REAL-TIME AUTONOMIC DECISION MAKING UNDER UNCERTAIN ENVIRONMENTS FOR UAV-BASED SEARCH-AND-RESCUE MISSIONS

by

VIDYASAGAR SADHU

A dissertation submitted to the

Graduate School–New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

Graduate Program in Electrical and Computer Engineering

Written under the direction of

Prof. Dario Pompili

and approved by

New Brunswick, New Jersey

October, 2020

ABSTRACT OF THE DISSERTATION

Real-Time Autonomic Decision Making Under Uncertain Environments For UAV-Based Search-And-Rescue Missions

By Vidyasagar Sadhu

Dissertation Director:

Prof. Dario Pompili

Real-time smart and autonomous decision making in an Intelligent Physical System (IPS), e.g., an autonomous car or Unmanned Aerial Vehicle (UAV), involves two main stages—*sensing* (collection and then transformation of sensor data into actionable knowledge by giving semantic meaning to the raw data) and *planning* (making real-time decisions using this knowledge). The *challenges* faced by an IPS during these two stages include coping with the various forms of uncertainty caused by the (non-stationary) changing environment in which the IPS acts. These sources of uncertainty can be broadly classified into the following categories—data-quantity and -quality uncertainty, model and parameter uncertainty, environmental uncertainty, multi-agent non-stationarity, communication uncertainty, partial observability, and computing-resource uncertainty. Further, the degree of uncertainty in all these sources changes with time, introducing yet another challenge.

In this dissertation, I propose novel solutions to deal with environmental, multi-agent non-stationarity, partial observability, and communication uncertainties, and present advanced techniques for real-time and autonomous operation of an IPS—or a group of IPSs—acting in a dynamic and unknown environment, primarily targeting UAV-based real-time autonomic search of objects/victims in a post-disaster scenario. To this end, I propose the following techniques needed to address the challenges mentioned above.

- *Multi-IPS coordination*—making high-level decisions in coordination with other UAVs acting in the environment in order to maximize the mission objective; this is challenging as the environment appears non-stationary from the view of any one UAV as other UAVs are taking actions independently. For this, I propose an actor-critic based Multi-Agent Deep Reinforcement Learning (MADRL) framework where the critic is trained in a centralized manner and the actor is decentralized and is used during deployment (testing).

- *Imperfect communication and partial observability*—the communication among the UAVs may not be perfect resulting in packet drops and delays; moreover, each UAV may not be able to observe the underlying state fully due to restricted field of view of the camera. For this, I propose to enhance the MADRL framework by augmenting it with Recurrent Neural Networks (RNNs) such as Long Short-Term Memory (LSTM) Neural Networks (NNs) to maintain state over time and to solve the partial observability/limited communication problem.

- *Context-awareness and validation*—*context* is a generic term used to denote the operating conditions e.g., environmental light/weather conditions, remaining operational time of a UAV, time, location, etc. It is well-known that context-aware decision making yields better results than decision making done without taking context into account; this means that it is important to validate the context (obtained from sensors) especially in some secure missions. For this, I propose a Hidden Markov Model (HMM) based context modeling and prediction engine leveraging the history of personal and group behavior of the UAV; the proposed model can be used to predict the current (and also future) context of the UAV, which can be used to validate the sensor context.

- *On-board proactive real-time monitoring and anomaly detection*—as the UAVs are operating in a dynamic/uncertain environment, it is paramount to monitor proactively the operation of the UAV. For this, I propose a Convolutional Neural Network (CNN) and bi-directional LSTM (bi-LSTM) NN-based multi-task learning framework for real-time anomaly detection using UAV scalar sensor (non-image) data, e.g., accelerometer and gyroscope obtained from the on-board Inertial Measurement Unit (IMU).

- *On-board diagnosis and anomaly identification*—after detecting that an anomaly has happened, it is important to also classify/identify/diagnose the type of anomaly so as to determine which (sequence of) corrective actions need to be performed. For this, I propose a CNN-biLSTM based deep network classifier for classification of (pre-known) anomalies using IMU data; I also profile the performance of these techniques when run on drone-mountable/comparable hardware such as NVIDIA Jetson TX2 Graphical Processing Unit (GPU), Raspberry Pi B, etc.

Finally, I thoroughly validate and assess the performance of all the techniques proposed in this dissertation towards enabling the application of real-time autonomous search missions using a group of UAVs operating under uncertain environments via computer simulations, hardware-in-the-loop emulations, and real-world experiments.

Acknowledgements

I would like to express my deepest gratitude and sincere appreciation to my Ph.D. advisor, Dr. Dario Pompili, for his constant support, guidance, and encouragement throughout the course of my doctoral study and research. I have always been inspired by his in-depth interdisciplinary knowledge as well as by his vision and aspiration for high-quality research. From Dr. Pompili I have learned how to always keep the bar high and to strive for the best possible achievement. I am honored to have been working under his supervision and I simply cannot thank him enough for his constant advice on the many aspects of a doctoral student's life and research career.

I would like to extend my gratitude to Drs. Saman Zonouz, Roberto Tron, and Bo Yuan for serving as committee members in my qualifying exam first, in my thesis proposal later, and ultimately in my dissertation defense. I sincerely thank them very much for their valuable comments and suggestions on various aspects of my research since the early days of my Ph.D program. I would also like to thank Drs. Teruhisa Misu and Akshay Nambi for being my collaborators and mentors during my three-month summer internships with Honda Research Institute USA, Mountain View, CA and with Microsoft Research, Bengaluru, India, respectively. Their invaluable industrial and research experience has helped me make strong connections between theoretical research and real-world problems.

