energy at multiple wavelengths of human-interpretable color (e.g. red, green, and blue). Rather, it provides details of an object's range and position relative to the source emitter as well as the velocity of the object. Through standard wave propagation equations, properties of reflected waves like magnitude, frequency, and phase are compared to those of the emitted waves, ultimately resulting in a representation of objects in the scene. With additional mathematical transformations, the objects can be represented with features spanning the range, azimuth, and doppler (velocity) dimensions.

The Doppler effect [109] is a well-known phenomenon most commonly experienced audibly by people when trains pass by or an ambulance runs its sirens. The compression and expansion of sound waves as an object moves towards, and then away from, a stationary observer is what is known as "Doppler shift". As a train moves toward the observer, the sound waves are compressed and thus the frequency increases, resulting in a higher pitch. Likewise, as the train travels away from the observer, the sound waves received by the observer expand as their frequencies slowly decrease, resulting in a lower pitch. The same concept applies to electromagnetic waves, e.g. radar signals reflecting off a car in Figure 5.2(a). While many hardware and other manufacturing details dictate how specific radar systems function, here we describe how a generic radar system generates data. A source transmitter (typically) emits a Frequency-Modulated Continuous Wave (FMCW) in a general direction. Individual surfaces on the objects in the scene reflect energy from the wavefront back to the source, with modified amplitudes, frequencies, and phases. The source then computes complex-valued representations of the overall returned energy as a function of range (in meters), azimuth (in degrees), and velocity (in meters per second). Range and azimuth are computed by the distribution in amplitude and phase of the reflected energy. The velocity, or Doppler, component, is computed by calculating the Doppler shift in the frequency of the reflected energy. This velocity component only represents the velocity directly toward



(a) The car reflects compressed waves (higher frequency) when traveling toward the source. The car reflects expanded waves (lower frequency) when traveling away from the source. Source: [108]



(b) The Doppler component derived from the reflected power is maximum when the car is moving directly toward or away from the emitting source. If the motion of the car is mostly perpendicular to the emitted energy, the Doppler component will be close to zero. Thus, the Doppler component is a projection of the car's velocity on to the direction of the reflected waves back to the emitter.

Figure 5.2: Doppler Shift Visualizations

or away from the source (a projection) where an object that moves in a completely orthogonal direction to the incident wave will not cause a significant Doppler shift. Consider a train that is still audible from a significant distance from the observer. If the train moves right to left but because of the large distance away it remains at approximately the same angle from the observer, the pitch of the sound will not noticeably change. Figure 5.2(b) depicts two cars in motion. The red car moves almost directly toward the emitter where the projection of its actual velocity is close to the actual velocity, resulting in a higher Doppler shift. The yellow car moves mostly orthogonal to the emitted waves, resulting in a minimal velocity projection in the direction of the emitter. A single pixel's worth of information about part of an object is represented in the range, azimuth, and doppler dimensions. As in visual images, the combination of many pixels in a cluster represents the presence of a large object in the scene, such as a car, truck, or person. We can thus analyze images containing pixels representing radar data with current ML models

Number of Instances & Samples			
Class	Train	Test	Total
Person	5210	1280	6490
Bicycle	729	204	933
Car	13537	3377	16914
Motorcycle	67	21	88
Bus	176	38	214
Truck	3042	720	3762
Total Instances	22761	5640	28401
Total Samples	8126	2032	10158

Table 5.1: RADDet Dataset Class Statistics

as with RGB data. The next section describes the technical details of a specific radar dataset we used for Hawk integration.

5.2 Data Preparation and Preprocessing

To demonstrate Live Learning on radar data, we searched for a dataset that could be transformed into a compatible format that Hawk is able to process. While several candidates are sufficient in the number of classes, raw format, and overall number of samples, the final two that fit our requirements the best were the RADDet [110] and CARRADA [111] datasets. The statistics for the RADDet dataset are shown in Table 5.1. We focused our analysis and experiments on this dataset for multiple reasons. First, it contained the optimal class diversity with a sufficient number of samples per class to perform our experiments. For example, while care and truck are intuitively similar, even in the radar domain, the classes: car, truck, person, and bicycle are sufficiently diverse in their general feature representations. These classes also provide a plentiful number of samples for various experimental purposes. Second, dataset preprocessing and preparation for Hawk integration was the most seamless with respect to file format, storage, and minimal required transformation. Third, the original RADDet authors provided additional tools for use in visualizing radar data samples and other utilities. This section describes the data transformation process to enable Hawk to perform Live Learning with radar data.

Each sample in the RADDet dataset is represented by a three-dimensional array of complex-valued pixels as a function of range (meters), azimuth (degrees), and doppler (meters/second), an example of which is shown in Figure 5.3. The complex values describing each pixel are derived from the energy of the emitted and reflected waves from and to the source, influenced by specific hardware manufacturing properties as well as physical signal transmission configurations. There are a number of different approaches available to transform the raw complex data into a format that can be inferenced by ML models. We follow the same core procedures as with the original construction of the RADDet and other equivalent datasets. Each sample in the dataset is a raw `numpy` tensor of shape (256, 256, 64) in range, azimuth, doppler (RAD). The authors in [110] built a model architecture that could process samples for training and inference in which samples dimensions were represented by range (height of 256 pixels), azimuth (width of 256 pixels), and

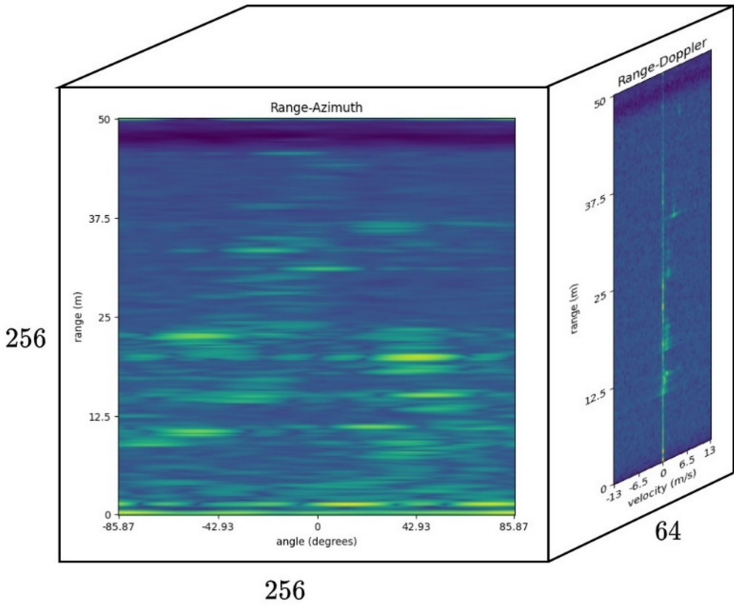


Figure 5.3: Raw 3D Range-Azimuth-Doppler Sample from the RADDet Dataset (Source: [110])

doppler (64 pixels, or channels). The primary goal across our two tasks of image classification and object detection is to accurately determine the class of the image, or object within an image. We determined that the 3D RAD image should be converted to a Range-Doppler map with three azimuth channels. Observe in Figure 5.3 how energy is smeared across many pixels for a single object.

This is much less pronounced in the range dimension because it is in the direction of the emitted signal itself. If we were to use azimuth as our physical dimension, the energy of all objects at roughly the azimuth position across all ranges up to 50 meters would be combined. This would complicate the model from extracting unique features from individual objects. Using range as our physical dimension with three azimuth channels, we can ensure proper feature extraction about objects with similar range-doppler representations. One channel may represent a car on the left side of the image while another channel may represent a car on the right side.

To convert the original samples into our desired dimensions and format for use in Hawk, we used a short sequence of steps depicted in Algorithm 2. First, in order to work with purely real data, we compute the magnitude of the complex data on line 1. To retain a small amount of information in the azimuth channel and avoid losing knowledge about discrete objects at different azimuth locations, we divide all 256 pixels of azimuth into three groups and assign each pixel of azimuth to one of $|\mathcal{C}|$ groups (channels) on line 2. (depending on the number of desired channels in each final sample). For each of these groups, we sum over the azimuth dimension to create three azimuth channels on line 3. Finally, we take the log to compress the full range of potential values on line 4. The result is a sample with 256 pixels of range, 64 pixels of doppler, and 3 azimuth channels. These derivative samples are similar to standard RGB images in overall dimension and thus require fewer online inference and training transformations. We call these samples Range-Doppler (RD) maps. RD maps are most commonly used in meteorological predictions but can be applied in many other scenarios like long-range aircraft detection and

Algorithm 2 Transformation from RAD to C-Channel RD Map

1: $L_{r,a,d} \leftarrow ||X_{r,a,d} + jY_{r,a,d}||_2, \forall r, a, d$ // convert from complex to real data
2: $c \leftarrow \left\lfloor a\frac{|\mathcal{A}|}{|\mathcal{C}|} \right\rfloor + 1, \forall r, a, d, c$ // assign azimuth slice to appropriate channel group
3: $L_{r,d,c} \leftarrow \sum_{a_c} L_{r,a_c,d}, \forall r, d, c$ // sum az. slices in group to create each channel of RD map
4: $L_{r,d,c} \leftarrow \log(L_{r,d,c}), \forall r, d, c$

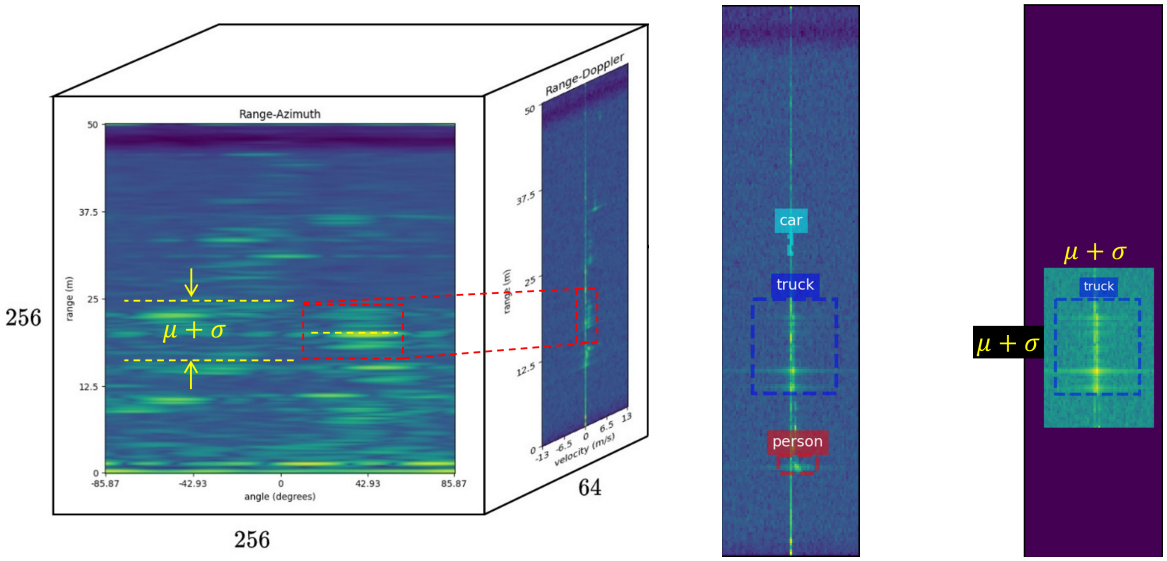
short to medium-range ground-based vehicle detection, as previously discussed. Just as any human with reasonable vision can identify most or all objects in an RGB image, domain experts can likewise identify objects by simply analyzing the energy contained within an RD map. This ability undoubtedly requires significant training and experience, but such experts are able to differentiate between airplanes and helicopters or cars and people with enough practice. Samples with a single ground truth instance can be used for image classification or object detection tasks while those with multiple instance are only used for object detection.

The core purpose of Algorithm 2 is to create a dataset for training and inference that can be processed by Hawk without unreasonable system modification. This ensures that Hawk retains its extensibility in supporting both visual and radar data. Upon the completion of the aforementioned transformation steps, these samples are suitable for object detection tasks as each RD map of shape (256, 64, 3) contains one or more object instances. However, many of them (those with multiple such instances) are not yet optimized for image classification tasks (which ideally contain a single instance of any class). Thus, we need one additional set of steps to accomplish this. It is important to note that we have the ground truth label coordinates for every instance in each RAD sample, used for computing the crop coordinates described in the next section. The RADDet dataset also provides an associated stereo image pair for each sample. This allows for the non-radar expert to gain an intuitive understanding of which object instance in a standard RGB image corresponds to an energy cluster in an RD map. It also provides a validation check to ensure that the coordinates of each instance in the RD map reasonably match the range and (likely) velocity of the object. Because the magnitude of the velocity is difficult to accurately

Mean and St. Dev. of each Class						
	Azimuth		Range		Doppler	
Class	μ	σ	μ	σ	μ	σ
Person	5	2	10	6	8	10
Bicycle	7	3	12	8	9	11
Car	16	6	25	18	18	18
Motorcycle	10	3	19	14	18	21
Bus	38	13	46	28	27	24
Truck	25	11	30	21	22	21

Statistics are derived from the widths, heights, and depths (in pixels) of all instance bounding boxes by class for the RADDet dataset.

Table 5.2: RADDet Dataset Physical Dimension Statistics (Source: [110])



(a) Original RAD sample depicting RA box projection and RD box projection along with the crop width in the range dimension ($\mu + \sigma$).

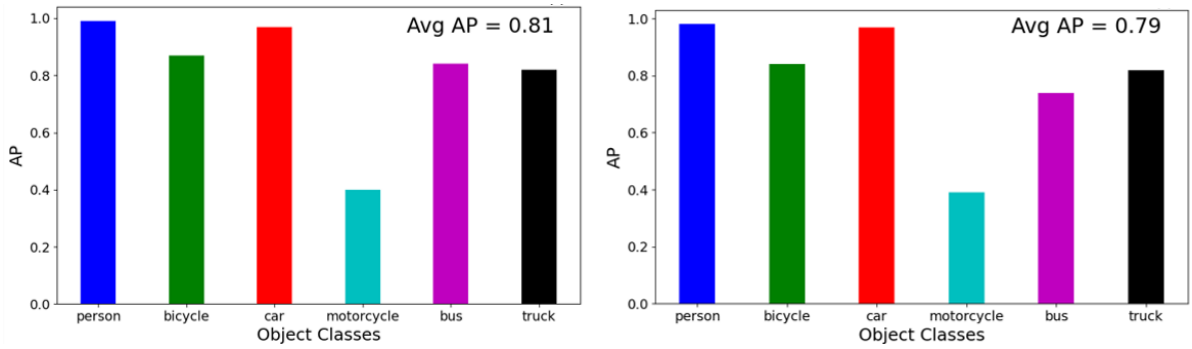
(b) RD map after transformation process.

(c) RD map cropped for the given truck instance.

Figure 5.4: Radar Samples at Different Phases of the Data Transformation Process

estimate from a single image, the general direction of object movement is generally straightforward to assess (object moving away has negative velocity and object moving toward has positive velocity). Therefore, the stereo image at least gives us a hint as to the approximate location of the energy in the RD map based on its estimated distance from where the image was taken and its general direction of travel.

To perform the extraction process, we first computed the mean and standard deviation of the widths of each dimension in the original 3D bounding boxes of all object instances by class across the entire dataset, as shown in Table 5.2. We then took the largest of the per class sum of the mean and one standard deviation ($\mu + \sigma$) to use as global values for cropping the overall image into derivative images containing only a single instance, depicted in Figure 5.4(a). Thus, our cropping width for the range dimension was $46 + 28 = 74$ and for the doppler dimension was $27 + 24 = 51$. Once the sample has been transformed into a three-channel RD MAP according to Algorithm 2, shown in Figure 5.4(b), all instanced from the original RAD cube are depicted in RD dimensions. The center of each crop is placed at the center coordinate of the ground truth label for the respective instance. A crop of the given width and height is then extracted from the original sample centered at this point, as shown in Figure 5.4(c) for the single truck instance. Finally, we pad the remaining pixels of the new cropped sample with zero values such that the final dimension is the same as the original (256, 64, 3). So, for example, an original sample containing three object instances of any number of classes is converted into three separate samples, each of which contain exactly one instance cropped from their respective bounding box locations and padded to ensure identical dimensions.



(a) Per Class Average Precision for ResNet18

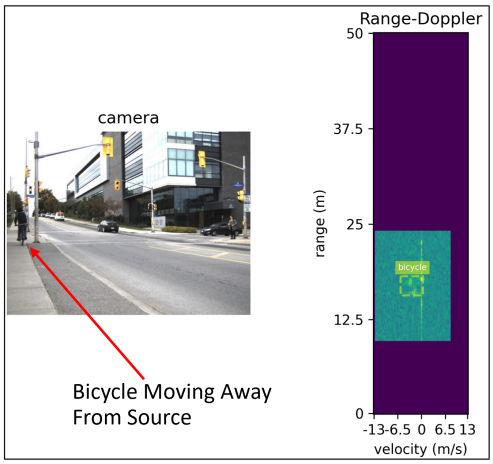
(b) Per Class Average Precision for ResNet 34

Figure 5.5: Base Model Performance Comparison

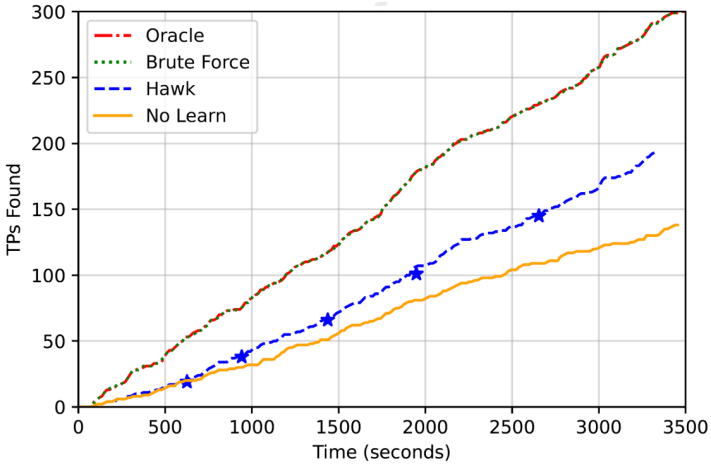
5.3 Classification with Live Learning

As mentioned in §5.1, performing Live Learning with Hawk on radar data requires no fundamental changes to its core functions. The only significant changes are the format of the radar data itself as well as the models that are trained on it and subsequently used for inference. Hawk was originally optimized to find the maximum number of true positives within a highly imbalanced streaming dataset ($\sim$0.1% base rate) by prioritizing likely positive samples for labeling and improving its models over time. Further, as mentioned, because Hawk was originally validated on visual data in [18], it leveraged a wide array of pre-trained benchmark classification models available for transfer learning in PyTorch [112]. However, while some existing literature trained ML models on radar data, there are no recognized benchmark models readily available in such common ML frameworks.