I am also grateful to Drs. Waheed Bajwa, Shantenu Jha, and Salim El Rouayheb for serving as the committee members for my PhD qualifying examination and for their helpful feedback and advice, which have helped me in the latter part of my PhD career. I am also grateful to Drs. Saman Zonouz and Abdeslam Boularias for being my course instructors during the initial years of my PhD; they further strengthened my interest in Reinforcement Learning and Artificial Intelligence domains.

I would like to thank all the CPS Lab members and collaborators, Xueyuan Zhao, Mehdi Rahmati, Parul Pandey, Gabriel Salles-Loustau, Chuanneng Sun, Khizar Anjum, Yung-Ting Hsieh, Anthony Yang, Sanjana Devaraj, Mohammad Nadeem, Wenjie Chen, Ayman Younis, Parsa Hosseini, and Abolfazl Hajisami for the discussions and collaborations. My sincerest thanks to all the co-authors of my publications. I would like to also thank all my wonderful friends at Rutgers University, especially Rohith Jagannath, Vishwanath Sindagi, Sriharsha Etigowni, Amol Kumtakar, Sudarshan Kolar, Abhilash Narayana, Ravi Kumar, Naveen Yerratha, and Pankaj Shukla for making my Ph.D. student life enjoyable.

I am grateful for the financial support from the Department of Electrical and Computer Engineering, Rutgers University and from the Department of Homeland Security Science & Technology Directorate (DHS S&T) Cyber Security Division (Contract No. D15PC00159), Office of Naval Research (YIP Grant No. 11028418), National Science Foundation (NSF) (Grant Nos. NRI-1734362, RTML-1937403), which have provided me with the necessary resources to carry on research and finish this dissertation.

Last, but not least, I would like to thank my parents, Mr. Veeranjaneyulu Sadhu and Mrs. Anasuya Devi, my aunt, Mrs. Visalakshmi Sadhu, my uncle, Mr. Narasimha Murthy Sadhu, my brother and sister-in-law, Mr. Hanumantha Kumar Sadhu and Mrs. Satya Mrudula Vellala, my wife, Mrs. Sravani Prakki, my in-laws Mr. Bhagiratha Sarma Prakki and Mrs. Kalyani Prakki, and all my relatives for the continued understanding, moral support and encouragement. Their unconditional love and support have given me the strength to chase my dreams and aspirations. To them, I dedicate this dissertation.

Dedication

To my parents, uncle, aunt, brother, sister-in-law, wife, and in-laws

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview and Motivation

Real-time smart and autonomic decision making in an Intelligent Physical System (IPS) e.g., an autonomous drone or car, involves two major stages—*(1) sensing* (of sensor data and then transformation into actionable knowledge) and *(2) planning* (taking decisions using this knowledge). Sensing refers to reading data from on-board sensors such as Inertial Measurement Unit (IMU) or externally-mounted sensors such as LiDAR/Camera sensors [8–15]. Planning refers to operating the IPS at a high-level such as navigation, so as to execute the human-provided top-level instructions.

The *challenges* faced by an IPS during these two stages include coping with various forms of uncertainty, caused by the dynamic environment in which the IPS acts. These sources of uncertainty can be broadly classified into the following categories

1. *data-quantity and -quality uncertainty*: Data quantity uncertainty refers to how much data needs to be sampled from sensors in order to extract meaningful information and generate effective decisions in a timely fashion. Data quality uncertainty refers to noise associated with sensor and actuator data (control input), which could be due to faulty sensors/actuators or malicious sensors/actuators that may have been hacked. In general the challenges arising from data quantity/quality need to be dealt with jointly as they are intertwined; as a matter of fact, data-quantity and -quality uncertainties, if unchecked, may propagate up the "raw data$\rightarrow$information$\rightarrow$knowledge" chain and have an adverse effect on the relevance of the generated results.

2. *model and parameter uncertainty*: Model/parameter uncertainty refers broadly to

the uncertainty in what model/parameters need(s) to be used to process the data and generate outputs (e.g., a neural network vs. a Support Vector Machine (SVM) model to detect anomalies from sensor data). This can be distinguished into two types—parameter uncertainty and model uncertainty: knowing the model but not its parameters refers to parameter uncertainty, whereas not knowing the model altogether refers to model uncertainty. Model and parameter uncertainties are especially important to an IPS system such as a drone because of the next form of uncertainty viz., uncertain and dynamic environment in which it operates. While it is possible to fully model/control the process and environment for certain scenarios (e.g., industrial chemical plants) it is difficult to model the environment in case of a drone or car.

3. *environment uncertainty*: Environmental uncertainties refer to unknown and changing environment in which the IPS acts. These uncertainties can be either *accidental* or *malicious*. Examples of the former are hits of a drone by a bird resulting in breaking of blades, engine failures, etc. or real-life sudden events such as wind changes, all events that cannot be easily predicted. The latter encompasses intentional attacks such as a compromised/hacked sensor resulting in data uncertainties. Environmental uncertainties indirectly are the source of model and parameter uncertainties.

4. *communication uncertainty*: Communication uncertainty refers to imperfect communication with other IPSs acting in the mission. This could be due to several factors including network impairments, interference, geographical and privacy/security constraints.

5. *partial observability*: Partial observability is a challenge where the agent/IPS is not able to observe the entire state for decision making. It is only able to receive partial observations from the environment, using which, it has to make decisions. This could be due to limited field of view of the IPS camera, occlusions/obstructions, etc. In multi-agent settings, this could be due to limited communication among the agents.

6. *computing-resource uncertainty*: Computing-resource uncertainty refers to the non deterministic availability of sufficient computing resources needed for timely decision making. In case of a single IPS such as a drone, though this uncertainty refers to competition for CPU cycles among different applications running at the same time, this is not significant. This is especially important for a network/team of IPSs where the computation may be shared with other IPSs. Also, this form of uncertainty may not be key for non-real-time systems.

Notice that the degree of uncertainty in all these sources (data, models, and environment) changes with time, introducing yet another challenge.

1.2 Research Objectives

In this dissertation, I focus on dealing with environmental and communication uncertainties and partial observability. *The main research objective of this proposal is develop real-time and autonomic solutions to the problems/challenges mentioned above.* Here autonomic refers to properties such as *self-coordination*, *self-optimization*, and *self-healing.*

Research Objectives: The following are the research objectives: **(i)** take high-level instructions from humans and execute them autonomously—either via pre-programmed flight plans or complex Artificial-Intelligence (AI) algorithms—with minimal human intervention (which is needed to reduce human-induced errors and to deal with human resource scarcity). For example, two requirements in the case of autonomous data collection for disaster regions are coverage and privacy preservation (e.g., obscuring the faces of people who do not want to be found). The IPS should be able to provide full coverage of the entire situation in an autonomous manner by sensing the current state, reasoning on what needs to be captured (e.g., more information pertaining to relevant regions such as fire regions in case of forest fires or as indicated by humans), communicating with other IPSs, and acting appropriately to accomplish the mission. In case of a team of IPSs/drones mapping/sensing a region, a good coordination is needed among themselves so as to not collide, to divide the task at hand into multiple subtasks, and then to assign those subtasks to different agents in order to maximize the amount of useful information collected. In such cases, additional coordination with humans may also be needed. Furthermore, with privacy becoming an important requirement in people' lives, it is essential that the IPSs behave ethically by not capturing any private information of people who chose to be so.

(ii) specifically considering the application of 'search' missions using a group of UAVs, the first main challenge is represented by the fact that the environment is typically unknown and needs to be explored. Existing formulations represent the environment as a 2-D grid map (also known as a occupancy map), and use frontier-based/reinforcement learning-based approaches which are mostly limited to single agents. The second major challenge is the coordination of multiple agents. Technically, this is due to the fact that the environment, from the point of view of any given agent, is non-stationary, due the actions of other agents. In case of aerial multi-UAV SAR missions, most existing multi-agent techniques focus on the offline planning of routes that obtain the best coverage in an apriori fully known environment; such plans are not generalizable to new environments.

(iii) decision making that takes context into account yields better results; here the context can comprise of several attributes such as location, time, remaining operational time, environmental conditions such as light levels. In some secure missions, it is not recommended to rely solely on the context obtained from sensors (such as camera, GPS, etc.) as they could be hacked unknowingly. As decision making relies on context, it is very important to validate the context from sensors, before using it for decision making.

(iv) as the UAV(s) are operating in a dynamic and uncertain environment, it should proactively monitor its operation so as to detect if it has deviated from the normal operation i.e., if it has entered any danger mode. For example, in the case of a drone, if a propeller breaks down due to wear or any other reason, the drone should first detect that something abnormal has happened;

(v) it should identify/diagnose the type of misoperation so as to determine what (series of) corrective actions to pursue. For example, in case of a drone, the action to take differs between the cases of a gradual wind change and a malicious attack on the propellers causing an irregular Revolutions Per Minute (RPMs) of the rotor blades (e.g., opposite direction/different RPMs).

Contributions: In this dissertation, I propose solutions to address the above mentioned challenges and research objectives. In particular, novel reinforcement learning based techniques are proposed to impart self-coordination and self-optimization capabilities, while novel neural-network based techniques for proactive fault detection and classification bestow the self-healing property. Specifically,

(i) in order to address the multi-UAV coordination problem, as a preliminary step, I consider the example of coordinated (media) data collection (among drones and human bystanders) from a disaster scenario, based on human-provided directions such as whether to give importance to total coverage, or to some specific scenarios (e.g., fire regions in case of a fire accident). For this, I design a Multi-Agent Reinforcement Learning (MARL) framework based on Q-learning that uses a centralized Q-table for all agents (drones/human bystanders). This framework has been tested via simulations and shows that the exploration time significantly reduces as the number of agents participating in the solution increases.

(ii) the preliminary MARL framework proposed above has several limitations such as—(a) unable to scale to high-dimensional state and action spaces; (b) prone to single point of failure due to centralized updating of Q-table; (c) prone to divergence when distributed/local Q-tables are considered due to the non-stationary issue—this is because the environment appears non-stationary from the view of any one agent as other agents are taking actions independently, violating the Markov conditions required for convergence. In order to address these limitations, I propose and implement a distributed actor-critic based Multi-Agent Deep Reinforcement Learning (MADRL) framework for multi-agent real-time coordination with special attention to the application of victim/object search in dynamic and unknown environment such as a post-disaster scenario. The proposed framework leverages deep reinforcement learning to address the scalability issues and "centralized training with decentralized execution" approach to address the remaining two issues.