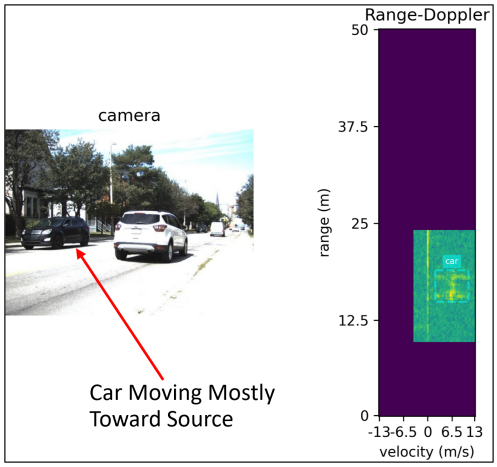
For visual data, Hawk trains a derivative bootstrap classification model on a small number of data samples assumed available prior to the mission. The goal, as with visual data, is to collect as many more samples of this positive class as possible during the mission. The bootstrap model leverages transfer learning from an existing model like ResNet and others pretrained on ImageNet (or other datasets), modifies it's architecture slightly, and trains it on the respective positive and negative samples. Because pretrained models for radar are not readily available, in order to leverage a similar style of transfer learning, we first had to train an equivalent "pretrained" or "base" model on relevant radar data. Hawk assumes the class of objects we are trying to find within a low base rate streaming dataset are not available for the pretrained model to be trained upon. Therefore, we do not want any knowledge of the target class embedded in the base model. Therefore, to create the radar base model for a given mission, we simply remove data samples of the class of interest and train on all other classes of data. Performance of example base models are shown for ResNet18 (Figure 5.5(a)) and Resnet 34 (Figure 5.5(b)) architectures, with Resnet18 achieving slightly better performance. We trained six separate base models (one for each class in the RADDet dataset) using the ResNet18 architecture, each one of which was trained on all samples from all classes except that which is the positive class for the given mission. For example, the *person* base model is trained on all samples but those of the *person* class, the *car* base model is trained on samples but those of the *car* class, and so forth. Now that we have the equivalent base models as that of one available direct from the PyTorch library, we can



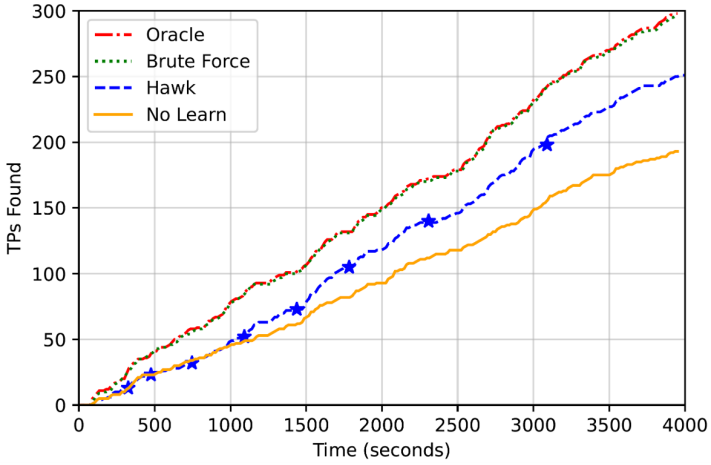
(a) Example instance for Class: Bicycle



(b) Classification Results for Class: Bicycle



(c) Example instance for Class: Car



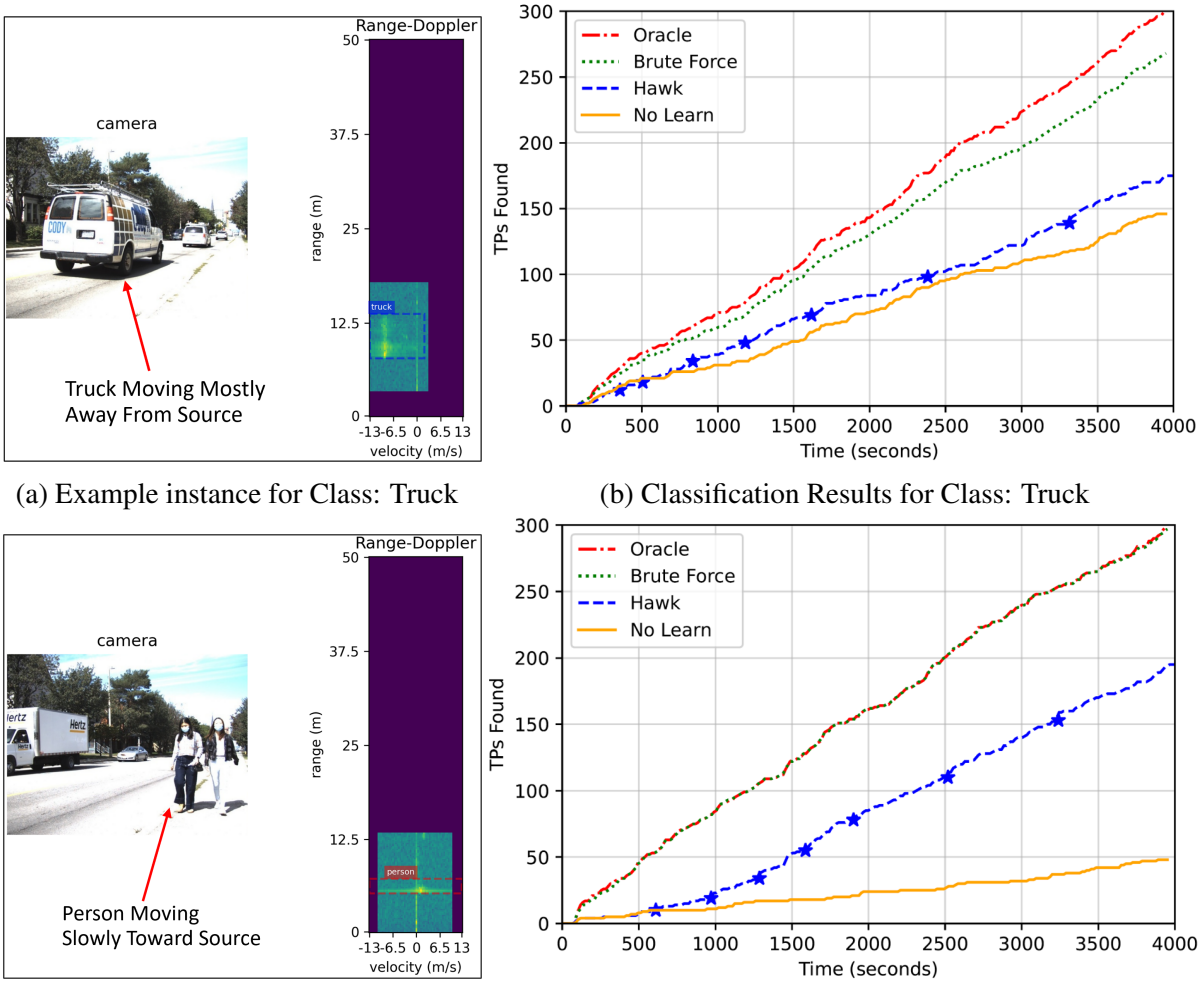
(d) Classification Results for Class: Car

Figure 5.6: Classification Results for Classes: Bicycle, Car

perform transfer learning from the base model to the bootstrap model at the start of the mission.

5.3.1 Classification Results

Now that we've defined how to produce the base model used for transfer learning on radar data, we discuss the results of standard Live Learning experiments with Hawk. Experiments for classification tasks are identical to that of those with visual data. Again, the only difference is in the models that are used: 1) the base model from which to derive the bootstrap model and 2) each subsequent model that is retrained, which is thus used for inference. The data itself the scouts expect to receive and process is also assumed to represent RD maps rather than RGB images. Figures 5.6 and 5.7 depict results for classes: bicycle, car, truck, and person. As with RGB missions, a streaming dataset is defined for the mission which contains a small number of samples of the positive class and a much larger number of negative samples of all other classes. For these experiments, we set the number of positives to be 300 to ensure sufficient analysis could be con-



(a) Example instance for Class: Truck

(b) Classification Results for Class: Truck

(c) Example instance for Class: Person

(d) Classification Results for Class: Person

Figure 5.7: Classification Results for Classes: Truck, Person

ducted on the results. Therefore, we did not run experiments for the bus and motorcycle class as they contained far fewer than a minimum of 300 samples available for experiments.

Results for each class have slightly different behavior. Similar to results with RGB data, we use the four experimental modes to analyze the value of Hawk: Oracle, Brute Force, Hawk, and No Learn.

- **Oracle**: This represents the ground truth number of positive samples that have been processed by the scouts (in aggregate) as a function of time. It is the theoretical limit in how many positive samples have been received at Home as the mission progresses.
- **Brute Force**: This experiment uses an offline-trained model on the 300 positive samples along with negatives that are expected to be processed during the mission. This is similar to overfitting in traditional ML where we test on the training data to ensure the model learns features correctly. This curve represents the theoretical maximum performance an ML model of the given architecture could achieve if using a model trained on all positive

samples from the streaming data.

- **Hawk**: This experiment uses a bootstrap dataset size of 20 positives and 100 negatives to train an initial model (via transfer learning) and retrain it periodically during the mission on samples prioritized according to their inference score and labeled at Home.
- **No Learn**: This experiment is identical to Hawk except that no additional learning (retraining) occurs during the mission once the initial model is trained.

Figures 5.6(a) and 5.6(b) depict an example RD map of a bicycle instance and results (the number of TPs discovered as a function of time) for the four experimental modes. The bicycle (pointed to by the red arrow) can be seen in the stereo image (left) traveling generally away from the source emitter, and thus the energy within the bounding box spans pixels representing negative velocity values (object moving away from the source emitter). In terms of total True Positives (TPs) by the end of the mission, Hawk improved over No Learn 193 to 138, which is approximately a 40% increase. Because Brute Force closely tracks Oracle, we observe that samples from the bicycle class are sufficiently unique in feature space compared to samples of all other classes. Therefore, training the Brute Force model on the 300 positive samples inferenced in the stream yields a highly accurate model. In order for Brute Force to track so closely to Oracle in this manner, each inference score is consistently close to 1, putting such samples at the head of the queue for transmission to Home.

In contrast to the bicycle example, Figure 5.6(c) shows the black car on the left side of the stereo image moving mostly toward the emitter, resulting in an instance with positive velocity in the RD map. The white car on the right side of the stereo image would be represented in a separate crop for the same original RD map, but it would be on the left side of the map where velocity is negative (moving away from the emitter). Figure 5.6(d) shows equivalent results, where again Brute Force closely tracks Oracle. The improvement of Hawk over No Learn was 252 to 193, which is approximately a 31% increase. This difference in value that Hawk provides is attributable to the fact that instances of cars and trucks are more similar to each other than they are to instances of bicycles. This is intuitive as both cars and trucks travel at similar speeds, have similar physical profiles, and similar structures, e.g. body, four wheels. Bicycles, on the other hand, have two wheels, a thin frame, a single person riding it. For the bicycle experiments, the models are better able to differentiate between bicycles and all other classes. For the car experiments, car instances must compete for transmission prioritization with all other classes, including trucks, which is more difficult.

As per Table 5.1, the car class has by far the greatest number of instances in the RADDet dataset. This results in fewer hard negatives (samples containing instances of other classes) present in the streaming dataset processed by scouts in the car experiments than the bicycle experiments. Positive samples containing a car instance have to compete with fewer hard negatives (approximately 9,200) for transmission prioritization than do positive samples containing a bicycle instance (22,000). The results is that the absolute number of TPs for both Hawk and No Learn are greater for the car results than they are for the bicycle results, as shown in Figure 5.6(d).

We see for the truck class in Figure 5.7(a) that a truck relatively close to the emitter generates a more dispersed energy signature in the RD map. Because cars and trucks are the two most similar classes in RD map features, the Brute Force curve in Figure 5.7(b) does not perfectly track Oracle. This is because car samples, which are the most frequent, more directly compete

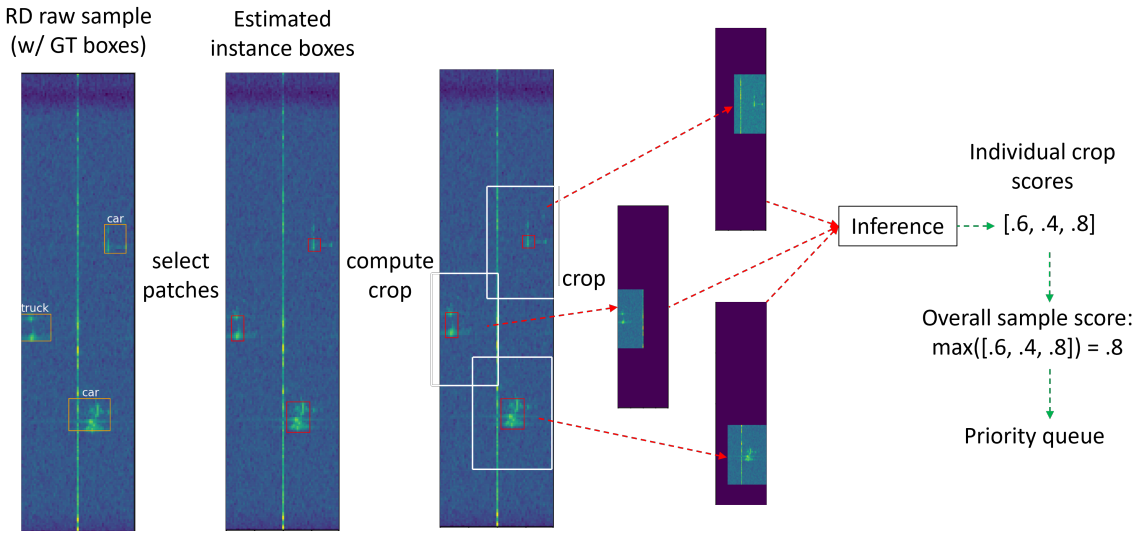


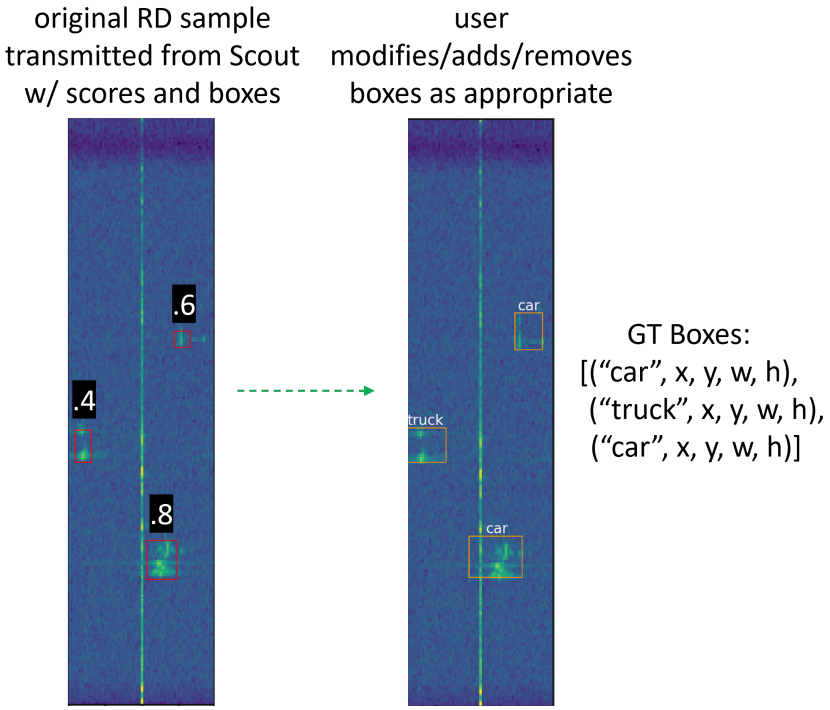
Figure 5.8: Patching and Classification Workflow for Object Detection

with truck samples in the priority queue for transmission. The Hawk curve remains mostly parallel to No Learn as retraining on more samples does not greatly increase model quality over time. The No Learn still performs relatively well at 146 due to the same problem that although the models may improve over time, cars and trucks are very similar and thus require far more samples to improve the latter models to a greater degree.

The results for the person class in Figure 5.7(d) show the clearest benefit of Hawk, which outperforms No Learn 196 to 48, a gain of about 308%. This is because the continued learning from samples beyond the original 20 positives in the bootstrap model clearly add value to the subsequent models. Also, instances of the person class are quite unique compared to the larger classes of car and truck, which results in far less confusion of the model when facing hard negatives containing instances of other classes. The person and bicycle classes also do not suffer the same differentiation difficult as car and truck due to greater difference in velocity and physical shape (body sitting with wheels compared to upright body). Here again Brute Force tracked oracle completely because training on the 300 positives seen during the mission saturated the ability of the model to learn any new features, resulting in almost all samples immediately being inserted at the head of the queue for transmission.

5.4 Object Detection with Live Learning

In traditional ML, the object detection task is slightly different than simple image classification. Most well-known detection models output an estimated set of bounding boxes, one for each instance in the image, along with class predictions for each instance. Because of the nature of radar data, specifically RD maps, there are generally fewer potential instances present in a single sample as reflections of energy off of stationary surfaces are consolidated around the origin of the Doppler dimension (vertical bright line down the middle of an RD map). Thus, the RD map effectively filters out stationary objects for which we generally have no interest in



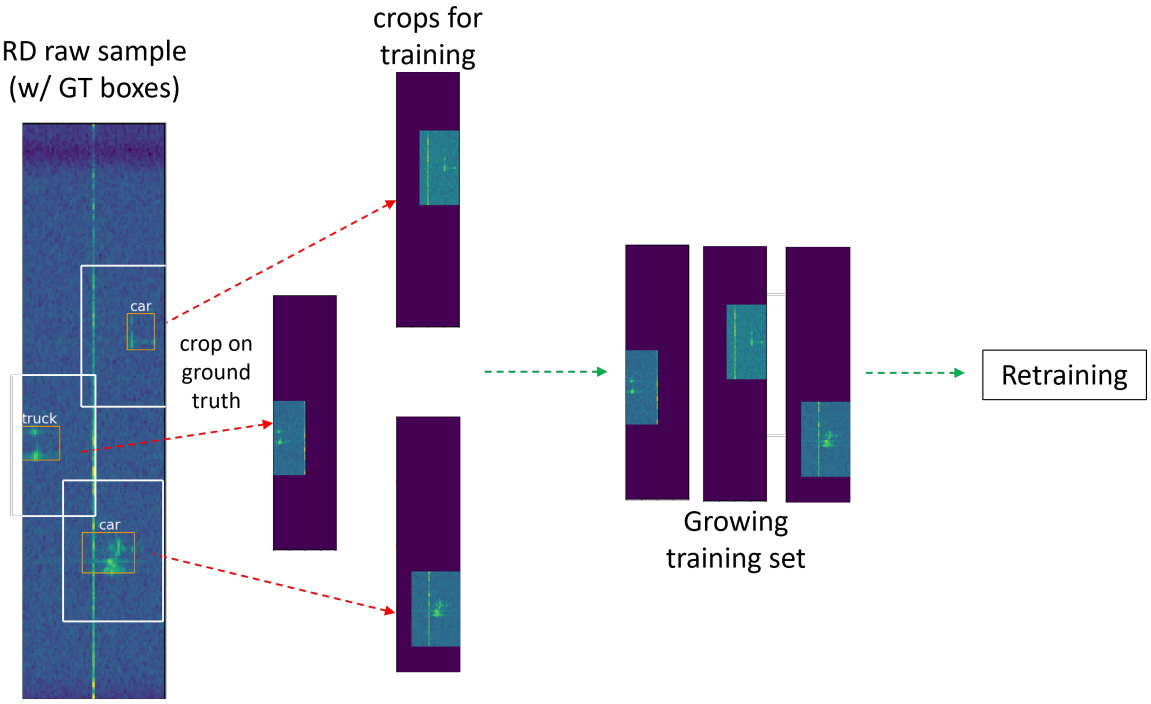
The labeling process at home where a sample that contains multiple potential instances with their respective scores are given a ground truth class and bounding box by a domain expert.

Figure 5.9: Labeling Process for Multiple Object Instances

detecting. Unlike with classification, we do not crop each of the separate instances out of the original images. We retain the entire 3-channel RD map containing all instances in the image for the detection task. As discussed previously, we take an approach to decouple the instance localization subtask from the classification subtask in order to allow for maximum modularity of localization algorithms within Hawk.