(iii) in order to validate the context obtained from sensors (for secure missions), I propose two stochastic models based on the theory of Hidden Markov Models (HMMs)—*personalized model* (*HPContext*) and *collaborative filtering model* (*HCFContext*). The former predicts the current context using sequential history of the UAV's past contextual observations; the latter enhances *HPContext* with collaborative filtering features, which enables it to predict the current context of the primary UAV based on the context observations of UAVs related to the primary UAV, e.g., UAVs operating in the same mission, following same paths, etc. Furthermore, in collaborative filtering based multi-party settings as above, where the private information of different entities is used to collectively build a model as above, it is important to preserve each party's privacy i.e., hide the information of one party from the other. For this, I propose a a privacy-preserving method to derive *HCFContext* model parameters based on the concepts of homomorphic encryption.

(iv) in order to bestow the self-healing capability, I first propose a deep multi-task learning based anomaly detection framework that takes scalar sensor (e.g., accelerometer, gyroscope, etc.) data to detect if the UAV has entered any potentially abnormal situation/danger mode. The proposed multi-task anomaly detection framework consists of two tasks which complement each other, in that, one task acts as a regularizer for the other so as to improve the former's performance and vice-versa. The first task consist of Convolutional Neural Network (CNN) and bi-directional Long Short Term Memory (bi-LSTM) Neural Network autoencoder, while the second task consists of CNN and bi-LSTM symbol predictor. While this work is originally developed to detect anomalies in driving data, I borrow some of the ideas proposed here for UAV anomaly detection discussed in the next point. The proposed deep multi-task learning framework has been tested on 150 hours of driving data collected by the autonomous car division of Honda Research Institute USA. The framework has shown promise in detecting different types of anomalies such as sudden braking and abnormal lane changes. It also performed better when compared to baseline approach such as a simple LSTM autoencoder and its multi-class variant.

(v) I extend the above framework for UAV fault detection and classification. In particular, I design a CNN and bi-LSTM neural network autoencoder to detect abnormal

UAV operation and a CNN-biLSTM neural network classifier to identify/classify the type of anomaly the UAV has encountered. Since it is not feasible to list all the types of anomalies that a UAV will encounter beforehand, we assume a subset of anomalies viz., all the combinations of broken propeller scenarios (one/two/three propeller breakdowns) to collect data and test our classifier. To show the real-time capabilities of our algorithms, We have profiled these algorithms on a drone-comparable/mountable hardware such as Nvidia Jetson TX2 GPU to find out the times required for inference. We show that the inference times are appropriate for real-time operation/intervention.

(vi) I validate and assess the performance of all the above proposed solutions especially the self-coordination, self-optimization, self-healing and real-time aspects via computer simulations, hardware-in-the-loop emulations and real-world experiments.

1.3 Dissertation Organization

The rest of this dissertation is organized as follows.

Chapter 2 details our solution for MARL based autonomous and coordinated data collection from disaster regions. The proposed Multi-Agent Reinforcement Learning (MARL) framework allows for coordinated (media) data collection (among different drones and human bystanders) from a disaster scenario, based on human-provided directions such as whether to give importance to total coverage, or to some specific scenarios (e.g., fire regions in case of a fire accident). This proposed MARL framework has been tested via simulations and results that the exploration time significantly reduces as the number of agents (drones/human bystanders) participating in the solution increases.

Chapter 3 presents our approach of designing a novel actor-critic based MADRL framework for autonomous aerial exploration of an area for target detection using a group of UAVs. In this approach, each UAV maintains a neural-network based actor and critic networks, where the critic is trained by augmenting it with the state and action information of all other UAVs during training. Such a critic is used to update the parameters of the actor network during training. During testing, only the actor network is used without using the actions of other UAVs. We also augment the actor and critic networks with LSTMs so as to deal with imperfect communications and partial observability scenarios. We test our approach via Python simulations in a grid-world based environment with full, partial observability and limited communication scenarios.

Chapter 4 presents our privacy-preserving, sequential history-based personalized and collaborative-filtering techniques for context modeling and prediction. Specifically, we first propose a stochastic model based on Hidden Markov Model (HMM) theory for learning the personal sequential patterns from the contextual history of a UAV. We then enhance it by also learning the UAV's group contextual patterns via HMM based collaborative filtering. Lastly, we also present privacy-preserving algorithms for multi-party parameter estimation of the proposed HMM models using homomorphic encryption. Our approach can be used to validate and/or enhance the sensor context in certain secure missions. Their feasibility for practical deployment is shown by evaluating on a real-life dataset with over 80% accuracy.

Chapter 5 details our solution for anomaly detection via deep multi-task learning framework consisting of CNN and bi-LSTM autoencoder and a symbol predictor, that takes scalar sensor data as input. The proposed deep multi-task learning based anomaly detection framework takes scalar sensor (e.g., accelerometer, gyroscope, etc.) data to detect if the IPS has entered any potentially abnormal situation/danger mode. The framework consists of two tasks which complement each other, in that, one task acts as a regularizer for the other so as to improve the former's performance. The first task consist of Convolutional Neural Network (CNN) and bi-directional Long Short Term Memory (bi-LSTM) Neural Network autoencoder, while the second task consists of CNN and bi-LSTM symbol predictor. The proposed deep multi-task learning framework has been tested on 150 hours of driving data collected by the autonomous car division of Honda Research Institute. The framework has shown promise in detecting different types of anomalies such as sudden braking and abnormal lane changes. It also performed better when compared to baseline approach such as a simple LSTM autoencoder and its multi-class variant.

Chapter 6 details our solution for UAV anomaly detection and classification via CNN-biLSTM deep network classifier with scalar sensor data as input. The proposed

CNN-biLSTM neural network classifier identifies the type of anomaly the UAV has encountered. Since it is not feasible to list all the types of anomalies that an IPS will encounter beforehand, we assume a subset of anomalies viz., all the combinations of broken propeller scenarios (one/two/three propeller breakdowns) to collect data and test our classifier. We have profiled the algorithm on a drone-comparable/mountable hardware such as Nvidia Jetson TX2 to find out the running times for inference. We show that the running times are appropriate for real-time operation/intervention.