5.4.1 Localization Process

To perform the object detection task in a Live Learning context on radar data, we must execute a few additional steps beyond standard model inference. The first is part of the inference process at the scout. Once a sample is received by a scout, the localization algorithm estimates the set of regions of interest present (if any), as shown in Figure 5.8. The RD map on the far left is the original sample received as input to the localization or “patching” algorithm” where ground truths are displayed only for reference (during inference on the scout the GTs are unknown). The patching algorithm itself is a simple pixel clustering algorithm that groups adjacent pixels within a certain distance of each other that are above a threshold value. The second RD map depicts the red boxes computed from the patching algorithm to signify the approximate location where an object may exist. We then use the original crops widths and heights as described in §5.1 to crop each section containing a single patch from the original image, as seen in the third RD map. These crops are padded and added to the batch for inference as in the classification task. For a



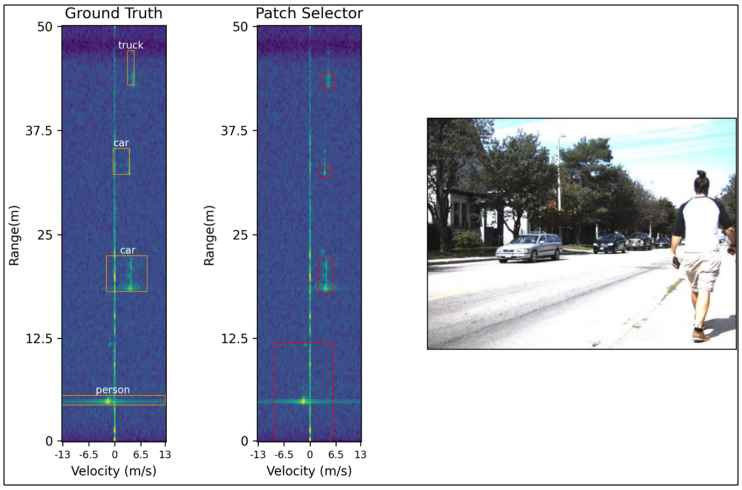
The process by which individual ground truth instances are cropped from original sample once a label arrives at a scout. The crops are then stored for future retraining.

Figure 5.10: Cropping Process of RD Map Once Label Received at Scout

single sample, we then take the maximum of the inference scores for these three crops, in this example, to determine the prioritization for transmission to Home, as shown on the far right. In summary, we compute likely regions of interest, extract crops of these regions determined by the patching algorithm, inference each of them with our existing classification model, determine the best score for its transmission priority, and transmit the original sample to Home for labeling.

Although a single score is computed for prioritizing transmission, the individual scores for each bounding box are retained for display at Home. The left side of Figure 5.9 shows an example of a sample received at home with its estimated bounding boxes (from the patching algorithm) and their associated classification scores. This information displays in the user interface where the user can adjust the position and sizes of the boxes and a class can be assigned. The ground truth boxes and class labels are depicted in the sample on the right side, at which point the labels are sent back to the scouts for future retraining.

Once a sample is received back at a scout, the ground truth instances are cropped out according to their coordinates, padded, and each cropped sample is stored for future retraining, as shown in Figure 5.10. This is because we need additional samples within single instances to further refine and improve our existing classification model. What is important to note here is that a single image may have multiple instances of more than one class within it. Thus, each of these samples is labeled appropriately after cropping but before storing locally to ensure accurate training set construction.

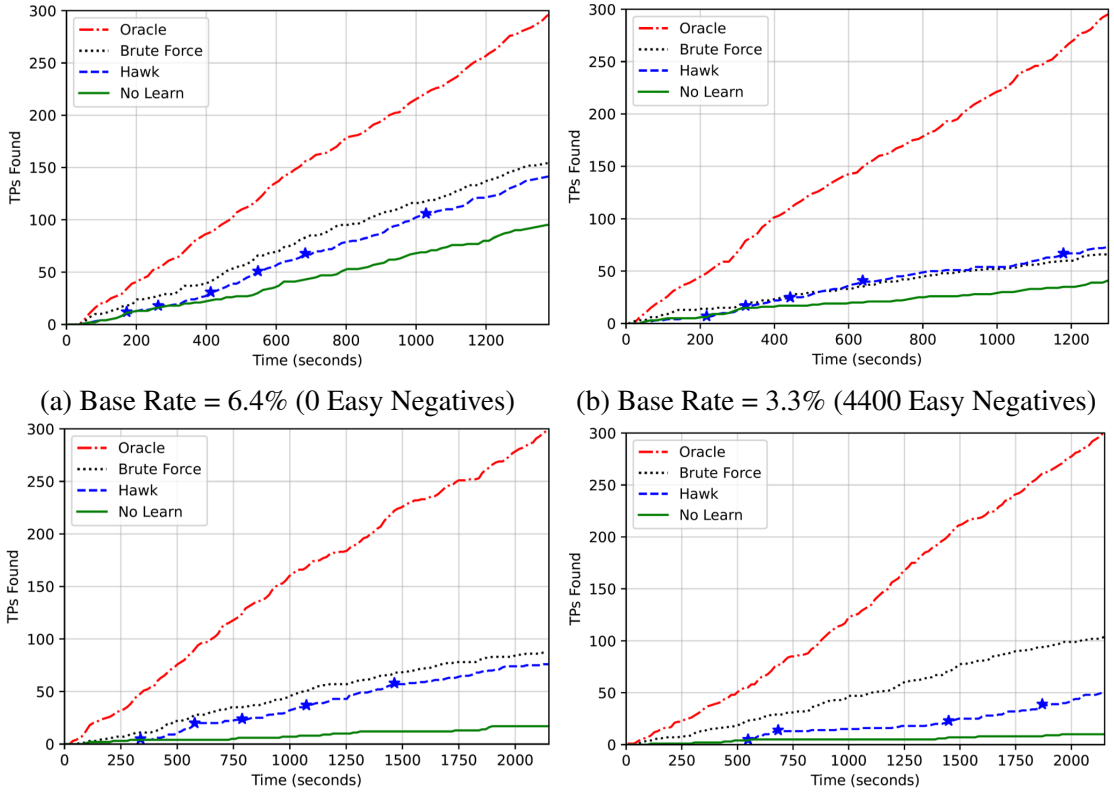


The figures depict an example ground truth detection sample and associated patching algorithm output with reference stereo image (Class: Person).

Figure 5.11: Ground Truth RD Map, Patching Output, and Stereo Image

5.4.2 Object Detection Results

We took a slightly different approach to evaluate detection experiments as compared to classification experiments. Primarily, we chose to evaluate the impact of the proportion of overall negative samples that were background samples. Background samples are samples cropped from original samples at locations where no objects are present. Thus, it contains only background noise. We performed this analysis with the person class, an example of which is show in Figure 5.11. The RD map on the left represents a sample that is received at the scout with ground truth boxes depicted for reference. The second RD map shows that the patch selector identifies potential objects in the image, regardless of class. Therefore, the classification model will inference four cropped samples for this specific original sample. There is only a single person instance in this original sample, which should score higher than the other instances, and as a result should influence prioritization for transmission. As with the classification experiments, we set the number of positive samples that contained at least one person instance to 300. A universal trend across all plots in Figure 5.12 is that the Brute Force curves are quite weak. This is primarily because the patching algorithm fails to identify all potential object instances in a given sample. The algorithm has a simple tradeoff between clustering too many pixels together, resulting in bounding boxes that potentially combine multiple instances, and clustering too few, resulting in no reasonably-positioned box or one that is much too small to represent a sufficient portion of an object. Future optimizations can improve the patching algorithm, which would likely bring all curves up to a greater number of TPs found. All plots in Figure 5.12 are derived from experiments using 300 positives, 4,400 hard negatives (samples containing at least one instance of any class except person) and a varying number of background (or easy negative) samples. We assess the impact of different numbers of background samples because in radar data there often may be samples sensed from the environment with no moving objects present. Thus, the increasing number of background samples may impact how Hawk is able to prioritize inferenced samples



(a) Base Rate = 6.4% (0 Easy Negatives)

(b) Base Rate = 3.3% (4400 Easy Negatives)

(c) Base Rate = 1.4% (17600 Easy Negatives)

(d) Base Rate = 0.68% (39600 Easy Negatives)

Figure 5.12: Live Learning for Object Detection with Various Base Rates

for labeling and model retraining. We use the base rate metric to describe the dataset imbalance as the percent of the whole dataset that is comprised of positive samples. Figure 5.12(a) has a base rate of 6.4%. Hawk (142) tracks Brute Force (155) closely throughout the whole mission, while No Learn (95) finds the fewest TPs. Hawk thus improves significantly beyond No Learn by about 50%. In Figure 5.12(b), we add background samples to decrease the base rate down to $\sim$ 3.3%. Here, Hawk increased the number of TPs found by 74% over No Learn, from 42 TPs to 72 TPs. Because the base rate decrease to $\sim$3.3% with the addition of easy negatives (background samples), Brute Force only achieves about the same as Hawk. In Figure 5.12(c), the base rate is reduced to $\sim$1.4%. The improvement of Hawk over No Learn greatly increases to 347% (76 compared to 17). This is because some background samples will get transmitted to Home and be labeled as negatives. As a result, the future retrained models will be trained on positives, hard negative, and easy negatives. This is optimal compared to training only on positives and hard negatives, where the model focuses on differentiating between subtle features between classes rather than balancing between subtle broader features. Secondly, the absolute number of TPs found in No Learn decreases (from 42 to 17) as TPs are competing with a greater number of total samples placed in the priority queue for transmission. Finally, with a base rate of 0.68% in Figure 5.12(d), Hawk improves by 400% over No Learn (50 compared to 10). But because of the even lower base rate, the absolute number of TPs found is the lowest across all four plots.

The overall theme we can take away from these charts is that with rising class imbalance

(lower base rate), the value that Hawk provides in iterative model improvement has an increased impact over No Learn: $50\% \rightarrow 71\% \rightarrow 347\% \rightarrow 400\%$. This is because retraining with a greater mix of positives, hard negatives, and easy negatives generates a stronger model that yields more accurate inference scores. We also observe that greater class imbalance, in most cases for Hawk but especially for No Learn, depresses the absolute number of TPs found due to the greater competition that positives experience by more background samples with respect to placement in the priority queue for transmission.

5.5 Conclusion

We demonstrated in this chapter how Live Learning with Hawk as a mechanism for SCML can be adapted to a non-standard data domain, such as radar, using Range-Doppler maps. Unique about this data type that it does not detect energy at different wavelengths with respect to 2D points in space, rather it detects such energy using active sensing to compute the velocity of an object at a given range from an emitting source. The primary system sub-components required some, but minimal adaptations as the main differences come in the model training and inference processes. To show such extensibility with Live Learning, we modified a publicly available dataset to convert into samples similar in dimension to standard RGB images for both image classification and object detection tasks. Further, we demonstrated its similar capability to iteratively retrain its models on real-time streaming data in a human-in-the-loop manner with Hawk as with visual data. This work clearly shows that with some data transformations and system modifications, the core functions of Hawk can potentially support a wide array of data domains and use cases when Live Learning is critical. SCML systems will surely operate in adversarial environments where weather, distance, and natural obstructions become barriers to sensing with high resolution in the visual domain. Thus, extending Live Learning to the radar domain is a clear first step toward developing a generalized Live Learning system for SCML. While SCML analysis is not explicitly performed with Live Learning radar experiments in this chapter, we have shown that the key enabler of it, Live Learning, is extensible to radar data.

Chapter 6

Modeling SCML as a Markov Decision Process

In previous chapters, we defined the SCML analytical model and provided experimental results that describe actual behavior of an SCML system leveraging Live Learning. We saw the impact on survivability of the countermeasure deployment threshold and the need to be able to adapt it to the current state of the mission. The process by which one or more SCML systems execute their mission is sequential in nature, and therefore in this chapter we can model an SCML mission as a Markov Decision Process (MDP). The goal of this chapter is not to provide a practical and ready implementation of an SCML MDP. It is rather to link SCML and Live Learning to other well-understood ML concepts such as MDPs and Reinforcement Learning (RL). In an SCML MDP, the objective is to maximize the probability of mission success by selecting the optimal threshold for launching a CM at each time step given the current state. MDPs are frameworks that model sequential decision making processes governed by state-action-state transition probabilities, associated rewards, and other characteristics. RL is an ML paradigm in which an agent learns optimal actions at each state in a sequential process through iterative feedback, as depicted in Figure 6.1.

The goal in sequential decision making processes is to maximize the agent's expected cumulative reward over time. The agent takes an action based on the current state of the environment. From this state-action pair, the future state and an associated reward (none, positive, or negative)

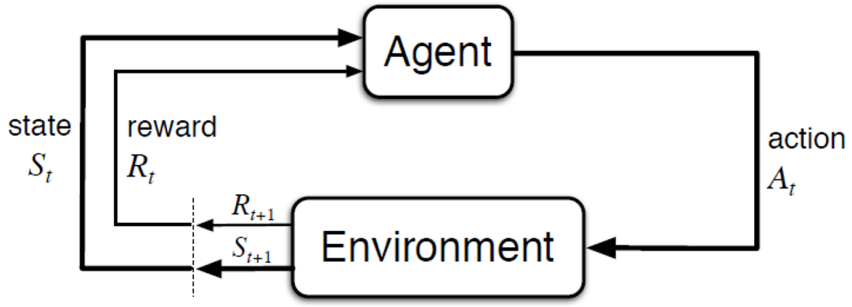


Figure 6.1: Agent-Environment Interaction in an MDP (Source: [113])

are derived. This process repeats for a definite or indefinite period of time until an absorbing, or terminal, state is reached from which the agent cannot depart. As stated, the purpose of modeling an SCML mission as an MDP is to compute the optimal threshold given the state of the mission. By adjusting the threshold as a function of state, we can improve the survivability of a team of SCML systems. MDPs have been applied to a wide variety of application domains that involve sequential decision making, especially with respect to autonomous systems where the geographic location, velocity, etc. are encoded as the state. While the aforementioned components (states, actions, rewards) describe the core functions of RL and MDPs, how they are applied and their special properties govern its operation. In the remaining sections of this chapter, we describe how to model SCML in an MDP framework with its unique properties. We also define methods for solving an SCML MDP in order enhance survivability and thereby maximize the probability of mission success.

6.1 Unique Characteristics of SCML MDPs

SCML is a new framework that analyzes and evaluates survivability of autonomous systems that heavily rely on model inference accuracy and judicious use of limited resources during missions of finite duration. Therefore we seek to answer the question: *What unique characteristics of SCML translate to an associated unique MDP framework?* We formally define the core components of the SCML MDP in detail in §6.2 but in this section we answer the question conceptually using some MDP-specific terms.

6.1.1 Finite Horizon

In previous chapters, we defined a successful SCML mission as one in which the system survives until the end of the mission, lasting a specific duration of time. The vast majority of autonomous systems will be limited to some extent in their operational endurance by their ability to carry on-board finite energy resources, e.g. fuel or battery power. As a result, in MDP vernacular our model must be analyzed as a *finite horizon* task. Conversely, some tasks may be indefinite or infinite in which the agent interacts with an arbitrarily large number of time steps. The goal of an SCML system is to survive the given mission duration, and therefore it must simply avoid destruction from start to finish. In most MDP tasks, the objective is for the agent to reach one or more specific goal states. However, in SCML the goal is to *avoid* a specific state - the *destroyed* state. Thus, the goal of an SCML system in an MDP framework is to avoid the destroyed state for all time steps in a finite horizon scenario (mission duration).

6.1.2 Undiscounted Reward

A derivative effect of defining mission success in binary terms (survival until mission completion or not) is in the modification of the reward discount factor. The agent's goal in any MDP is to maximize the cumulative reward. In most MDP frameworks, the reward is discounted as a function of time according to the *discount factor*, commonly represented as γ but defined as d in this chapter (we use γ for lethality in SCML). Rewards earned at earlier time steps contribute a

greater amount to the cumulative reward than do rewards earned at later time steps. Our goal in modeling the MDP is to maximize the probability that the SCML system avoids destruction for the entire mission duration, and therefore it weights rewards at every time step equally. Equation 6.1 depicts the default future cumulative reward at time t (G_t) on the left where a discount factor d^k discounts the reward at each future time step (R_{t+k+1}) and the simplified reward on the right when $d = 1$. The case when $d = 1$ represents an *undiscounted reward* where the reward for surviving each time step across the entire mission is uniform. This approach ensures that surviving the maximum amount of time (to the end of the mission) is implicitly the agent's objective.

$$G_t = \sum_{k=0}^{\infty} d^k R_{t+k+1} \rightarrow G_t = \sum_{k=0}^{\infty} R_{t+k+1} \tag{6.1}$$

6.1.3 Partial Observability

Observability in the context of MDPs refers to the agent's ability to perceive its true state in the environment. A fully observable MDP is one in which the agent's observation of the state of the environment is identical to the true state of the environment. By extension, this means the agent, in theory, can take the optimal action at each time step. However, in most realistic scenarios, the agent's observation of the environment does not perfectly match the true state of the environment. For example, agents often use multiple sensors to compute its exact geographic location, its current velocity, detect objects, etc. These sensors, even if precise, may provide only a partially accurate or incomplete representation of the current state of the environment. This is an example of a partially observable MDP, or POMDP. From the description above, the SCML MDP we define in this chapter would be considered fully observable in that the agent observes the true state of the environment at each time step. However, we will see in §6.2 that the manner in which a particular action is taken relies on external input (a classification inference score) and therefore our SCML MDP is considered a POMDP due to the external input required to determine the action at a given state. In other words, the agent cannot simply select the action itself from the optimal action distribution, rather it must receive external input to determine the action. The *action distribution* in an SCML MDP is defined in Equation 6.2:

$$\pi(s) = \{p(FP), p(TP), p(FN), p(TN)\} \tag{6.2}$$

for each state s. $\pi(s)$ represents the policy at state s, which determines the likelihood that each of the four actions is selected. The action actually selected for each sample is determined by the inference score and the current CM deployment threshold. How specifically each inference score influences this distribution will be discussed in greater detail in §6.2.

6.1.4 Stochasticity

When solving an MDP, the policy $\pi(s)$ represents the probability distribution over the actions given the observed state s, as per Equation 6.2. In most MDPs, the selection of the optimal action is deterministic - the optimal action is determined by that which results in the greatest

cumulative future reward:

$$\pi(s) = \underset{a}{\text{argmax}}\ q_\pi(s, a) \tag{6.3}$$

where $q_\pi(s, a)$ represents the cumulative future reward when in state s, taking action a, and following policy π thereafter. In the SCML context, each action is dependent on the new countermeasure deployment threshold and the inference score of the current sample, the latter of which is determined externally to the MDP algorithm itself. Thus, the SCML system (agent) cannot simply select the action according to the maximum q value using Equation 6.3, which is for deterministic policies. Instead, it selects the optimal threshold which yields the minimum L2 distance between its associated action distribution and that of the the optimal policy $\pi_*(s)$ for the given state s. This approach requires the MDP to be solved using a *stochastic* policy approach. More details on this process will be provided in §6.2. The practical result of this is that although one action may have a greater q value assigned to it than all others, it does not guarantee that it will be the action taken. The SCML MDP requires this stochastic action selection approach, which is depicted in Equation 6.4:

$$\pi(s) = \underset{a}{\text{softmax}}\ (q_\pi(s, a)) = \{p(FP), p(TP), p(FN), p(TN)\} \tag{6.4}$$

Using the solved MDP, the agent makes sequential decisions based on its current state by selecting the threshold that generates the minimum L2 distance between its associated action distribution and that of the optimal policy $\pi_*(s)$ for the given state s. Thus, there is a requirement to have some knowledge (a priori or online) as to how any individual threshold maps to a unique action distribution (described in detail in §6.3).

6.1.5 Stationarity

MDP stationarity is the property in which the fundamental characteristics of the environment do not change over time. Specifically, if the state probability transitions do not change over time, then the MDP is stationary, as shown in Equation 6.5.

$$P_t(s'|s, a) = P_{t+1}(s'|s, a), \forall a, s, t \tag{6.5}$$

As defined in §6.2, lethality is relevant to the transition values in this MDP. In this dissertation, we have treated lethality as constant throughout the mission. It is possible that in complex scenarios lethality could change over time, but we do not consider that situation in this work. We also define a stationary reward function, which states that the reward given for each state-action-state transition does not change as a function of time, as shown in Equation 6.6.

$$R_t(s'|s, a) = R_{t+1}(s'|s, a), \forall a, s, t \tag{6.6}$$

The SCML MDP is stationary as both the probability state transitions and the reward function do not change as a function of time. In summary, we have now defined the SCML MDP to be a Finite Horizon, Undiscounted, Partially Observable, Stochastic, Stationary MDP.

Item	Definition	Short Description
$\mathcal{S}$	$\{(c,t) \mid t \in \{0,1,...,T\}, c \in \{0,1,...,C\}\}$	State: CMs remaining and time step
$\mathcal{A}$	$\{FP, TP, FN, TN\}$	Set of all possible actions
$\mathcal{P}$	$P[s_{t+1} = s' \mid s_t = s, a_t = a]$	State probability transition matrix
$\mathcal{R}$	$R_s^a = E[r_{t+1} \mid s_t = s, a_t = a]$	Reward function
d	$[0,1]$	Discount factor

Figure 6.2: SCML MDP Components

6.2 Formally Defining the SCML MDP

An MDP is defined by a core set of components required to represent the environment, actions, reward, etc. Once these have been defined, various algorithms are used to solve the MDP such that an agent selects the optimal action (or samples from the distribution of actions for stochastic MDPs) given the current state. Any MDP is defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, d)$, each element of which is defined in Figure 6.2. $\mathcal{S}$ represents the set of all possible states in the environment, which consists of two dimensions: the number of countermeasures remaining (c) and the time step (the sample number) t. Thus, the total number of states is (CT), where C is the total number of countermeasures available and T is the mission duration (in number of sample inferences). The set of all actions $\mathcal{A}$ simply contains all possible outcomes of the inference score and threshold logical comparison. $\mathcal{P}$ represents the state probability transition matrix, which maps each state-action pair (s, a) to a future state s'. Similarly, $\mathcal{R}$ represents the reward function, which is the expected reward value of the next state given the state-action pair. Finally, d is the discount factor (normally γ in most RL literature), which discounts all future rewards by some factor.

Now that we have defined the fundamental components of the SCML MDP, we describe the unique states, actions, and transition probabilities, as shown in Figure 6.3. As mentioned previously, the objective of an SCML system is to avoid the *destroyed* state by avoiding or mitigating all potentially lethal threats over the course of a mission. In other words, the agent seeks to remain in the *alive* state after each inferenced sample. The agent begins the mission in the center green oval, where $c = C$ and $t = 0$. As the agent encounters (inferences) each sample, one of the four possible actions is determined by the current countermeasure deployment threshold and the inference score from the model. In all but two cases, the agent survives to the next time step. From left to right, the unique action outcomes (chance nodes) and their consequences are described below:

- TP when $c = 0$: When countermeasures have been depleted, even a correctly classified threat cannot be neutralized and thus is considered a potentially lethal threat. It is lethal (the agent transitions to the destroyed state) with probability γ (lethality), depicted by the dashed red line. Therefore, it survives with probability $1 - \gamma$ and returns to the default alive state.
- FP when $c = 0$: The sample is incorrectly classified as a threat when no threat is present, therefore the agent always returns to the default alive state.
- TP when $c > 0$: When countermeasures are still available, if the model correctly classifies

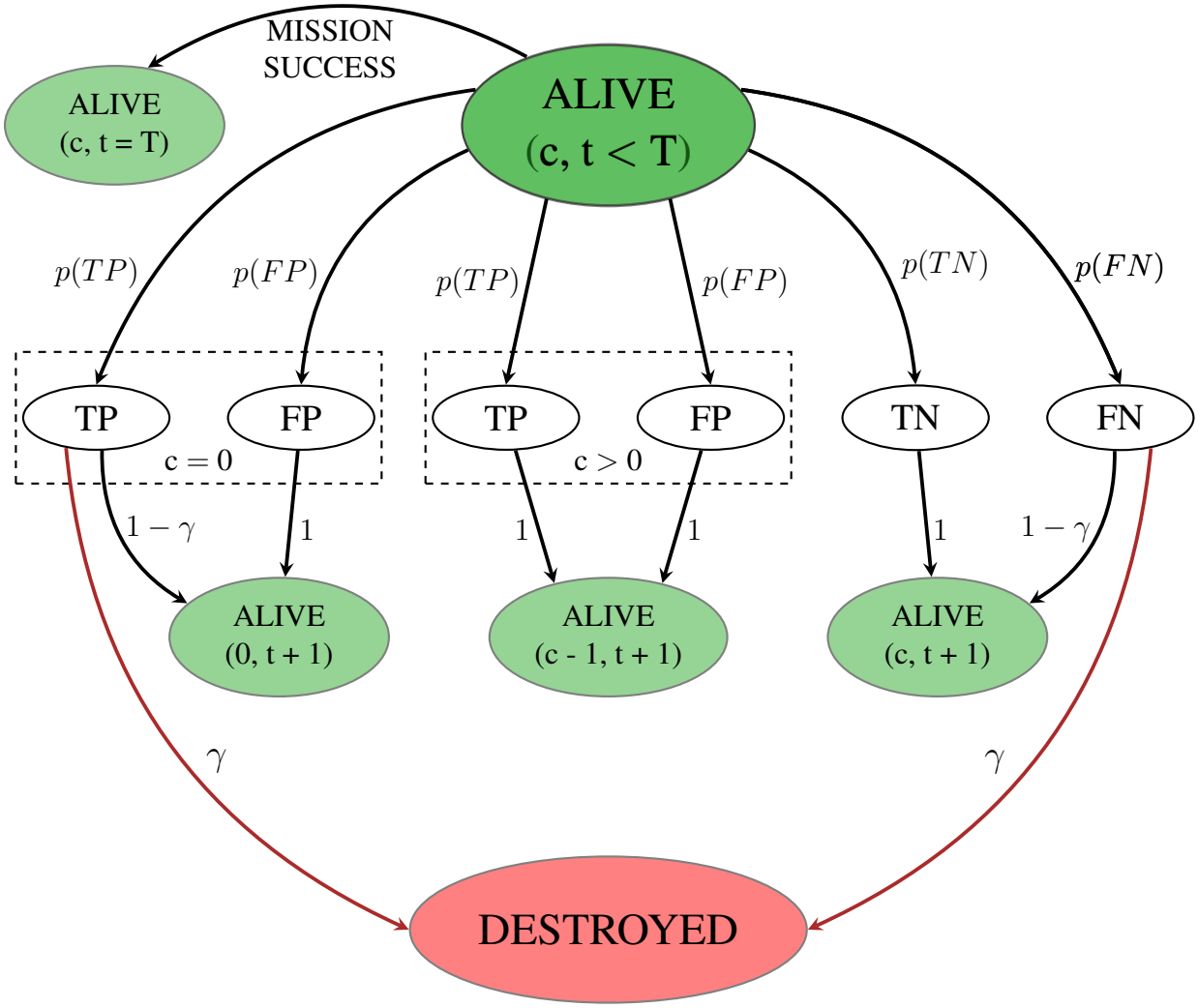
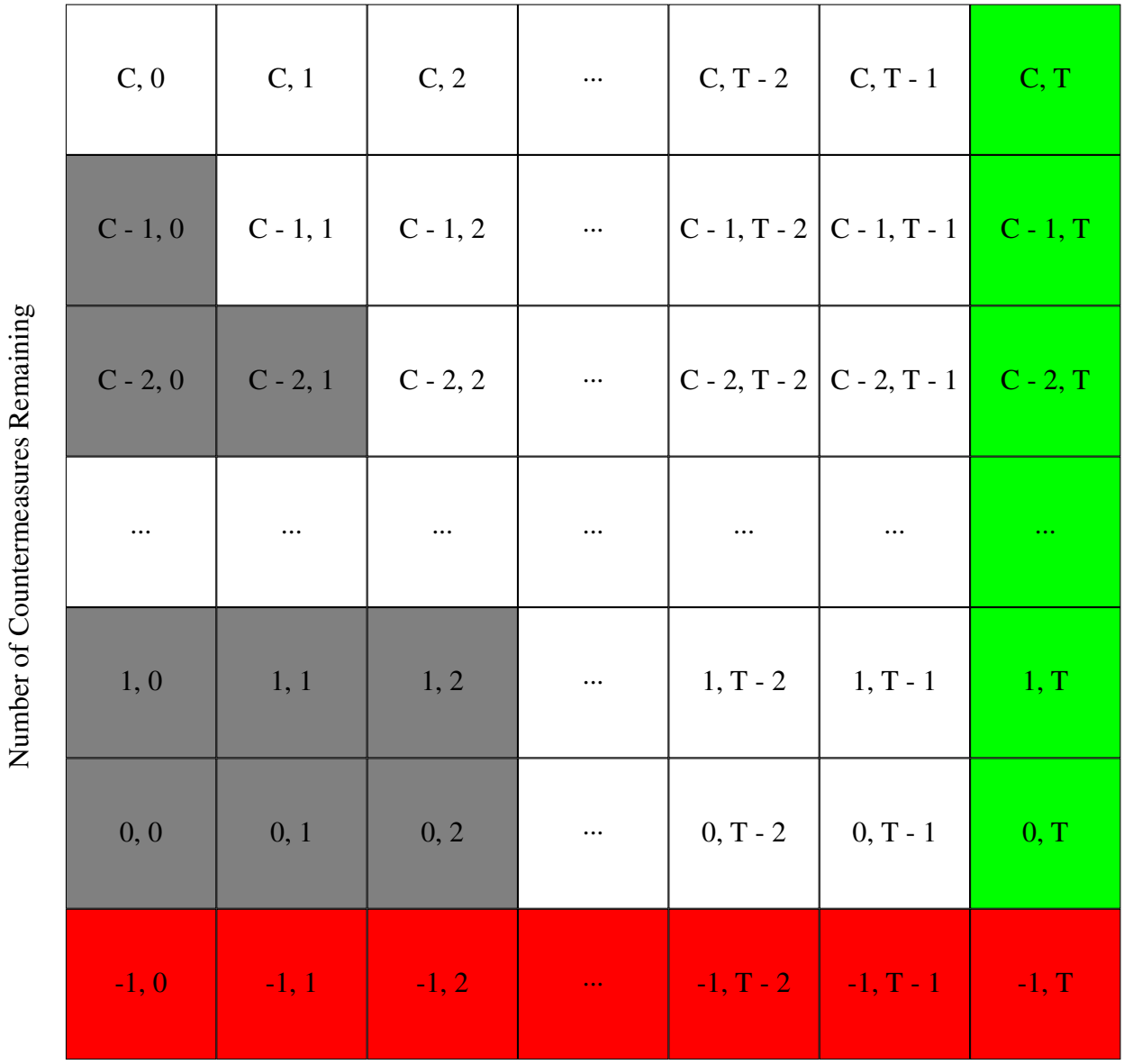


Figure 6.3: SCML MDP State Diagram

a present threat, a CM is deployed to neutralize it. As a result, it returns to the default alive state with one less CM.

- FP when $c > 0$: A sample is incorrectly classified as being a threat. The agent deploys (wastes) a CM and returns to the default alive state with one less CM.
- TN: For a correct classification when no threat is present, the agent returns to the default alive state.
- FN: The model incorrectly classifies a present threat, and therefore transitions to the destroyed state (threat is lethal) with probability γ. It returns to the default alive state with probability $1 - \gamma$, if it is not lethal.

After one of the four possible actions is determined (with six possible chance nodes), the next state can be computed. There are three possible next states relative to the current state of $(c, t < T)$ meaning the agent has not completed the mission and it has anywhere from $0 \rightarrow C$



Each box represents a unique state in the form of (number of countermeasures remaining, time step) or (c, t) for example. C is the number of initial countermeasures available at the beginning of the mission and T is the mission duration.

Figure 6.4: SCML MDP State Space

CMs remaining. If the agent currently has 0 CMs and the action is a TP, it transitions to the destroyed state with probability γ. It transitions to the state $(0, t+1)$ (the next time step) with probability $1-\gamma$ after a TP and always after an FP. Similarly, if the agent does have at least one CM available and it receives either a TP or FP, it always transitions to the state $(c-1, t+1)$ as it deploys one CM. The agent always transitions from the current state to the state $(c, t+1)$ if

the action is a TN and with probability $1-\gamma$ if the action is an FN. It transitions to the destroyed state with probability γ if the action is an FN.

Once the agent reaches the state (c, $t = T$), it has successfully completed (survived) the mission. This state is represented by the green oval in the top left of the figure. The destroyed state and the alive state when $t = T$ are the two terminal states. As we will see later in a visualization of the MDP state space, there are technically many more terminal states as the agent can complete the mission with various numbers of remaining CMs but a single successful terminal state suffices for this diagram.

While Figure 6.3 shows the states, actions, and state transition probabilities, Figure 6.4 depicts the two-dimensional state space in more detail. The x-axis represents the time step, equivalent to one sample inference, and the y-axis represents the number of remaining countermeasures. The last column on the right with the green background represents all possible mission success (terminal) states. Regardless of how many CMs remain, the agent must survive by avoiding the destroyed state until time T, the duration of the mission. The bottom row with the red background represents all possible destroyed states. Although the destroyed state is terminal in that the agent cannot leave the state, it is depicted here with $c = -1$ to visually include it in the diagram. It constitutes an entire row as the agent could enter the destroyed at any time step. Cells with gray backgrounds depict unreachable states, that is, it is impossible for the agent to ever reach such states. This is because an agent can never deploy more countermeasures than the current number of inferenced samples (time steps).

The initial state is always the cell in the top left corner, in the state $(C, 0)$. In this diagram, C is the total number of countermeasures available at the start of the mission and T is the mission duration. As the x-axis represents time, the agent always moves exactly one column to the right at each time step. However, it can remain in the same row, drop to the row below, or drop to the bottom row (destroyed state) depending on whether it deploys a CM or is destroyed. As described previously, the agent's objective is to reach any cell in the green column by avoiding transitioning to a cell in the red row. Intuitively, it is generally optimal to remain in the highest row possible (retaining the maximum number of CMs) as it reduces the risk of depleting them prior to time T. Depleting all CMs puts the agent in the $c = 0$ row, which causes TPs to be potentially lethal in addition to FNs, increasing the risk of destruction and mission failure.

The comprehensive probability state transitions and reward function are shown in Figure 6.5. The first two columns depict a given (s, a) pair, the current state and the action taken (in the SCML MDP the output of the threshold and inference score logic). The third column lists all possible future states (s') that can potentially be reached from the current state-action pair (s, a). The fourth column represents the probability of transitioning to future state s' given the current state-action pair (s, a). Finally, the fifth column shows the reward associated with transitioning to future state s' given the current state-action pair (s, a). Rows where the transition probability is zero simply means that the given state-action pair will never result in a transition to that state while a probability of one always results in a transition to that state (reference Figure 6.3 for visualization). The reward function is rather simple in that any time the agent avoids transitioning to the destroyed state (where $c = -1$), it receives one unit of reward. This approach ensures that surviving each time step in the future is just as valuable as surviving the current time step. The agent seeks to "live to fight another day" as it is incentivized to survive as long as possible. The bottom four entries in the table represent the fact that once in the destroyed state,

s	a	s'	$p(s'\|s,a)$	$r(s'\|s,a)$
$(c>0,t)$	FP	$(c,t+1)$	0	1
		$(c-1,t+1)$	1	1
		$(-1,t+1)$	0	0
	TP	$(c,t+1)$	0	1
		$(c-1,t+1)$	1	1
		$(-1,t+1)$	0	0
	FN	$(c,t+1)$	$1-\gamma$	1
		$(c-1,t+1)$	0	1
		$(-1,t+1)$	γ	0
	TN	$(c,t+1)$	1	1
		$(c-1,t+1)$	0	1
		$(-1,t+1)$	0	0
$(c=0,t)$	FP	$(c,t+1)$	1	1
		$(-1,t+1)$	0	0
	TP	$(c,t+1)$	$1-\gamma$	1
		$(-1,t+1)$	γ	0
	FN	$(c,t+1)$	$1-\gamma$	1
		$(-1,t+1)$	γ	0
	TN	$(c,t+1)$	1	1
		$(-1,t+1)$	0	0
$(c=-1,t)$	FP	$(-1,t+1)$	1	0
	TP	$(-1,t+1)$	1	0
	FN	$(-1,t+1)$	1	0
	TN	$(-1,t+1)$	1	0

Figure 6.5: SCML MDP Probability State Transitions & Reward Function

the agent remains in the destroyed state for every future time step (for purposes of experimental consistency - in realistic scenarios the mission ends when the system is destroyed).

6.3 Solving the SCML MDP: Model-Based Offline Approach

Most MDP solution techniques are grouped into two broad categories: Model-based and model-free. Model-based approaches assume the environment is fully observable in that the agent has complete knowledge of its current state, the transition probabilities from a given state and action to the next state, and the reward function. Model-free approaches assume the agent does not have access to this information and therefore relies on an exploration/exploitation strategy where the sequence of actions taken through the state space yields some unknown unit of reward. For model-free approaches, the agent must execute many episodes of the task in order to learn the state transition probabilities and reward function in order to ultimately select the optimal action at each state. In this work, we focus our work on the model-based approach.

The two most common types of algorithmic approaches to Model-based MDP analysis is Value iteration and Policy iteration. Value iteration finds the optimal state-value function across all states, which represents the expected cumulative reward at each state. This is useful in many applications, but in SCML we must compute the optimal policy.