Chapter 7 provides a conclusion to the thesis and also presents future research directions to further extend the state-of-the-art for each of the above chapters.

Chapter 2

Autonomous Data Collection via Multi-Agent Reinforcement Learning for Disaster Situational Awareness

Disaster management involves a very large number of heterogeneous agents in an uncontrolled and potentially hostile environment. These agents include victims and the rescue personnel that eventually intervene. Situational-awareness techniques involving better incident management strategies would help to reduce disaster damage.

2.1 Introduction

Motivation: Traditionally, incident reporting relies on expensive information collection from authorities. These reportings happen most often in an offline manner and fail to provide ahead of time actionable information to incident response team. However, in these situations it is common to find people present at those locations (bystanders/onlookers) to record the scene using their smartphones. Very few of the existing incident response frameworks take advantage of this valuable information. We argue that this information is valuable because it is first hand, real time and local, and could provide fine-grained information about the scene. Also, in some cases, this information can be complemented by drones taking pictures or videos of the disaster scene.

Our Approach: Contrary to existing solutions such as those based on CCTV cameras, our solution, *Argus* (from *Argus Panoptes*, the “all-seeing" 100-eye giant in the Greek mythology), utilizes this valuable information coming directly from the bystanders and also provides a base framework wherein both the drones/robots and humans can collaboratively work to provide valuable and fine-grained information to the rescue authorities. Argus is implemented as a mobile application that can be installed on the

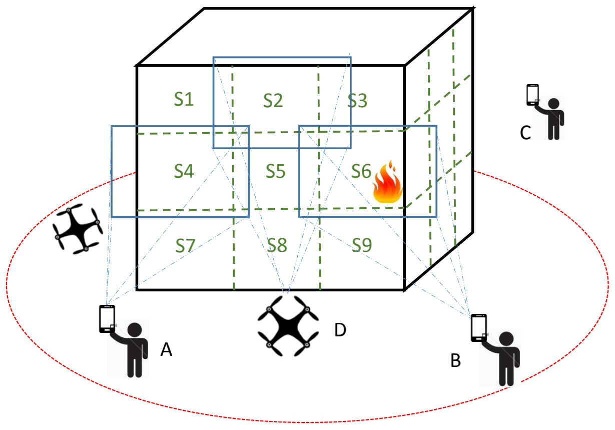


Figure 2.1: Argus use case: a building on fire and its modelization with human agents (A,B,C) and flying drones (D).

users' smartphones to capture and process data such as images of the incident scene. We then consolidate this valuable data into an easy-to-visualize form such as a 3D map of the disaster scene in real time. The reasons to consider 3D maps over scanning individual images is as follows: (i) it is difficult to process individual images one by one as they could be in huge number; (ii) there could be redundant/duplicate images e.g., same image from different angles;However, these same reasons become advantageous for 3D reconstruction as it needs several images from different views. Also, generating a 3D model from a sequence of images is much cheaper than using other techniques such as 3D scanners. We believe that integrating other well-known technologies such as audio source separation into our framework can add value (such as to prioritize some regions over others). We will investigate it as part of our future work. In this chapter, we make use of a Multi-Agent Reinforcement Learning (MARL) framework for adaptive and effective data collection. The reasons for considering MARL are as follows: (1) to enable full coverage of the scene, without which most of the images would correspond to a few conspicuous regions only (e.g., fire regions); (2) to enable fine-grained (high resolution in 3D maps) coverage in regions of interest specific to the operating personnel (e.g., behind the building). A high-level overview of our framework can be seen in Fig. 2.2.

Main contributions can be summarized as follows:

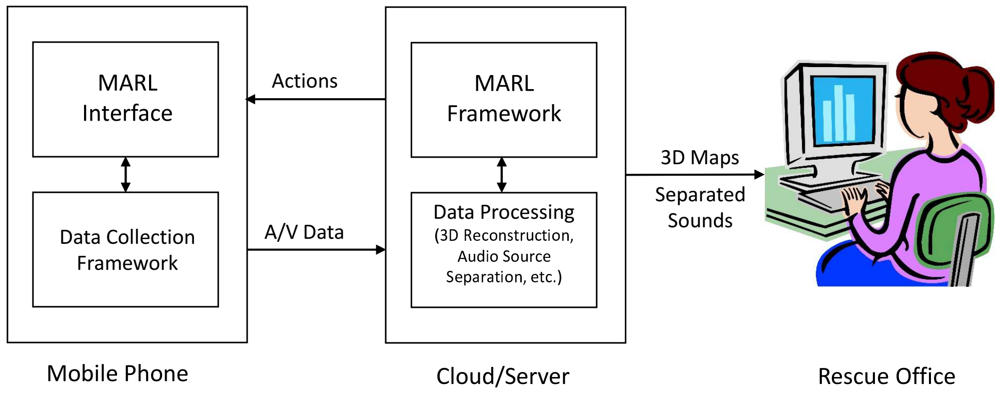


Figure 2.2: Argus high-level architecture.

- A Multi-Agent Reinforcement Learning (MARL) based framework for real-time coordinated incident response data collection using both human-bystanders and drones.
- Evaluation of the MARL framework through simulations to find optimal design parameters and to study its behavior for random fire occurrences.