6.3.1 Stochastic Policy Iteration

To perform policy iteration, we first introduce additional required terms and equations in this section. The following derivations in this section are directly from Sutton and Barto [113]. The state-value function is shown in Equation 6.7. The third and fourth lines are derived from the first two lines according to Equation 6.1, which defines the cumulative future reward, G_t. The function is recomputed after each policy update in the policy iteration algorithm, described in detail later. It defines the expected total future cumulative reward if the agent is in state s and follows policy π. As shown, the value of all possible future states (s') is added to the immediate reward of arriving in state s' and weighted by the state transition probabilities. The action probability distribution of the current policy then determines the final state-value. Ultimately, $v_\pi(s)$ is a matrix with $|\mathcal{S}|$ elements, or the total number of states. Equation 6.7 represents the Bellman equation for v_π [113]:

$$\begin{aligned} v_\pi(s) &= E_\pi[G_t \mid S_t = s] \\ &= E_\pi[R_{t+1} + G_{t+1} \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)\big[r + E_\pi[G_{t+1}|S_{t+1} = s']\big] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\big[r + v_\pi(s')\big], \forall s \in \mathcal{S} \end{aligned} \tag{6.7}$$

The action-value function, defined in Equation 6.8, is a more granular version of Equation 6.7. It is similar to the state-value function except it computes a vector of length $|\mathcal{A}|$ for each state s. Each element within the vector represents the expected cumulative reward for an agent taking action a from state s and following policy π. As shown in the bottom two lines of the equation, it is not dependent on the current policy π.

$$\begin{aligned} q_\pi(s, a) &= E_\pi[G_t \mid S_t = s, A_t = a] \\ &= E_\pi[R_{t+1} + G_{t+1} \mid S_t = s, A_t = a] \\ &= \sum_{s'} \sum_r p(s', r|s, a)\big[r + E_\pi[G_{t+1}|S_{t+1} = s', A_t = a]\big] \\ &= \sum_{s',r} p(s', r|s, a)\big[r + v_\pi(s')\big], \forall s \in \mathcal{S}, a \in \mathcal{A} \end{aligned} \tag{6.8}$$

The goal of policy iteration is to converge to an optimal policy π_* that maximizes the expected future cumulative reward from the given state. For a deterministic problem example, if the optimal action-value function at state s is $q_*(s, a) = \{2, 4, 1, 2\}$, the policy for state s, $\pi_*(s)$,

Algorithm 3 Stochastic Policy Iteration

Require: CMs: countermeasures, T: mission duration, γ: lethality

1: Initialization
2: $Q_\pi(s,a) \leftarrow 0$, $V_\pi(s) \leftarrow 0$, and $\pi(s) \leftarrow$ random([0, 1]) for all $s \in \mathcal{S}$
3: **while** $|\pi(s) - \pi'(s)| >$ tol **do**

4: Policy Evaluation
5: $V_\pi(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\big[r + V_\pi(s')\big], \forall s \in \mathcal{S}$ *// Compute state-value funct.*

6: Policy Improvement
7: $\pi'(s) \leftarrow \pi(s)$
8: $Q_\pi(s,a) \leftarrow \sum_{s',r} p(s',r|s,a)\big[r + V_\pi(s')\big], \forall s \in \mathcal{S}, a \in \mathcal{A}$ *// Comp. action-value funct.*
9: $\pi(s) = \text{softmax}_a(Q_\pi(s,a))$ *// Convert action values to action probability distribution*

10: **end while**
11: **return** $\pi \approx \pi_*$

is a one-hot vector, e.g. $[0, 1, 0, 0]$, computed from Equation 6.9:

$$\pi_*(s) = \underset{a}{\text{argmax}}\ q_*(s,a) \tag{6.9}$$

In this simple example, action 2 will always be taken. Our problem is stochastic, and therefore the optimal policy at state s is not represented as a one-hot vector but rather a probability distribution over the actions. As a result, our final optimal policy π_* for any state must also have a distribution derived from a softmax function rather than an argmax function, as shown in Equation 6.10. Using the same $q_*(s,a)$ for state a as above, the optimal (stochastic) policy at state s would be $[0.1, 0.76, .04, 0.1]$.

$$\pi_*(s) = \underset{a}{\text{softmax}}\ q_*(s,a) \tag{6.10}$$

Algorithm 3 depicts the Stochastic Policy Iteration algorithm. Two core steps are taken during each iteration: Policy Evaluation and Policy Improvement. First, the action-value function and the policy are initialized on line 2. It runs alternating iterations of policy evaluation and policy improvement until the policy changes less than a small tolerance. The policy evaluation step on line 5 computes the new state-value function for each state according to the current state-value function and the current policy. Conceptually, this step evaluates the current policy by determining the value of every state. For policy improvement, the current policy is stored as the old policy on line 7. On line 8, the action-value function is computed for all state-action pairs. On line 9, the policy is updated by computing the softmax over the action values of the given state. Thus, for the SCML MDP (as in Equation 6.2), $\pi(s) = \{p(FP), p(TP), p(FN), p(TN)\}$. The algorithm returns the optimal policy. Other parameters are required as input to the algorithm are the following:

- Lethality (γ): used in the state transition probabilities, as shown in Figures 6.3 and 6.5.

- Reward function: a reward of 1 unit is given if the agent moves to the next time step (regardless of countermeasure deployment) and 0 if it transitions to the destroyed state. The comprehensive list is contained within Figure 6.5.
- Probability State Transitions: The probabilities of transitioning to any state given the current state and action taken, shown in Figures 6.3 and 6.5.
- Time Steps: The number of inferenced samples during an SCML mission. A single time step represents a single inferenced sample, resulting action, and state transition.
- Total Countermeasures: The mission starts at $t = 0$ with all countermeasures available and slowly deploys them over the course of the mission. The product of the number of time steps and number of countermeasures determines the size of the state space.

6.3.2 Optimal Policy Action Probabilities by State

Figure 6.6 depicts the action components of an example optimal policy as a function of state. These figures are derived from the output of Stochastic Policy Iteration defined in Algorithm 3 and do not depend at all on ground truth action distributions of an actual SCML mission. The only input parameters that affect these results are the probability transition matrix, reward function, and lethality. Each heatmap (specifying a specific action) represents its respective action probability at each state (number of remaining countermeasures and the time step, which is implicitly the time until mission completion). The vertical bar to the right of each heatmap depicts the optimal probability that the respective action be taken at the given state. The black triangle on the bottom left of each plot depicts the unreachable state space where the number of countermeasures deployed is greater than the number of elapsed time steps. The probability values are an indirect reflection of the value that each action yields (derived from the action-value function). However, because the action values are normalized with a softmax function, the absolute probability values on the [0, 1] scale are less important than their values relative to each other.

Immediately we observe that FNs have the lowest value across all four actions early in the mission. This is intuitive because while CMs remain, it is the only action that has any probability of transitioning the agent to the destroyed state. On the other hand, TPs early in the mission are likely to be neutralized by available CMs. The FN plot also shows us that as the agent moves closer to mission completion, the value begins to increase. This is because the lost future cumulative reward from a potential FN becomes less and less the closer the agent moves to mission completion. For example, on the very last time step, if $\gamma = 1\%$, an FN would result in a reward of $1 * 0.99$ whereas any other action would result in a reward of one. Therefore, the relative reward outcomes for the final step are almost identical. We also compare the bottom of the TP and FN plots where $c = 0$. The probability of taking action FN or TP is equal across this row because either type of threat can be lethal and send the agent to the destroyed state. The TP and FP plots are mostly identical as their effect is largely restricted to the deployment of countermeasures, which is an indirect threat to the system when depleted. As a result, the FP and TN values are equal here and and have much higher action probabilities than TP and FN as the former guarantees survival at least until the next time step. To summarize, these heatmaps show a simple example of a computed optimal policy. The state space, lethality, and reward function

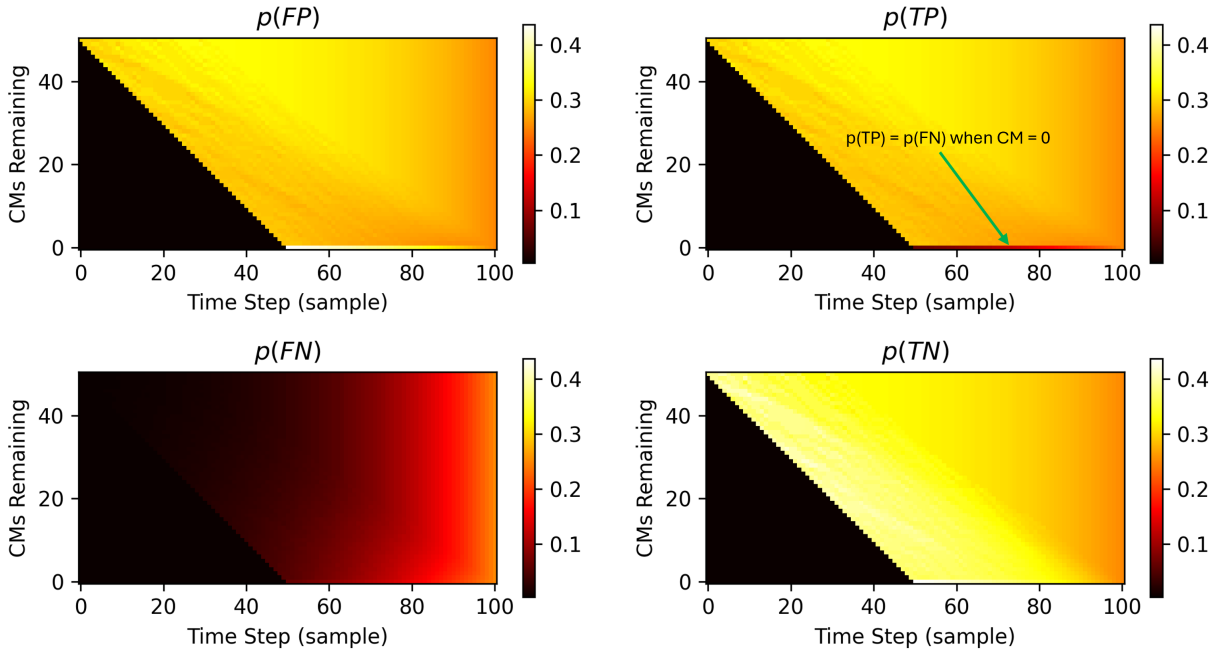


Figure 6.6: Optimal Policy as a Function of Remaining Countermeasures and Time Step

can all have unpredictable impacts as to the resulting optimal policies and their visualizations.

In this section, we have defined the complete SCML MDP model and the associated Stochastic Policy Iteration in Algorithm 3 to solve for the optimal policy as a function of state. From this, we can now demonstrate how to compute the optimal countermeasure deployment threshold at each state. In each of the following subsections, we take a unique approach and describe each step in detail.

6.3.3 Static Threshold Action Distribution

We begin with the simplest approach. An SCML agent is assumed to have a priori knowledge of the exact action distributions, which requires offline ground truth information of all samples. In real-time systems, we may not have access to this information but for experimental purposes and specifically for this approach, we assume it is available (we can compute these distributions by performing an SCML mission and extracting all inference scores offline).

Threshold Action Distribution Map

The threshold action distribution map links each threshold value θ to its respective action outcome distribution $B(\theta)$, an example of which is shown in Figure 6.7. Each row in the table represents a single mapping from one threshold value to the distribution of the four action outcomes, defined by Equation 6.11. These values are computed simply by dividing the total number of each outcome, e.g. total TPs, by the total number of inferenced samples. Therefore, when the threshold is zero, $p(FP)+p(TP)=1$ and $p(FN)+p(TN)=0$, and reversed when the threshold is one. We note that in SCML we assume the scenario of extreme class imbalance where

θ	$p(FP)$	$p(TP)$	$p(FN)$	$p(TN)$
0.00	0.999	0.001	0	0
0.01	0.997	0.001	0	0.002
0.02	0.992	0.0009	0.0001	0.007
...	...	...	...	...
0.98	.012	0.0002	0.0008	0.987
0.99	0.005	0.0001	0.0009	0.994
1.00	0	0	0.001	0.999

Figure 6.7: Example Threshold Action Distribution Map

ground truth positives are rare, on the order of 0.1%. As a result, $p(FN) + p(TP) \approx .001$ for every threshold value as approximately one of every thousand samples are ground truth positive threat samples. Such extreme imbalance causes significant difficulty when mathematically comparing the respective policy map vector to a given threshold vector. Therefore, we must renormalize according to the base rate b, as shown in Equation 6.12. The result of this method is that all four elements of the distribution will be within roughly the same order of magnitude, which will stabilize the comparison operation at each time step.

$$B(\theta) = \{p(FP), p(TP), p(FN), p(TN)\} \tag{6.11}$$

$$B(\theta) = \{p(FP), p(TP)/b, p(FN)/b, p(TN)\} \tag{6.12}$$

Online Static Distribution MDP Threshold Computation

Our objective, given the current state, is to **determine the threshold for which the distance between its action distribution and that of the optimal policy is minimum**. In other words,

Algorithm 4 Online Static Distribution Threshold Computation

Require: $\mathcal{Z}$: set of all inference scores and ground truths, C: available CMs, γ: lethality

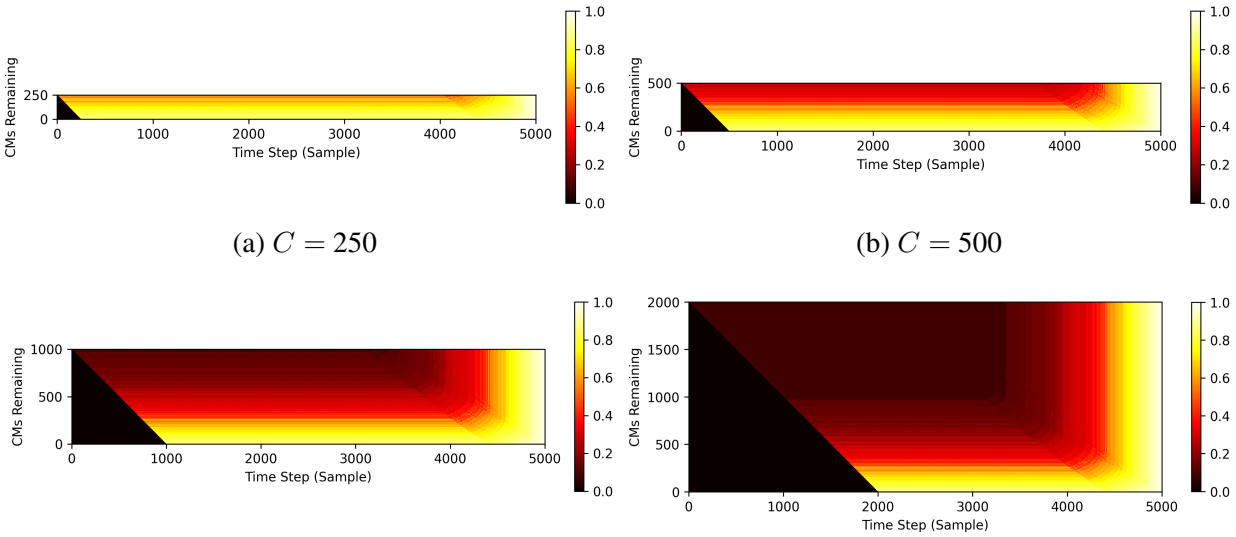
Initialization
$\pi_*(s) \leftarrow$ SPI(C, $|\mathcal{Z}|$, γ) // Initialize Policy Matrix from Alg. 3
$B(\theta) \leftarrow f(\mathcal{Z})$ // static threshold distribution derived from inference scores and GTs
$B(\theta)_{TP,FN} \leftarrow f(B(\theta))_{TP,FN}$ // renormalize according to Equation 6.12

Perform SCML Mission: Select Optimal Threshold and Inference Each Sample
for t $\leftarrow 1$ **to** $|\mathcal{Z}|$ **do**
    $\theta_s \leftarrow \underset{\theta}{\text{argmin}} \|B(\theta)_{TP,FN} - \pi_{*TP,FN}(s)\|$ // compute optimal threshold at given state
    $a_t \leftarrow f(\theta_s, z_t)$ // compute action from threshold and inference score
    $P(s_{t+1}) \leftarrow P(s'_{t+1}|s_t, a_t)$ // transition to next state
end for

the agent seeks to find the threshold with an action distribution which most closely matches the action distribution of the optimal policy of the current state. Algorithm 4 describes the process in detail. It first initializes the optimal policy by executing stochastic policy iteration from Algorithm 3. It also initializes the threshold action distribution map from the SCML inference logs and renormalizes it at each threshold to ensure reasonably balanced distributions. Once renormalized, we extract the $p(TP)$ and $p(FN)$ elements of each threshold distribution to compute the distance from the equivalent elements of the optimal policy action distribution. Once the optimal policy for each state and the threshold action distribution map have been computed, the algorithm then iterates through each encountered sample in the SCML mission. Prior to inferencing, it first computes the optimal threshold, whose action distribution $\{p(TP), p(FN)\}$ has the minimum distance to the equivalent $\{p(TP), p(FN)\}$ of the optimal policy action distribution.

We took the additional step of extracting $\{p(TP), p(FN)\}$ in computing the optimal threshold as including all four elements of the action distribution yielded uninformative results. We also attempted to use the three elements: $p(TP)$, $p(FN)$, and $p(FP)$ as FPs affect CM depletion rates, however, this approach also yielded uninformative results. Considering only the distribution shift between TPs and FNs proved to be the most numerically stable approach with the most informative results, we elected to use this method. The weakness of this pragmatic approach is that the expected future reward (or reduced future reward) of a potential FP is not integrated into the threshold calculation. For example, if the agent were in a state where only 20 time steps remained, but 30 CMs remained, then the value of an FP action would be large compared to others. To maximize the probability of an FP, the threshold would be set to zero. As long as the number of CMs remaining were greater than or equal to the number of time steps remaining, the optimal action would be an FP. Yet after normalizing the rate of TPs and FNs due to extreme class imbalance and including FP in the distance calculation, numerical instability caused uninformative results. This prevented the effective use of $p(FP)$ in the threshold selection process.

For this offline approach, a set of SCML mission inference scores is required to compute the threshold action distribution map $(B(\theta))$ that is ultimately used to compute the optimal threshold at each state. In our previous SCML experiments, we typically had a ground truth positive base rate of approximately 0.1%. However, when first generating experimental results, such extreme class imbalance yielded uninterpretable optimal policies. This translated into poor threshold computation at each time step. Separately, the size of the state space is a function of CMs and time steps. In our previous results, a single scout would inference roughly 36,000 samples and have, at a maximum, 14,000 CMs available. This rendered GPU memory and computation time requirements to be intractable for our purposes of demonstrating the feasibility of this algorithm. Therefore, we decided to reduce the number of negative samples inferenced by each scout. The seven scouts, on average, each inference approximately 47 ground truth positives. With 36,000 original negative samples, the base rate is on the order of 0.1%. To mitigate the two problems, we simply selected a random set of negatives across the entire sample set to ensure the number of samples in a single mission is 5000. As a result, we reduced the state space to a more reasonable size in which the stochastic policy iteration algorithm could be executed. Also, we reduced the ground truth positive base to around 1%, which yielded a much more interpretable and informative optimal policy. This therefore translated into a threshold map as a function of state space that was more intuitive with respect to the impact of probability state transitions and reward function.



(a) $C = 250$ (b) $C = 500$

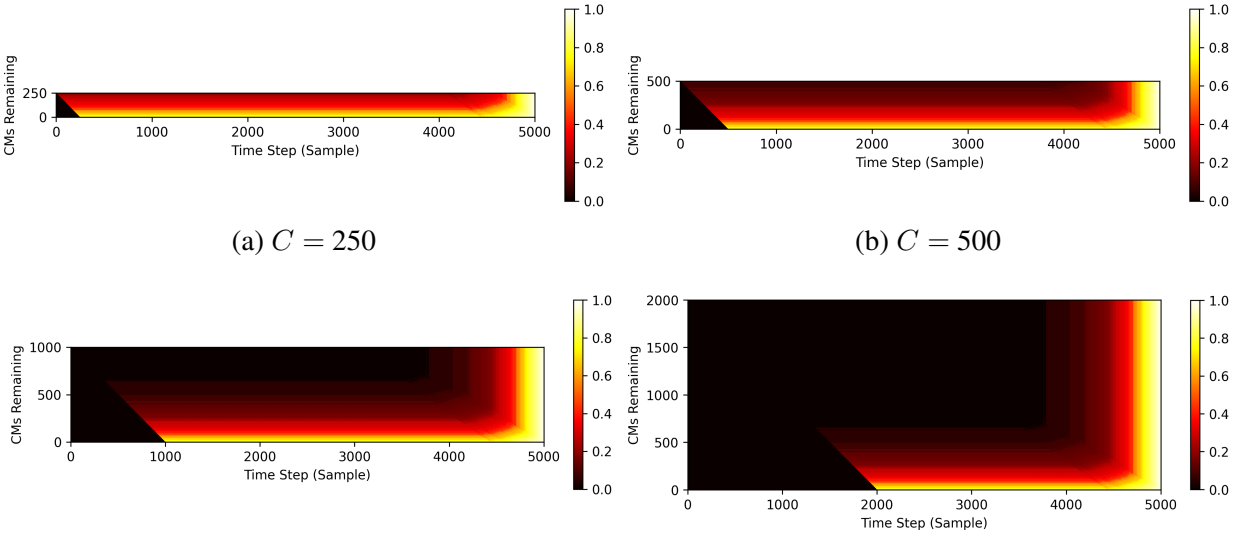
(c) $C = 1000$ (d) $C = 2000$

All graphs above assume a threat lethality of $\gamma = 0.01$. These results correspond to Model 0 (No Learning) missions of class Roundabout for the DOTA dataset.

Figure 6.8: Threshold by State: $\gamma = 1\%$, Model 0.

Experimental Results: Threshold Maps by State & Survivability

In Figure 6.6 we showed example heatmaps of each action distribution component of an optimal policy. As we have described in previous chapters, FNs are more likely to be potentially lethal earlier in the mission when sufficient countermeasures remain. However, as the agent approaches the end of the mission, the FN component of the policy $\pi_*(FN|s)$ slowly increases as the FN component of the overall expected reward increases. If the agent depletes its CMs too quickly, it will be at the mercy of both FNs and TPs, which would effectively double its probability of being destroyed. Threshold therefore increases when the agent moves into states with few remaining CMs. This trend is depicted in Figure 6.8, for a lethality of 1%. We observe in all four plots that threshold generally increases as CMs are depleted and the agent moves closer to mission completion. This is due to the share of cumulative reward each action (TP or FN) produces for the current state. With 250 CMs, Figure 6.8(a) depicts a relatively high starting threshold, increasing slightly over the course of the mission. With 500 CMs in Figure 6.8(b), the threshold starts at a reduced level than with 250. More available CMs allows the agent to keep the threshold lower for longer into the mission timeline in order to avoid as many FNs as possible. Figure 6.8(c) shows an even further reduced starting threshold with double the number of available CMs than the previous figure. This trend continues with 2000 CMs in Figure 6.8(d) where the starting threshold is the lowest, as expected. We also see in this final plot that after the agent has expended a certain number of CMs over a given number of time steps, the current threshold will clearly be different from a plot with fewer CMs available. For example, starting with 250 CMs in Figure 6.8(a) and ending with 2000 CMs in Figure 6.8(d), consider an agent that has progressed 2000 time steps into the mission and has expended 250 CMs. With only 250 available CMs from the start, the threshold will be 0.7. Starting with 500 CMs, it will be roughly



(a) $C = 250$ (b) $C = 500$

(c) $C = 1000$ (d) $C = 2000$

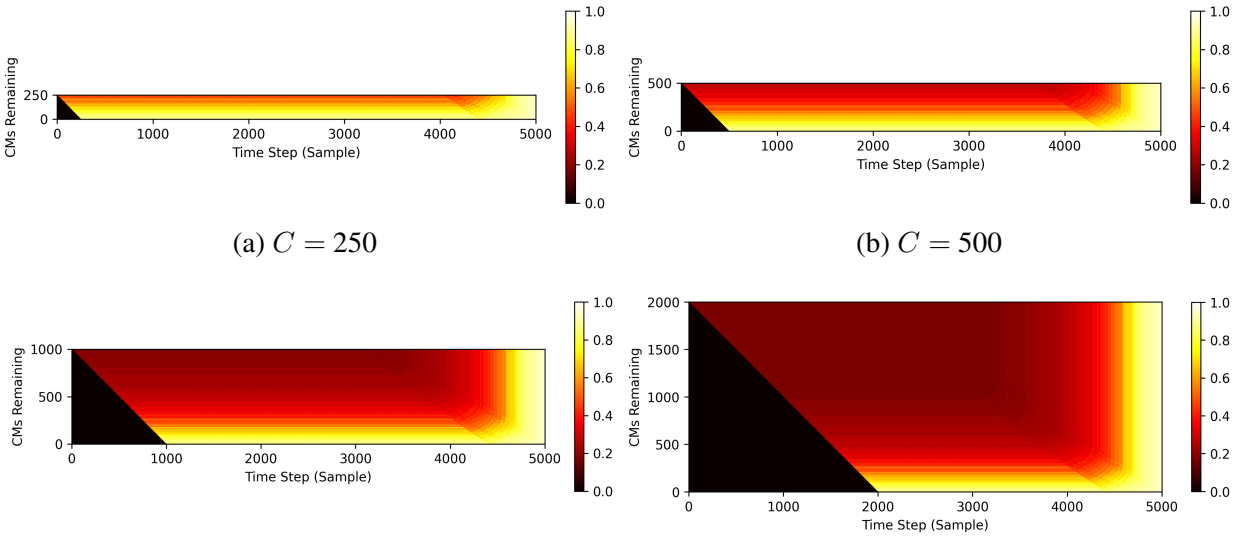
All graphs above assume a threat lethality of $\gamma = 0.02$. These results correspond to Model 0 (No Learning) missions of class Roundabout for the DOTA dataset.

Figure 6.9: Threshold by State: $\gamma = 2\%$, Model 0.

0.5. Starting with 1000 CMs, it will be 0.2. Finally, starting with 2000 available CMs, the threshold will be about 0.1. This shows how the original number of available CMs can influence the optimal threshold, even if the same number of CMs have been expended at the same time step in the mission.

We observe that all heatmaps in Figure 6.8 exhibit subtle behaviors that are a result of the method we use to compute the optimal threshold as a function of state. First, threshold monotonically increases as the mission progresses and CMs deplete. However, in the extreme upper right of the state space where the number of CMs remaining is greater than the remaining time steps, the threshold should be set to zero. At this position in the state space, there are no negative consequences to this strategy given that CMs will not be depleted and all subsequent threats will be neutralized. Further, as the agent simply approaches this region of the state space, the optimal threshold should drop. For example, if there are 30 time steps remaining and the agent possesses 25 CMs, setting the threshold to a low, but non-zero, value would be optimal. Therefore, generally speaking, the threshold should decrease if the agent moves to the right (close to the end of the mission) but maintains a reasonably proportional number of CMs. If the agent reaches the upper right region in the state space where the number of CMs remaining is greater than or equal to the number of remaining time steps, the threshold should be set to zero. Clearly, this behavior is not reflected in the heatmaps in Figures 6.8, 6.9, and 6.10, and stems from the reasons described below.

As discussed previously, we based the optimal threshold computation on $\{p(TP), p(FN)\}$ as opposed to all four action elements. This is because using all four elements or only adding $p(FP)$ resulted in uninformative and suboptimal results. As a result, both the extreme class imbalance, which we attempted to mitigate with renormalization, and softmax sensitivity to im-



(a) $C = 250$ (b) $C = 500$

(c) $C = 1000$ (d) $C = 2000$

All graphs above assume a threat lethality of $\gamma = 0.01$. These results correspond to Model 8 (No Learning) missions of class Roundabout for the DOTA dataset.

Figure 6.10: Threshold by State: $\gamma = 1\%$, Model 8.

balanced input distributions make the integration of $p(FP)$ generate poor performance. In future work, perhaps in scenarios where raw action distributions are reasonably balanced, integrating all elements of the action distribution would be tenable. However, we assume extreme class imbalance (relatively rare observations of threat objects) in SCML scenarios, which results in the challenges we have described. Under less extreme conditions of class imbalance, we could integrate all four action elements in our distance comparison to compute the optimal threshold. As a result, the upper right regions of our threshold heatmaps would tend toward darker colors (lower threshold) when the number of remaining CMs approaches the number of time steps. Similarly, the threshold would turn black (zero) when the number of remaining CMs surpassed the number of remaining time steps. This is the reason for the lack of the unexpected behavior in the respective heatmaps representing optimal computed threshold as a function of state space.

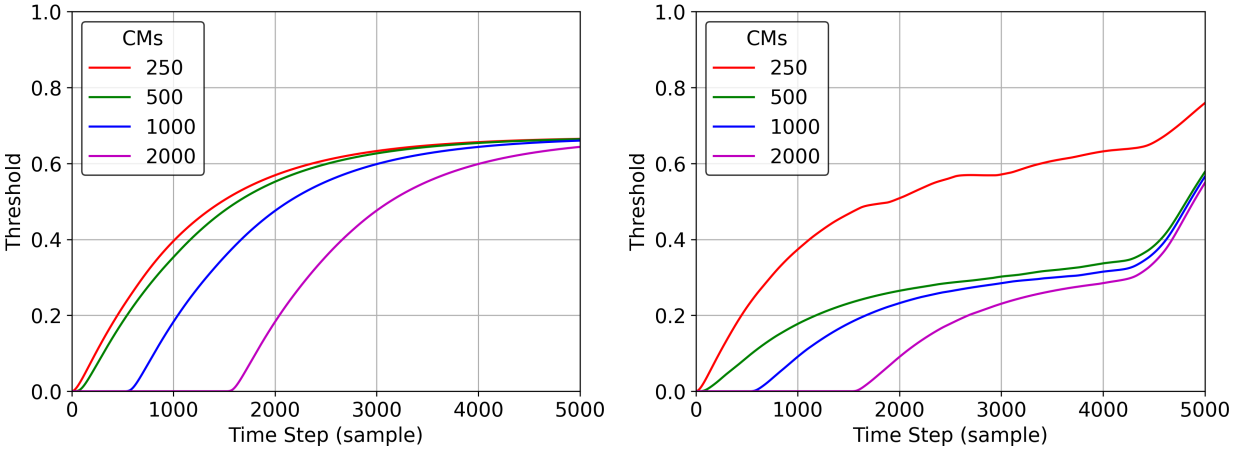
In our initial results in §4.6, we saw that lethality had a predictable effect on survivability. If lethality doubled, the probability of mission success was reduced by 50%. However, as we will see later, this is not the case when modeling SCML as an MDP. When performing stochastic policy iteration, lethality is a required parameter to the probability state transition matrix, and therefore impacts the optimal policy and thus the threshold map. Figure 6.9 demonstrates this effect when lethality is 2%. It is clear that the agent seeks to hold the threshold at a lower level for the vast majority of the state space. For Figure 6.9(d), much of the state space yields a threshold of zero as long as plentiful CMs remain. Once the agent only has approximately 500 CMs remaining (regardless of how many it started with), the threshold starts to increase in order to conserve them. For this figure and all other figures with fewer CMs: 6.9(a), 6.9(b), and 6.9(c) the agent, especially early in the mission and with few expended CMs, specifically avoids to a greater degree any chance of a potentially lethal FN. The agent learns that a greater

lethality across the entire mission renders the occurrence of an FN at any time step a much greater aggregate threat. The weighted reward of an FN at each time step is less than when lethality was 1%, therefore the agent learns that relative to other actions, FNs are even more potentially lethal, resulting in much less total reward.

Figures 6.8 and 6.9 computed their threshold action distributions based on inference scores from model 0, the initial bootstrap model. Figure 6.10 depicts threshold maps derived from inferences scores of model 8 (the final model trained in a Live Learning mission), with a lethality of 1%. We saw in §4.9 the effects that improved models had on score distribution. For the Roundabout class, while 79% of all ground truth positive samples had increased scores with Live Learning, still 21% of the scores decreased. This effect is visualized in Figure 4.34(d). The decreased scores of a fifth of the ground truth positives are rather evenly spread across the score spectrum. This results in the smooth gradient seen most clearly in Figure 6.10(d), where the threshold smoothly rises from around 0.3 up to about 0.8. The other primary value that Live Learning provides in improving models over time is the reduction of ground truth negative scores. The initial threshold in Figures 6.10(a), 6.10(b), and 6.10(c) are equal to or slightly above that for the equivalent plots for model 0 in Figure 6.8. This is because model 8 can achieve the same ratio of $\{p(TP), p(FN)\}$ dictated by the optimal policy at a higher threshold. Model 8 can therefore achieve the same probability of TPs and FNs while minimizing the rate of CM deployment. If the threshold had to be much lower to achieve the optimal policy action distribution, then many more CMs would be wasted, as in Figure 6.8.

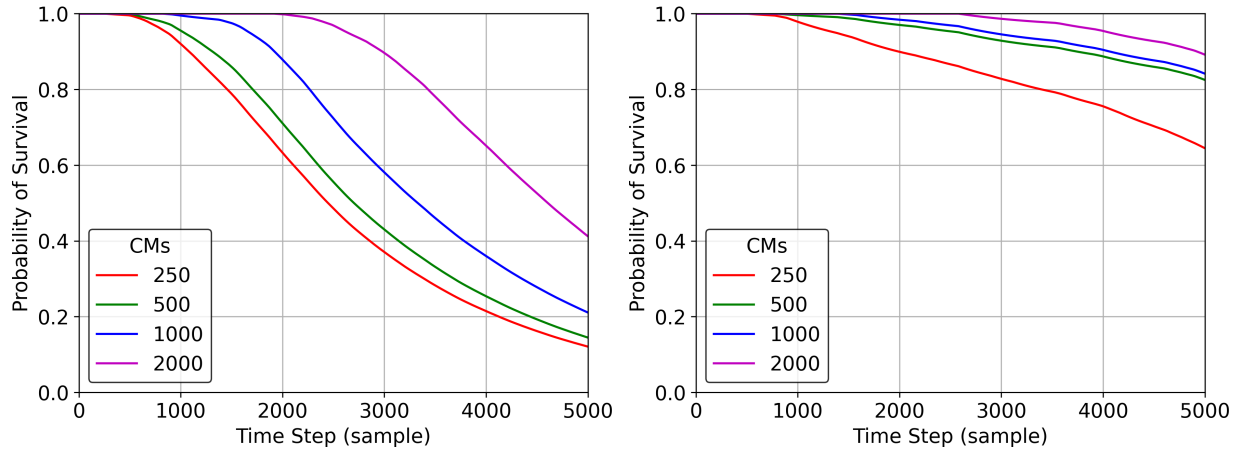
The previous threshold maps depict the optimal thresholds derived from the comparison between the computed optimal policy action distribution and the threshold action distribution map. Now we show results when sending the agent through the state space according to the set of encountered samples (i.e. run the identical SCML mission), according to Algorithm 4. Figure 6.11 depicts the optimal threshold and survivability curves for a lethality of 2% for models 0 and 9 of a static (No Learning) SCML mission for class Swimming Pool. In Figure 6.11(a) we observe that curves for numbers of CMs have the same shape yet curves of fewer CMs rise more rapidly at the start of the mission to avoid depleting CMs prematurely. With 2000 CMs, the threshold is held the longest at zero until it depletes a certain number of CMs, at which point the threshold is forced to rise. All four curves converge to approximately 0.65 in a stable manner by the end of the mission, meaning CMs are depleting at a steady rate and no sudden jumps in threshold are necessary. The corresponding survivability curves are shown in Figure 6.11(c), where intuitively the curve with 2000 CMs has the highest likelihood of mission success. The primary reason for this is that with more CMs, this curve is able to hold survivability at 100% for the longest amount of time (due to a threshold of zero) and thus decays for less time than the other curves (from less cumulative exposure to potentially lethal threats).

Figure 6.11(b) depicts the threshold when model 9 is employed for the entire mission. Although model 9 is generated from a Live Learning SCML mission, we use it here to compare performance to model 0 if both were used exclusively in a static No Learning mission. Threshold behavior is similar to model 0 in that greater numbers of CMs allows it to be held at zero for a longer duration at the beginning of the mission. The primary differences here are the shape of the curves compared to model 0. All but the smallest number of CMs (250) mostly converge late in the mission and then rise together at the very end. We see this late spike in threshold when CMs deplete too quickly and the agent seeks to balance the distribution of TPs and FNs with roughly



(a) Optimal Threshold, Model 0

(b) Optimal Threshold, Model 9



(c) Survivability, Model 0

(d) Survivability, Model 9

These results correspond to No Learning (Static Distribution) missions of class Swimming Pool for the DOTA dataset, $\gamma = 2\%$.

Figure 6.11: Optimal Threshold and Survivability by Time Step (No Learning)

500 samples remaining. Compared to model 0, CMs depleted more rapidly due to significantly lower threshold for the first 90% of the mission. Although model 9 has superior precision to model 0 (reduced frequency of FPs), the threshold can still be reduced a bit too low to be sustainable for the remainder of the mission, thus requiring the spike at the end (to prevent premature CM depletion). Model 9 clearly achieves improved survivability over model 0 in Figure 6.11(d). All levels of available CMs provide significant gains in the probability of mission success. The primary difference compared to model 0 is that survivability for 250 available CMs is the outlier whereas with model 0 the curve with 2000 CMs is the outlier. This effect can be visually intuited from analogous patterns in the threshold curves.

We have shown differences in threshold and survivability performance in the replay of a real SCML mission by comparing the resulting behavior of two different models. Model 9 is clearly superior to model 0 in terms of survivability at all levels of available CMs. We thus

demonstrated that within the framework of an MDP, model performance within a static (No Learning) SCML system has an overwhelming impact on optimal threshold computation and the resulting probability of mission success.

6.3.4 Variable Threshold Action Distribution

This approach is an extension to Static Threshold Action Distribution, whereby the threshold action distribution map changes as a new model is installed during a Live Learning mission. For all results in the previous section, a single threshold action distribution map was computed offline prior from the previous SCML mission and used to compute the optimal threshold for each state. In this section, as each model is periodically retrained, we assume that the new model has a unique threshold action distribution map, and therefore install each new map version associated with each new model.

Online Variable Distribution MDP Threshold Computation

Algorithm 5 defines this extension, which differs slightly from the static version in Algorithm 4. The primary difference is that a separate threshold action distribution map is computed for each model offline according to the previous SCML mission inference scores. This is noted on line 3. During Live Learning mission execution, the respective threshold action distribution map is used according to which model is currently inferencing. This is noted on line 7 of the algorithm, while line 8 depicts the output action being partially a result of the inference score from model M.