Chapter Outline: In Sect. 2.2, we position our work with respect to state-of-the-art in related research. In Sect. 2.3, we present the Argus architecture and the system design including the MARL and Q-learning frameworks. In Sect. 2.4, we study our framework by varying the design parameters and also evaluate its performance for random fire occurrences. Finally, we summarize the chapter in Sect. 2.5.

2.2 Related Work

MARL has been investigated for various emergency scenarios previously. For example, the applicability and usefulness of MARL to building evacuation simulations for emergency/disaster situations and to model a large building containing multi-agent heterogeneous population attempting to evacuate in the presence of a non-stationary fire is investigated in [16]. In [17], design and development of a mobile system is presented that reconstructs 3D model of a building's interior structure in real time and fuses the visualization with the image of a thermal camera. RescueNet [18] proposes a networking paradigm based on MARL to enable reliable and high data-rate wireless multimedia communication among public safety agencies using mission policies that enable graceful degradation in the Quality of Service (QoS) of the incumbent networks in times of emergency. The RoboCup Rescue project [19] hosts yearly competitions to promote research and development in the rescue management of disaster situations such as earthquakes, fires, etc. The goal of the agent competition is to implement control policies for emergency services such that the maximum number of lives are saved and fires extinguished.

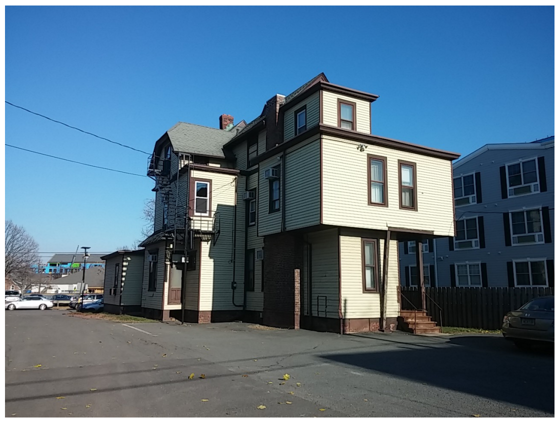
Several works focus on modeling incident response operations but very few can be used in real disasters as there are several challenges to be tackled in practical situations [20–29]. Various solutions for real-time monitoring of incident zones including interactive systems are proposed in [30, 31]. Design and implementation of a framework to simulate incident response such as building evacuation with modeling victims as agents with specific set of properties such as their visibility is presented in [32]. A multiple-agent Markov Decision Process (MDP) model is leveraged to synchronize the actions of the response team after an incident in [33]. None of these frameworks use MARL for data collection from the bystanders to create 3D maps as we do.

Regarding MARL, there are two types of frameworks, cooperative and competitive [34]. In the former the agents work cooperatively towards a common goal (with same rewards), while in latter agents compete against each other with opposite rewards (e.g., the sum of the rewards of all agents is zero). Our scenario corresponds to the former one. A simple MARL framework for cooperative agents without assuming any coordination among the agents is proposed in [35]. Each agent maintains a local Q-table and a policy, both of which are updated only if there is an improvement in the Q-values. However, for each local (state, action) pair, it requires that the other agents' actions happen infinitely often, which may not be practical in time-constrained scenarios. The idea of Q-value sharing is proposed in [36, 37]. We make use of similar concept except that all agents share the same Q-table instead of individual tables with knowledge sharing as they propose, and also our application setting is different from theirs.