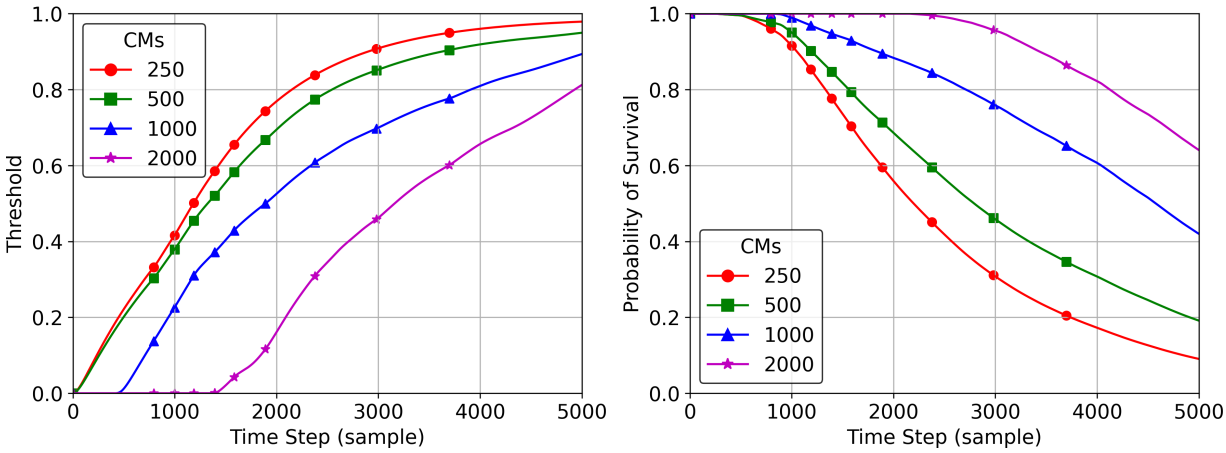
As we have seen in previous chapters, Live Learning does not always produce a monotonically increasing AUC on a held out test set. An even more difficult metric to achieve is one that guarantees all positive sample scores to increase while all negative score decrease. This is especially difficult with extreme class imbalance. Given these facts and our analysis in §4.9, we expect that the threshold changes will be somewhat unstable as the model turnover occurs. When

Algorithm 5 Online Variable Distribution Threshold Computation

Require: $\mathcal{Z}$: set of all inference scores and ground truths, C: available CMs, γ: lethality, $\mathcal{M}$: set of all models used for inference from a Live Learning mission

1: Initialization
2: $\pi_*(s) \leftarrow$ SPI(C, $|\mathcal{Z}|$, γ) // Initialize Policy Matrix from Alg. 3
3: $B_M(\theta) \leftarrow f(\mathcal{Z})$ // var. thresh. dist. derived from infer. scores and GTs for each model M
4: $B_M(\theta)_{TP,FN} \leftarrow f(B_M(\theta))_{TP,FN}$ // renormalize according to Equation 6.12

5: Perform SCML Mission: Select Optimal Threshold and Inference Each Sample
6: **for** t $\leftarrow 1$ **to** $|\mathcal{Z}|$ **do**
7: $\quad \theta_s \leftarrow \underset{\theta}{\text{argmin}} \| B_M(\theta)_{TP,FN} - \pi_{*TP,FN}(s) \|$ // compute optimal threshold at given state
8: $\quad a_{t,M} \leftarrow f(\theta_s, z_{t,M})$ // compute action from threshold and inference score from model M
9: $\quad P(s_{t+1}) \leftarrow P(s'_{t+1}|s_t, a_t)$ // transition to next state
10: **end for**



(a) Optimal Threshold

(b) Survivability

These results correspond to a Live Learning (Variable Distribution) mission of class Swimming Pool for the DOTA dataset, $\gamma = 2\%$.

Figure 6.12: Optimal Threshold and Survivability by Time Step (Live Learning)

transitioning from model to model throughout the mission, each transition may not occur in the exact same manner. Specifically, the action distributions at each threshold may not move in the same direction from model 0 to model 1 as from from model 1 to model 2. Therefore, optimal threshold by state may not appear as smooth as in the previous, static approach.

Experimental Results: Online Variable Distribution MDP Threshold Computation

Figure 6.12 shows both threshold and survivability results with a lethality of 2% for a Live Learning mission using variable threshold action distribution maps. Figure 6.12(a) depicts the steep increase in threshold at the start of the mission as we saw with previous curves for model 0 in Figure 6.11(a). At some model transition points, there are slight variances in the threshold slope. This represents the minor disruptions in threshold adjustment given that each model has a unique associated threshold action distribution map. Figure 6.12(b) shows the corresponding survivability curves for an SCML mission with Live Learning. With 2000 CMs after six new models, the probability of mission success holds at 100% but the threshold has already risen close to 0.2, demonstrating that Live Learning does not sacrifice recall at the cost of improving precision, but rather improves it as well.

The threshold curves in Figure 6.12(a) begin to slowly converge at the end of the mission, but to a higher threshold than in Figure 6.11(a). All four curves also demonstrated that thresholds could not be kept to a minimum until the end of the mission as in Figure 6.11(b) for model 9. Therefore, we observe that the threshold curves for a variable action distribution (Live Learning) mission combined the behavior of a weak model (model 0), a strong model (model 9), and all models generated in between, as expected. Overall performance in survivability improved over model 0 in all cases except for 250 CMs. This is due to extreme resource constraint, giving the agent minimal flexibility in navigating the state space, and imperfect score diffusion in that Live Learning cannot guarantee every positive score will increase and every negative sample score will decrease.

6.4 Online Model-Free Q-Learning

This approach assumes the agents (scouts) have no foreknowledge of the score distribution of the samples they will encounter. They also have no knowledge about the environment, such as the state transition probabilities, associated rewards, or lethality. This makes for a challenging problem in which the agents must simply react to each inference result and attempt to learn to compute the optimal policy over many episodes. Over time an optimal policy can be computed but one would have to decide whether the agent should have access to the accurate threshold action distribution from a previous mission prior to executing the MDP or if it must learn that distribution in real-time, which will of course result in poor and unstable performance. We leave this to future work.

6.5 Summary

In this chapter we have modeled SCML within a unique MDP framework. A team of scouts each execute their individual SCML missions and compute their own optimal threshold based on their current state, which is determined by the number of remaining CMs and the time until mission completion (in units of sample inferences). We specified the combination of characteristics that make the SCML MDP framework unique: Finite Horizon, Undiscounted, Partial Observability, Stochasticity, and Stationarity. We then defined a probability state transition matrix, an associated reward function, and the stochastic policy iteration algorithm required to compute the optimal policy for the given environment. Further, we showed how the optimal threshold is selected for both the static and variable online action distribution threshold computation algorithms. Results documented the optimal threshold selection at each time step during an SCML mission and the resulting survivability for both static (No Learning) and variable (Live Learning) threshold distribution maps.

Although the MDP defined in this chapter may not integrate every possible environmental characteristic (e.g. state, reward) or every potential SCML system detail that would exist in real-world scenarios, it is an initial model from which to build more specific SCML MDP frameworks. As previously stated, most MDPs in the context of RL research focus on geographic state space representation, at least in part. The performance of SCML and Live Learning, the interaction of which is part of the focus of this dissertation, is evaluated as a function of time, and therefore it was appropriate to model the state space here accordingly. In future iterations of SCML MDP frameworks, both time and space should represent the state space such that relative positions of friendly and adversarial agents as well as the model quality of the friendly agent impact its ability to accurately detect threats in the environment in the pursuit of maximizing survivability.

Finally, we discussed in detail in section 6.3.3 the challenges in the distance computation between the policy at the given state and the action distribution map when computing the optimal threshold. We documented how the lack of integrating $p(FP)$ in the distance comparison prevented the threshold map from intelligently reducing the optimal threshold if the agent approached the end of the mission with excess CMs. The primary cause of this is extreme class imbalance in SCML scenarios and imbalance in iterative softmax calculations.

Chapter 7

Future Work & Conclusion

7.1 Future Work

The identification of SCML as a new ML paradigm and the analytical and experimental exploration of its complexities in this dissertation show that it is a canonical use case for Live Learning. The intuitive requirement that semi or fully autonomous agents must adapt to survive in hazardous environments combined with the inherent capabilities of Live Learning generates adjacent and derivative research topics. Traditional ML practices involve collecting large amount of data over relatively long time horizons, spending months training on such vast quantities of data and evaluating on established benchmarks, and deploying for production (whether public or private) only after many iterations of fine tuning, additional evaluation, and testing for optimal performance. SCML requires a completely different approach in that systems must be able to adapt to unpredictable changes in adversarial environments where the penalty for poor ML performance can be system destruction and mission failure. We believe there are many paths for future research related to the survivability of learning systems in adversarial environments, as detailed below.

7.1.1 Extending SCML to Physical Space

Survivability and related topics have been studied in the context of autonomous systems operating in physical environments [70–72, 75, 76]. A common goal of agents is to find an optimal path that minimizes line of sight exposure to potential adversarial agents. There could be many scenario configurations where adversarial agents are either stationary or mobile. In response, the friendly agent either must move to a new position every time step or can stay in the same position if the threat is too great for a certain amount of time. Understanding SCML in time and space simultaneously can be explored by using our initial SCML model where the goal of the agent is largely defined by time to avoid being destroyed for the duration of the mission. It can then be extended such that the agent must also reach a certain physical location within the mission duration, or deadline. Live Learning would still provide model performance improvements over time but with the added challenge of encountering threats also based on location (line of sight). The performance metric might be a simple win-loss outcome or something more complex

if the scenario is a team of SCML friendly agents against a team of adversarial SCML agents. Effectively, this approach extends SCML to a hybrid spatial-temporal space.

7.1.2 Collaborative SCML System Design and Operation

Currently, scouts collaborate in Live Learning merely by sharing any labeled rare positive threat samples with each other to facilitate rapid model improvement. However, with respect to SCML operation, scouts (functioning as SCML systems) do not modify their behavior according the state of any other scout. For instance, if one scout encounters a dense threat environment, other scouts do not compensate to ensure they have a greater chance of survival. With Live Learning, the only way for a scout to know whether it is experiencing a dense threat environment is if it receives a large number of positive labels back from home. For purposes of SCML collaboration, we may assume that a system knows immediately whether a sample is an actual threat given that it will detect it as a TP if classified correctly. In this scenario, where a scout is experiencing a high-threat frequency, other scouts boost their rate of CM deployment, or their maximum rate for example, in order to ensure the probability of the survival of M scouts does not degrade too drastically. These scouts then would recover as designed according to adaptive thresholding algorithm described in §4.10. This is just one example of scout collaboration in SCML to compensate for differential threat density across individual scouts.

7.1.3 Complex Adversarial Agent Characterization

In this work, adversarial agents are defined purely by a single data sample within a large, streaming dataset. The threat they pose to SCML systems is determined by the currently inferencing model. We currently only model the ML capabilities of the SCML system, not the adversarial agent. We could model adversarial agent behavior by providing it with ML capabilities such that it influences its lethality and its ability to avoid detection by the SCML system. We would then be able to model the survivability of the adversarial agent itself compared to the friendly SCML system. Analyzing the impacts on survivability from both sides based on the ability to avoid detection by their counterpart. Included in this analysis would also be Live Learning parameters such as: the labeling rate of each home station, the strategic deployment of scouts (across time), and others.

7.1.4 Online Model-Free Q-Learning

In chapter 6 we defined an SCML mission as a sequential Markov Decision Process (MDP) whereby an agent selects the optimal CM deployment threshold according to the optimal action distribution of the current state. This action distribution is computed offline prior to the agent executing the mission. We restricted the work in that chapter to a model-based approach where both the score distribution and environmental characteristics like probability state transitions, reward function, and lethality are known. In a model-free approach, the agent must learn these state transition probabilities and associated rewards by running many iterative episodes and determining the optimal action distribution, and by extension the optimal threshold, at each state. Although an agent cannot run a realistic mission many times before running it for performance

evaluation, it is conceivable that many agents execute such missions in parallel over time. As a result, each new agent (scout) could leverage all knowledge learned about the environment from all previous missions performed by other agents to maximize survivability in adversarial environments. This could be applicable to temporally distributed drone swarms where new drones or other autonomous systems can learn from consequences from actions of previously operating drones.

7.1.5 Complex Threat Arrival Distribution

In all experiments in this dissertation, the threat interarrival times are exponentially distributed, with a constant average arrival rate. However, under the assumption that threats in SCML are intelligent actors, it is more realistic for their average arrival rates to vary during the mission. As we have stated previously, there is a tradeoff between exposure to threats and the ability to learn quickly enough to neutralize future threats (through improved models).

A densely clustered set of threats could prove much more fatal than that which is more evenly distributed across time, especially early in the mission prior to any learning iterations, exhibiting temporal variability. This is primarily because CM depletion will accelerate in a short period of time, which may cause throttling or even exhaustion. We could also model scouts with stealth capabilities, where adversarial agents have a lower lethality earlier in the mission but a greater lethality later in the mission as they learned to better detect the friendly SCML system.

Spatial variability may manifest in that adversarial agents might cluster around and defend more valuable targets in the environment. The practical effect on an SCML system is that a small minority of scouts may encounter the vast majority of threat samples but many scouts may encounter almost none. This would generally have a negative effect on survivability as threat samples are concentrated in the priority queues of fewer scouts, making it more likely that they do not get transmitted in a timely manner, if at all, to home for labeling. Deeper exploration is warranted for these unique scenarios as they are more likely in reality.

7.2 Conclusion

This dissertation introduced a new ML paradigm called Survival-Critical Machine Learning (SCML) where ML systems leverage their knowledge to survive in complex adversarial environments in which threats evolve, morph, and appear unexpectedly. Live Learning is the framework used to sense, act, and adapt to such environments by iteratively improving its internal ML models. SCML is designed to lay the groundwork for autonomous systems that must make real-time decisions to ensure their survival in adversarial, dangerous, and hostile environments.

As stated in Chapter 1, the thesis validated by this dissertation claims:

> For improved survival of edge-based systems in environments where threats evolve and morph, it is feasible and effective to implement a new model of continuous learning called Survival-Critical Machine Learning (SCML). SCML can be implemented with Live Learning. SCML encompasses a rich tradeoff space. It can be applied to well-known ML paradigms such as Markov Decision Processes (MDPs). SCML can work with a wide range of sensor modalities.

To validate this thesis, we defined an SCML analytical and experimental model that support a diverse set of SCML scenarios and enable us to evaluate their primary metric: the probability of mission success, also known as the probability of survival. The analytical model defined in Chapter 3 provided initial mathematical intuition for how to compute survivability as a function of time for a Live Learning SCML system. It involved the combination of basic system and ML parameters that dictate the rate of decay of survivability. We showed how the decay rate decreases with each new model (as recall improves) and specifically how the impact of the timing, magnitude, and the total number of recall improvements via new models affects long-term survivability.

In Chapter 4 we defined the experimental model, which is used to process all samples in a single SCML experiment for the respective scout. We documented the effect on survivability of Live Learning (both standard and biased), the number of available CMs, the CM deployment threshold, and lethality. Relaxing the survival criterion to M of N scouts for mission success also provided insight as to the impacts of these parameters when not all N scouts must survive the mission to be considered successful. We demonstrated how this relaxed criterion impacted survivability in staggered scout deployment scenarios. With novel class discovery, we described two methods of of dynamically expanding the set of threat classes during the mission to maximize the adaptability of SCML systems leveraging Live Learning. We also described the nuanced differences in metrics between Live Learning and SCML: relative vs. absolute inference scores. As a result, we discovered that Live Learning generally improves survivability but not necessarily in all possible scenarios or parameter combinations.

We demonstrated the extensibility of Live Learning to sensor modalities other than visual data in Chapter 5, specifically radar. We leveraged an existing dataset that required some transformation to adapt it to an optimal format for use in Hawk. Our results showed that Live Learning can increase the number of collected TP radar samples under the same challenging conditions for which Live Learning was designed. It is clear that radar data is less feature rich than visual data, which allowed for a smaller required number of samples for training near-perfect Brute Force models.

In Chapter 6 we modeled an SCML mission as a Markov Decision Process (MDP), a sequential decision making process. As an MDP, the task of an SCML system is to select the optimal threshold for each sample inference that generates an action probability distribution which most closely matches the optimal action probability distribution from the offline MDP solution. We demonstrated how to perform a No Learning SCML mission using an MDP as well as a Live Learning mission using an MDP where the threshold maps changed over time with training and installation of each successive model. We were ultimately able to show how MDP and Reinforcement Learning concepts can be applied to SCML in the pursuit of maximizing survivability. Additional exploration of modeling SCML as an MDP would be valuable in enhancing the performance of Live Learning autonomous systems.

In closing, SCML is a new ML paradigm that has high value in adversarial settings where threats evolve and morph. This dissertation sought to provide an initial exploration of how to model the survivability of autonomous SCML systems enabled by human-in-the-loop Live Learning. Additional study will be needed to optimize both individual system and collective team performance for maximum operational impact.

Bibliography

[1] José Ricardo Sánchez-Ibáñez, Carlos J Pérez-del Pulgar, and Alfonso García-Cerezo. Path planning for autonomous mobile robots: A review. *Sensors*, 21(23):7898, 2021. 1.1

[2] Mohamed Reda, Ahmed Onsy, Amira Y Haikal, and Ali Ghanbari. Path planning algorithms in the autonomous driving system: A comprehensive review. *Robotics and Autonomous Systems*, 174:104630, 2024.

[3] Lixing Liu, Xu Wang, Xin Yang, Hongjie Liu, Jianping Li, and Pengfei Wang. Path planning techniques for mobile robots: Review and prospect. *Expert Systems with Applications*, 227:120254, 2023.

[4] Siyu Teng, Xuemin Hu, Peng Deng, Bai Li, Yuchen Li, Yunfeng Ai, Dongsheng Yang, Lingxi Li, Zhe Xuanyuan, Fenghua Zhu, et al. Motion planning for autonomous driving: The state of the art and future perspectives. *IEEE Transactions on Intelligent Vehicles*, 8 (6):3692–3711, 2023.

[5] Lu Dong, Zichen He, Chunwei Song, and Changyin Sun. A review of mobile robot motion planning methods: from classical motion planning workflows to reinforcement learning-based architectures. *Journal of Systems Engineering and Electronics*, 34(2):439–459, 2023.

[6] S Nahavandi, R Alizadehsani, D Nahavandi, S Mohamed, N Mohajer, M Rokonuzzaman, and I Hossain. A comprehensive review on autonomous navigation. arxiv 2022. *arXiv preprint arXiv:2212.12808*.

[7] K Cai, C Wang, J Cheng, CW De Silva, and MQH Meng. Mobile robot path planning in dynamic environments: A survey. arxiv 2020. *arXiv preprint arXiv:2006.14195*, 2020.

[8] MWM Gamini Dissanayake, Paul Newman, Steve Clark, Hugh F Durrant-Whyte, and Michael Csorba. A solution to the simultaneous localization and map building (slam) problem. *IEEE Transactions on robotics and automation*, 17(3):229–241, 2001.

[9] Michael Montemerlo, Sebastian Thrun, Daphne Koller, Ben Wegbreit, et al. Fastslam: A factored solution to the simultaneous localization and mapping problem. *Aaai/iaai*, 593598:593–598, 2002.

[10] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: A versatile and accurate monocular slam system. *IEEE transactions on robotics*, 31(5):1147–1163, 2015.

[11] Johann Borenstein, Yoram Koren, et al. The vector field histogram-fast obstacle avoidance

for mobile robots. *IEEE transactions on robotics and automation*, 7(3):278–288, 1991.

[12] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

[13] Sebastian Thrun. Probabilistic robotics. *Communications of the ACM*, 45(3):52–57, 2002.

[14] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[15] Terrence Fong, Illah Nourbakhsh, and Kerstin Dautenhahn. A survey of socially interactive robots. *Robotics and autonomous systems*, 42(3-4):143–166, 2003.