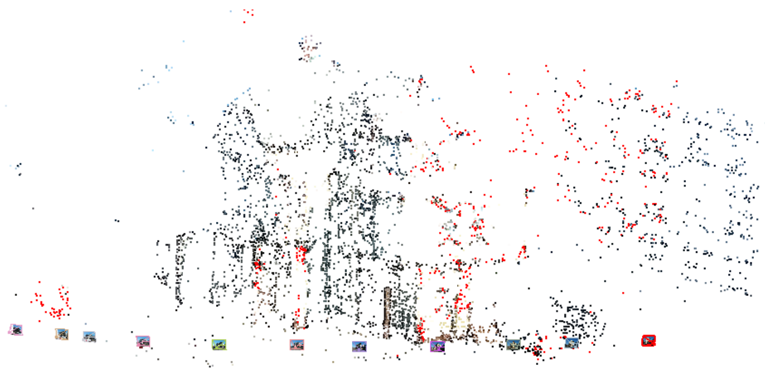
2.3 Proposed Solution

Argus use case: Fig. 2.1 presents a use-case scenario for Argus, which shows a building (represented by cube) on fire. Other use cases could be large forest fires, flooding, a terrorist attack, etc. Argus is installed as a mobile phone application and directs bystanders present at the incident zone on what actions to take (e.g., what pictures to take using their smartphones) to gather relevant information pertaining to the incident zone. This is achieved as follows. Argus decomposes the incident zone into multiple subzones with sufficient overlap. For example, in the case of building on fire, each subzone could correspond to a face or corner of the building with corners enabling overlap between the faces of the building. Each subzone is further divided into a rectangular grid consisting of states $S_0, ...S_9$, as shown in Fig. 2.1. The grid size is determined based on several factors such as the area of the incident scene, distance between the incident scene and the bystanders, etc., which we will investigate in more detail in future work. Each grid is cast as a Markov Decision Process (MDP), which is solved using MARL (specifically Q-learning as we will explain later) with agents being the bystanders around the incident zone (A,B,C) and any drones (D) that can assist humans in the data-gathering process. RL is a model-free process in that, it has the advantage of not knowing the state transition probabilities in advance. The actions are the standard actions of a grid world environment e.g., “Up”, “Down”, “Right”, “Left”. The Argus mobile application, which interfaces with the MARL framework running in the cloud, directs the agents (humans or drones) on what information to collect from the incident zone by giving directions such as “Left," “Up," etc. The drones could take part in the process just as humans or could complement them by capturing information that may not be accessible to humans (for example, a forest/dense plant covering along one face of the building that is not accessible to humans) or they could be assigned special tasks to capture information in regions where it is incomplete and immediately needed. As mentioned earlier, we have chosen to create a 3D map of the disaster scene, in which case the information gathered from the agents will be images or videos of the disaster scene. We aim at providing near real-time 3D mapping of the disaster scene so that the rescue personnel are aware of the entire situation, know where to immediately focus their attention on or plan appropriately. We used a GUI program called VisualSFM [38] for the 3D reconstruction, which consists of two parts, a well-known Structure From Motion (SFM) algorithm [39] for sparse 3D and CMVS/PMVS [39] for dense 3D reconstruction. Fig. 2.3(b) and Fig. 2.3(c), respectively, show the sparse and the dense 3D reconstruction of a building (Fig. 2.3(a)) using SFM and PMVS/CMVS packages found in VisualSfM. We have used 12 images of the building taken from different views using LG D800 smartphone with 1 MP resolution. We would like to mention that in real incidents it is realistic to assume to have hundreds of images, making the reconstruction more detailed. Also, there will be a higher resolution in areas of interest such as fire regions, damage regions, etc. as they will be the reward states in MARL, as we explain below.

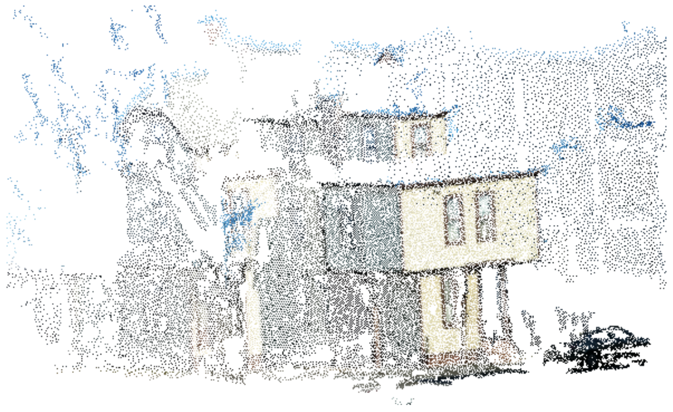
Argus architecture: Our framework consists of the following parts, as shown in Fig. 2.2: 1. A client-side data-collection framework that is part of the mobile application for collecting image and/or audio data from the users smartphone (audio data can provide additional situational awareness information about the incident zone e.g., baby cries which needs to be given more importance) 2. A server side, MARL framework that interfaces with the mobile application (MARL Interface) and directs the agents on what parts of the disaster scene to capture by providing directions such as “Left," “Up," etc. 3. A server-side data-processing framework that uses the images and/or audio data sent by the clients to generate 3D maps and other processing tasks (such as audio source separation). This processed data in the form of 3D maps is then sent to rescue office personnel who are in a position to guide rescue personnel using this information. The MARL framework directs the agents (humans/drones) to capture photos in a proper way, i.e., capturing a few photos to give the context and many others primarily focused around the regions of interest. Our framework achieves this by assigning more rewards to the data corresponding to interest regions. We make use of distributed Q-learning to solve our MARL problem. During the *exploration phase* of Q-learning, the agents explore the entire disaster scene to find reward states. This phase ends when the Q-values converge and simultaneously helps create a coarse 3D map of the entire scene. Then, during the *exploitation phase*, more time is spent in reward states to create a



(a)



(b)



(c)

Figure 2.3: (a) Original Image (for 3D reconstruction); (b) Sparse 3D reconstruction showing camera positions at the bottom; (c) Dense 3D reconstruction.

finer 3D maps of those regions. We chose Q-learning as it is model free and can adapt to changing environments (such as fire propagation).

System Design: We now describe our MARL framework consisting of a distributed

Q-learning approach for exploration and of a periodic exploration-exploitation mechanism to capture the dynamicity of the environment. To reiterate, our MARL framework is used for data collection only. As the images are obtained, 3D reconstruction is done using those images in real time, as mentioned in Sect. 2.3.

MARL Framework: Although our framework is quite general and can be applied to any scenario, in order to be concrete we will continue with our example of "building on fire" scenario as in Fig. 2.1. We formulate the problem as a MDP as follows. We divide the building into 8 surface views consisting of 4 faces and 4 corners assuming a regular cuboid structure. Each surface and corner are modeled as a rectangular grid, and the bystanders present near that view are assigned to that particular MDP making that process a Multi-Agent System (MAS). In case of corners, the grid is created by stitching some portions from both the constituent faces to enable overlap with the faces. Each MDP acts independently of others. Images from all MDPs are fused to form the 3D map of the building. The *states* are the cells (x,y) of the grid. For now, let us assume that the humans (agents) participating in our framework are static and can only pan or tilt their phones. An agent is said to be in a state (x,y) if it positions its smartphone camera focusing/centering on the cell (x,y). The *actions* for each state are that of a standard grid-world MDP: {Left, Right, Up, Down}. Here, Left and Right are realized by *panning* (rotating along vertical axis) the phone by a specific angle θ_x in those directions. Two continuous Right actions would mean panning by $2\theta_x$ to the right and so on. On similar lines, Up and Down are realized by *tilting* (rotating along horizontal axis) the phone by a specific angle θ_y. We could make use of gyroscopes in the phone to provide feedback to the users on when to stop panning (or tilting) the phone when they have covered an angle of θ_x (or θ_y) in the specified directions. Whenever an agent takes an action to move to a different state, it captures the image of the current state (grid cell in the MDP) before panning/tilting the phone. This image determines the *reward* the agent receives for that state. In other words, our reward is dependent on the state alone rather than on the state-action-state sequence. Since the regions with more fire are of primary interest, more rewards are assigned if a higher percentage of fire pixels is detected in the image. In order to avoid cheating by the bystanders (in the event of providing incentives to bystanders to use our application, which we will consider in our future work), we normalize this by the used zoom factor (Eq. 2.1) to bring to a standard zoom value. We note that the reward can be changed as per the application requirements. If uniform coverage is the final goal, incomplete regions will be reward states (e.g., images corresponding to cells with a low number of visits so far). Using this approach, the algorithm can gear the agents to capture images of interest to the user, where our reward is,