[16] Scott Drew Pendleton, Hans Andersen, Xinxin Du, Xiaotong Shen, Malika Meghjani, You Hong Eng, Daniela Rus, and Marcelo H Ang. Perception, planning, control, and coordination for autonomous vehicles. *Machines*, 5(1):6, 2017. 1.1

[17] OpenStax. Survivorship Curve. `https://bio.libretexts.org/Bookshelves/Introductory_and_General_Biology/General_Biology_1e_(OpenStax)/8%3A_Ecology/45%3A_Population_and_Community_Ecology`, 2020. Creative Commons Attribution 4.0 International License (CC BY 4.0). 1.2

[18] Shilpa George, Haithem Turki, Ziqiang Feng, Deva Ramanan, Padmanabhan Pillai, and Mahadev Satyanarayanan. Low-bandwidth self-improving transmission of rare training data. In *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*, pages 1–15, 2023. 1.6, 1.3, 2.2, 4, 5.1, 5.3

[19] Gui-Song Xia, Xiang Bai, Jian Ding, Zhen Zhu, Serge Belongie, Jiebo Luo, Mihai Datcu, Marcello Pelillo, and Liangpei Zhang. DOTA: A Large-Scale Dataset for Object Detection in Aerial Images. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018. 1.8, 4.2

[20] Shilpa George, Jan Harkes, Tom Eiszler, and Eric Sturzinger. Hawk Source Code, 2023. URL `https://github.com/cmusatyalab/hawk`. Last accessed July 22, 2023. 1.8, 2.2

[21] Mahadev Satyanarayanan, Wei Gao, and Brandon Lucia. The Computing Landscape of the 21st Century. In *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications (HotMobile '19)*, Santa Cruz, CA, 2019. 2.1

[22] Grace Lewis, Sebastián Echeverría, Soumya Simanta, Ben Bradshaw, and James Root. Tactical cloudlets: Moving cloud computing to the edge. In *2014 IEEE Military Communications Conference*, pages 1440–1446. IEEE, 2014. 2.1

[23] Mahadev Satyanarayanan, Grace Lewis, Edwin Morris, Soumya Simanta, Jeff Boleng, and Kiryong Ha. The role of cloudlets in hostile environments. *IEEE Pervasive Computing*, 12(4):40–49, 2013. 2.1

[24] Mahadev Satyanarayanan. The emergence of edge computing. *Computer*, 50(1):30–39, 2017. 2.1

[25] Likai Yao, Qinxuan Shi, Zhanglong Yang, Sicong Shao, and Salim Hariri. Development of an edge resilient ml ensemble to tolerate ICS adversarial attacks. *arXiv preprint*

arXiv:2409.18244, 2024. 2.1

[26] Vladimir Shakhov and Anastasiya Yurgenson. Reliability modeling for industrial edge computing systems. In *2021 17th International Asian School-Seminar" Optimization Problems of Complex Systems (OPCS)*, pages 108–112. IEEE, 2021. 2.1

[27] Paula Fraga-Lamas and Tiago M Fernández-Caramés. Tactical edge iot in defense and national security. *IoT for Defense and National Security*, pages 377–396, 2022. 2.1

[28] Eric M Sturzinger, Shilpa A George, and Mahadev Satyanarayanan. Tactical agility for ai-enabled multi-domain operations. In *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications V*, volume 12538, pages 209–220. SPIE, 2023. 2.1

[29] Major Greenwood. The first life table. *Notes and Records of the Royal Society of London*, 1(2):70–72, 1938. 2.3

[30] SJ Richards. A handbook of parametric survival models for actuarial use. *Scandinavian Actuarial Journal*, 2012(4):233–257, 2012. 2.3

[31] Ahmed Barakat, Aaina Mittal, David Ricketts, and Benedict A Rogers. Understanding survival analysis: Actuarial life tables and the kaplan–meier plot. *British Journal of Hospital Medicine*, 80(11):642–646, 2019.

[32] Edward W Frees, Jacques Carriere, and Emiliano Valdez. Annuity valuation with dependent mortality. *Journal of risk and insurance*, pages 229–261, 1996.

[33] Claudia Czado and Florian Rudolph. Application of survival analysis methods to long-term care insurance. *Insurance: Mathematics and Economics*, 31(3):395–413, 2002. 2.3

[34] R Swaminathan and H Brenner. Stastistical methods for cancer survival analysis. *IARC Sci Publ*, 162:7–13, 2011. 2.3

[35] David Collett. *Modelling survival data in medical research*. CRC press, 2023.

[36] EJJH Domingo and JJ Holland. RNA virus mutations and fitness for survival. *Annual review of microbiology*, 51(1):151–178, 1997.

[37] Lucila Ohno-Machado. Modeling medical prognosis: survival analysis techniques. *Journal of biomedical informatics*, 34(6):428–439, 2001.

[38] Xian Liu. *Survival analysis: models and applications*. John Wiley & Sons, 2012.

[39] A Mathew, M Pandey, and NS Murthy. Survival analysis: caveats and pitfalls. *European Journal of Surgical Oncology*, 25(3):321–329, 1999.

[40] Edward L Kaplan and Paul Meier. Nonparametric estimation from incomplete observations. *Journal of the American statistical association*, 53(282):457–481, 1958.

[41] Jason T Rich, J Gail Neely, Randal C Paniello, Courtney CJ Voelker, Brian Nussenbaum, and Eric W Wang. A practical guide to understanding kaplan-meier curves. *Otolaryngology-Head and Neck Surgery*, 143(3):331–336, 2010. 2.3

[42] M Biswal and Malaya Kumar. Statistical reliability analysis of interplanetary spacecraft operating at different interplanetary extremity. In *Region VII Student Paper Competition & AIAA Sydney Section Student Conference*, 2020. 2.3

[43] Nathan R Boone and Robert A Bettinger. Spacecraft survivability in the natural debris environment near the stable earth-moon lagrange points. *Advances in Space Research*, 67 (8):2319–2332, 2021.

[44] Mirko Trisolini, Hugh G Lewis, and Camilla Colombo. Spacecraft design optimisation for demise and survivability. *Aerospace Science and Technology*, 77:638–657, 2018. 2.3

[45] Matthew G Richards, Daniel E Hastings, Donna H Rhodes, and Annalisa Weigel. Defining survivability for engineering systems. In *Conference on Systems Engineering Research*, pages 1–12, 2007. 2.3

[46] Robert E Ball. *The fundamentals of aircraft combat survivability: analysis and design*. American Institute of Aeronautics and Astronautics, 2003.

[47] Gary L Guzie. Integrated survivability assessment. *ARLTR-3186, Survivability/Lethality Analysis Directorate, Army Research Laboratory*, 2004. 2.3

[48] Vickie R Westmark. A definition for information system survivability. In *37th Annual Hawaii International Conference on System Sciences, 2004. Proceedings of the*, pages 10–pp. IEEE, 2004. 2.3

[49] Somesh Jha and Jeannette M Wing. Survivability analysis of networked systems. In *International Conference on Software Engineering: Proceedings of the 23rd International Conference on Software Engineering: Toronto, Ontario, Canada*, volume 12, pages 307–317, 2001.

[50] Adel Ayara and Faiza Najjar. A formal specification model of survivability for pervasive systems. In *2008 IEEE International Symposium on Parallel and Distributed Processing with Applications*, pages 444–451, 2008. 2.3

[51] Ping Wang, Yan Li, and Chandan K Reddy. Machine learning for survival analysis: A survey. *ACM Computing Surveys (CSUR)*, 51(6):1–36, 2019. 2.3

[52] Jared L Katzman, Uri Shaham, Alexander Cloninger, Jonathan Bates, Tingting Jiang, and Yuval Kluger. Deepsurv: personalized treatment recommender system using a cox proportional hazards deep neural network. *BMC medical research methodology*, 18(1):1–12, 2018. 2.3

[53] Paul Baran. On distributed communications networks. *IEEE transactions on Communications Systems*, 12(1):1–9, 1964. 2.3

[54] Hervé Kerivin and A Ridha Mahjoub. Design of survivable networks: A survey. *Networks: An International Journal*, 46(1):1–21, 2005. 2.3

[55] Gangxiang Shen, Hong Guo, and Sanjay K Bose. Survivable elastic optical networks: survey and perspective. *Photonic Network Communications*, 31:71–87, 2016. 2.3

[56] Md Shohrab Hossain, Shaikh Shahriar Hassan, Mohammed Atiquzzaman, and William Ivancic. Survivability and scalability of space networks: a survey. *Telecommunication Systems*, 68:295–318, 2018. 2.3

[57] Michele Nogueira Lima, Aldri Luiz Dos Santos, and Guy Pujolle. A survey of survivability in mobile ad hoc networks. *IEEE Communications surveys & tutorials*, 11(1):66–77, 2009. 2.3

[58] Dongyan Chen, Sachin Garg, and Kishor S Trivedi. Network survivability performance evaluation: A quantitative approach with applications in wireless ad-hoc networks. In *Proceedings of the 5th ACM international workshop on Modeling analysis and simulation of wireless and mobile systems*, pages 61–68, 2002. 2.3

[59] Mohamed Al-Kuwaiti, Nicholas Kyriakopoulos, and Sayed Hussein. A comparative analysis of network dependability, fault-tolerance, reliability, security, and survivability. *IEEE Communications surveys & tutorials*, 11(2):106–124, 2009. 2.3

[60] Poul E Heegaard and Kishor S Trivedi. Network survivability modeling. *Computer Networks*, 53(8):1215–1234, 2009. 2.3

[61] Peter Stavroulakis, Maryna Kolisnyk, Vyacheslav Kharchenko, Nikolaos Doukas, Oleksandr P Markovskyi, and Nikolaos G Bardis. Reliability, fault tolerance and other critical components for survivability in information warfare. In *E-Business and Telecommunications: 14th International Joint Conference, ICETE 2017, Madrid, Spain, July 24-26, 2017, Revised Selected Paper 14*, pages 346–370. Springer, 2019. 2.3

[62] Ronald C Arkin and George Vachtsevanos. Techniques for robot survivability. In *Proc. 3rd International Symposium on Robotics and Manufacturing, Vancouver, BC*, pages 383–388, 1990. 2.4

[63] Ronald C Arkin. Survivable robotic systems: Reactive and homeostatic control. In *Robotics and remote systems for hazardous environments*, pages 135–154. 1993. 2.4

[64] QUEK BOON KIAT. A survivability framework for autonomous systems. 2008. 2.4

[65] Boon-Kiat Quek, Javier Ibanez-Guzman, and Khiang-Wee Lim. A survivability framework for the development of autonomous unmanned systems. In *2006 9th International Conference on Control, Automation, Robotics and Vision*, pages 1–6. IEEE, 2006. 2.4

[66] Changjun Li, Jingyu Yang, Xi Wu, and Guangwen Yang. Analysis of reliable architecture techniques for adaptive adversarial systems of system. In *2024 IEEE 24th International Conference on Software Quality, Reliability, and Security Companion (QRS-C)*, pages 619–624. IEEE, 2024. 2.4

[67] Aris Kanellopoulos, Kyriakos G Vamvoudakis, Vijay Gupta, and Panos Antsaklis. Dissipativity-based verification for autonomous systems in adversarial environments. In *Handbook of Reinforcement Learning and Control*, pages 273–291. Springer, 2021. 2.4

[68] Leonardo Bacelar Lima Santos, Luciana R Londe, Tiago José de Carvalho, Daniel S Menasché, and Didier A Vega-Oliveros. About interfaces between machine learning, complex networks, survivability analysis, and disaster risk reduction. *Towards mathematics, computers and environment: A disasters perspective*, pages 185–215, 2019. 2.4

[69] Yang Cai. Survivability. *Instinctive Computing*, pages 353–371, 2016. 2.4

[70] Roi Yehoshua, Noa Agmon, and Gal A Kaminka. Robotic adversarial coverage: Introduction and preliminary results. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 6000–6005. IEEE, 2013. 2.4, 7.1.1

[71] Roi Yehoshua, Noa Agmon, and Gal A Kaminka. Robotic adversarial coverage of known environments. *The International Journal of Robotics Research*, 35(12):1419–1444, 2016.

2.4

[72] Noa Agmon. Robotic strategic behavior in adversarial environments. In *IJCAI*, pages 5106–5110, 2017. 2.4, 7.1.1

[73] Kurt Klingensmith and Azad M Madni. Resilience concepts for architecting an autonomous military vehicle system-of-systems. In *Disciplinary Convergence in Systems Engineering Research*, pages 65–81. Springer, 2018. 2.4

[74] Brian K Hall. Autonomous weapons systems safety. *Joint Force Quarterly*, 86(3):86–93, 2017. 2.4

[75] Maxim Egorov, Mykel Kochenderfer, and Jaak Uudmae. Target surveillance in adversarial environments using pomdps. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016. 2.4, 7.1.1

[76] Miao Guo, Teng Long, Jingliang Sun, and Junzhi Li. Cooperative path planning method for enhancing ground-units survivability based on adaptive q-learning. In *International Conference on Autonomous Unmanned Systems*, pages 555–566. Springer, 2023. 2.4, 7.1.1

[77] Kiryong Ha, Zhuo Chen, Wenlu Hu, Wolfgang Richter, Padmanabhan Pillai, and Mahadev Satyanarayanan. Towards Wearable Cognitive Assistance. In *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*, 2014. 4.1

[78] Gary Doran, Steven Lu, Lukas Mandrake, and Kiri Wagstaff. Mars orbital image (hirise) labeled data set version 3. *NASA: Washington, DC, USA*, 2019. 4.2

[79] Malte Pedersen, Bruslund, Joakim Haurum, Rikke Gade, and Thomas Moeslund. Detection of marine animals in a new underwater dataset with varying visibility. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2019. 4.2

[80] NVIDIA. NVIDIA Jetson Orin. `https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/`, 2024. Last accessed July 22, 2024. 4.4

[81] Yuan H Wang. On the number of successes in independent trials. *Statistica Sinica*, pages 295–312, 1993. 4.8.1

[82] Jason Flinn and Mahadev Satyanarayanan. Energy-aware adaptation for mobile applications. *ACM SIGOPS Operating Systems Review*, 33(5):48–63, 1999. 4.10

[83] Falko Dressler and Gerhard Fuchs. Energy-aware operation and task allocation of autonomous robots. In *Proceedings of the Fifth International Workshop on Robot Motion and Control, 2005. RoMoCo'05.*, pages 163–168. IEEE, 2005. 4.10

[84] Arnav Vaibhav Malawade. *Context-Aware and Energy-Efficient Autonomous Systems*. University of California, Irvine, 2023. 4.10

[85] Jawad Naveed Yasin, Huma Mahboob, Mohammad-Hashem Haghbayan, Muhammad Mehboob Yasin, and Juha Plosila. Energy-efficient navigation of an autonomous swarm with adaptive consciousness. *Remote Sensing*, 13(6):1059, 2021. 4.10

[86] Mohmmadsadegh Mokhtari, Parham Haji Ali Mohamadi, Michiel Aernouts, Ritesh Kumar Singh, Bram Vanderborght, Maarten Weyn, and Jeroen Famaey. Energy-aware multi-robot task scheduling using meta-heuristic optimization methods for ambiently-powered robot swarms. *Robotics and Autonomous Systems*, 186:104898, 2025. 4.10

[87] Adam Seewald, Hctor Garca de Marina, Henrik Skov Midtiby, and Ulrik Pagh Schultz. Energy-aware planning-scheduling for autonomous aerial robots. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2946–2953, 2022. doi: 10.1109/IROS47612.2022.9981285. 4.10

[88] Carmelo Di Franco and Giorgio Buttazzo. Energy-aware coverage path planning of uavs. In *2015 IEEE international conference on autonomous robot systems and competitions*, pages 111–117. IEEE, 2015. 4.10

[89] Renan Maidana, Roger Granada, Darlan Jurak, Maurício Cecílio Magnaguagno, Felipe Rech Meneguzzi, and Alexandre de Morais Amory. Energy-aware path planning for autonomous mobile robot navigation. In *Proceedings of the 33rd International Florida Artificial Intelligence Conference, 2020, Estados Unidos.*, 2020. 4.10

[90] Omer Melih Gul. Energy-aware 3d path planning by autonomous ground vehicle in wireless sensor networks. *World Electric Vehicle Journal*, 15(9):383, 2024.

[91] Charlott Vallon, Mark Pustilnik, Alessandro Pinto, and Francesco Borrelli. Learning hierarchical control systems for autonomous systems with energy constraints. *arXiv preprint arXiv:2403.14536*, 2024.

[92] Giorgio Buttazzo, Mauro Marinoni, and Giacomo Guidi. Energy-aware strategies in real-time systems for autonomous robots. In *International Symposium on Computer and Information Sciences*, pages 845–854. Springer, 2004. 4.10

[93] B Wayne Bequette. *Process control: modeling, design, and simulation*. Prentice Hall Professional, 2003. 4.10.1

[94] Jan Achterbergh, Dirk Vriens, Jan Achterbergh, and Dirk Vriens. *Introducing organizations as social systems conducting experiments*. Springer, 2009. 4.10.1

[95] Heinz Unbehauen. *Control Systems, Robotics and Automation-Volume I: System Analysis and Control: Classical Approaches-I*. EOLSS Publications, 2009. 4.10.1

[96] Yanhui Zhou, Zhiwu Huang, Hongtao Liao, Heng Li, Yun Jiao, and Jun Peng. A predictive set-point modulation energy management strategy for hybrid energy storage systems. *IEEE Transactions on Industry Applications*, 55(6):6266–6277, 2019. 4.10.1

[97] MU Abuthahir and Nabil Magbool Jan. Time-varying setpoint tracking for batch process control using reinforcement learning. In *2024 18th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pages 1148–1153. IEEE, 2024. 4.10.1

[98] Daniel J Burns, Walter K Weiss, and Martin Guay. Realtime setpoint optimization with time-varying extremum seeking for vapor compression systems. In *2015 American Control Conference (ACC)*, pages 974–979. IEEE, 2015. 4.10.1

[99] Walter J Scheirer, Lalit P Jain, and Terrance E Boult. Probability models for open set recognition. *IEEE transactions on pattern analysis and machine intelligence*, 36(11):

2317–2324, 2014. 4.12

[100] Abhijit Bendale and Terrance E Boult. Towards open set deep networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1563–1572, 2016.

[101] Chuanxing Geng, Sheng-jun Huang, and Songcan Chen. Recent advances in open set recognition: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 43(10):3614–3631, 2020.

[102] Sagar Vaze, Kai Han, Andrea Vedaldi, and Andrew Zisserman. Open-set recognition: A good closed-set classifier is all you need? 2021. 4.12

[103] Abhijit Bendale and Terrance Boult. Towards open world recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1893–1902, 2015. 4.12

[104] Rocco De Rosa, Thomas Mensink, and Barbara Caputo. Online open world recognition. *arXiv preprint arXiv:1604.02275*, 2016.

[105] Fei Zhu, Shijie Ma, Zhen Cheng, Xu-Yao Zhang, Zhaoxiang Zhang, and Cheng-Lin Liu. Open-world machine learning: A review and new outlooks. *arXiv preprint arXiv:2403.01759*, 2024.

[106] Nie Hui, Wang Ruiping, and Chen Xilin. Open world object recognition and detection systems: Landscapes, challenges and prospects. *Journal of Computer Research and Development*, 61(9):2128–2141, 2024. ISSN 1000-1239. doi: 10.7544/issn1000-1239.202440054. URL https://crad.ict.ac.cn/en/article/doi/10.7544/issn1000-1239.202440054. 4.12

[107] radartutorial.eu. Radar Basics: Waves and Frequency Ranges. https://www.radartutorial.eu/07.waves/Waves%20and%20Frequency%20Ranges.en.html, 2015. Accessed: 2024-07-15. 5.1

[108] Neltronics. How does a Speed Camera or Radar Gun work? https://www.neltronics.com.au/how-does-a-speed-camera-or-radar-gun-work/, 2024. Last accessed December 31, 2024. 5.2(a)

[109] Einstein Online. Waves, motion, and frequency: the Doppler effect. https://www.einstein-online.info/en/spotlight/doppler/, 2024. Last accessed December 31, 2024. 5.1

[110] Ao Zhang, Farzan Erlik Nowruzi, and Robert Laganiere. Raddet: Range-azimuth-doppler based radar object detection for dynamic road users. In *2021 18th Conference on Robots and Vision (CRV)*, pages 95–102. IEEE, 2021. 5.2, 5.2, 5.3, 5.2

[111] Arthur Ouaknine, Alasdair Newson, Julien Rebut, Florence Tupin, and Patrick Pérez. Carrada dataset: Camera and automotive radar with range-angle-doppler annotations. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 5068–5075. IEEE, 2021. 5.2

[112] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory

Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. 5.3

[113] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018. 6.1, 6.3.1