$$r(s) = \frac{\%(\text{fire-pixels})}{\text{zoom-factor}}. \tag{2.1}$$

Finally, the size of the grid and θ_x, θ_y will be determined by these factors: the dimension of the incident zone, the average distance of the agents from the incident zone, and the average capture angle of the agents' devices.

1) Distributed Q-Learning: Q-learning [5] consists of learning/updating Q-values which are also called 'state-action' values. Q-value for a particular state and action pair and policy, indicates the long term reward the agent receives by taking that action in that state and following that policy thereafter. In Q-learning the agent learns from each experience as it interacts with the environment and updates its Q-values based on the reward that it gets from this experience. Let us denote the set of all states s as S, the set of all actions a as A, and the reward as r. A single experience with the environment at time t can be represented by the tuple (s_t, a_t, r_t, s_{t+1}). This means the agent was in state s_t, took action a_t, received reward r_t, and landed in state s_{t+1}. The agent learns (updates its Q-values) through a series of such experiences. We discount the rewards obtained from future experiences as we consider the present rewards to be more valuable than the future ones (which are more uncertain due to the probabilistic nature of RL). Let us denote the Q-value of a state-action pair at time t as $Q_t(s_t, a_t)$. With the experience from (s_t, a_t, r_t, s_{t+1}), the Q-value can be updated as follows,

$$Q^{\pi}_{t+1}(s_t, a_t) \leftarrow Q^{\pi}_t(s_t, a_t) + \alpha[r_t + \gamma max_a Q^{\pi}_t(s_{t+1}, a) - Q^{\pi}_t(s_t, a_t)], \tag{2.2}$$

where π is the policy being followed, α is the learning rate and γ is the discount factor. As can be seen from (2.2), Q-values are updated based on the immediate reward, r_t, and on the optimal expected return, $\gamma max_a Q_t(s_{t+1}, a)$. Note that the learning rate α is set to a large number in the beginning and is slowly reduced to ensure convergence.

Given that we have a MARL problem, we make use of distributed Q-learning (slightly modified from [36,37]) where the agents share the same Q-values using a shared database (with synchronous read/write) to expedite the exploration process. We would like to clarify that it is still a multi-agent system except that the Q-values are the same for all. Each agent still acts independently. As all agents are exploring in parallel and making use of the knowledge/experience gained from all other agents (Q-values), the exploration process is greatly sped up.

2) Periodic Exploration: There are two well-known exploration strategies: ϵ-greedy approach and Boltzmann exploration. Even though both strategies drive the randomness in selecting action to reduce over time, there is a disadvantage with the former as ϵ-greedy gives equal importance to all actions in case it needs to select a random action (with probability ϵ) with no regard to the Q-values. Let us say there are 3 actions with decreasing Q-values other than the optimal action, which has even higher Q-value. In case of ϵ-greedy, all 3 actions are given equal importance whereas it would be desirable to assign a probability that is directly proportional to the Q-value. Boltzmann exploration achieves this, where the probability of selecting an action at time t ($\pi_t(a|s)$) is directly related to its Q-value (proportional to $e^{Q_t(s,a)/T}$),

$$\pi_t(a|s) = \frac{e^{Q_t(s,a)/T}}{\sum_{a' \in A} e^{Q_t(s,a')/T}}. \tag{2.3}$$

Here, the temperature parameter, T, decides the amount of exploration or exploitation; if T is large, all actions have almost equal probability resulting in pure exploration whereas when T tends to 0 the optimal action has the highest probability, resulting in pure exploitation. In order to adapt to the dynamicity of the environment, we enable periodic exploitation. In each period, there is a short exploration phase followed by a long exploitation phase. Accordingly, we vary the exploration parameter (T) and the learning rate (α) in a periodic fashion (Fig. 2.4). We note there are two extremes to this mechanism. On one extreme there is a need to account for random occurrences of fire in scenarios where the environment changes randomly, e.g., during a terrorist

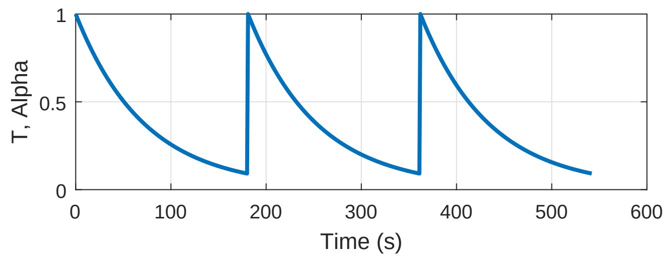
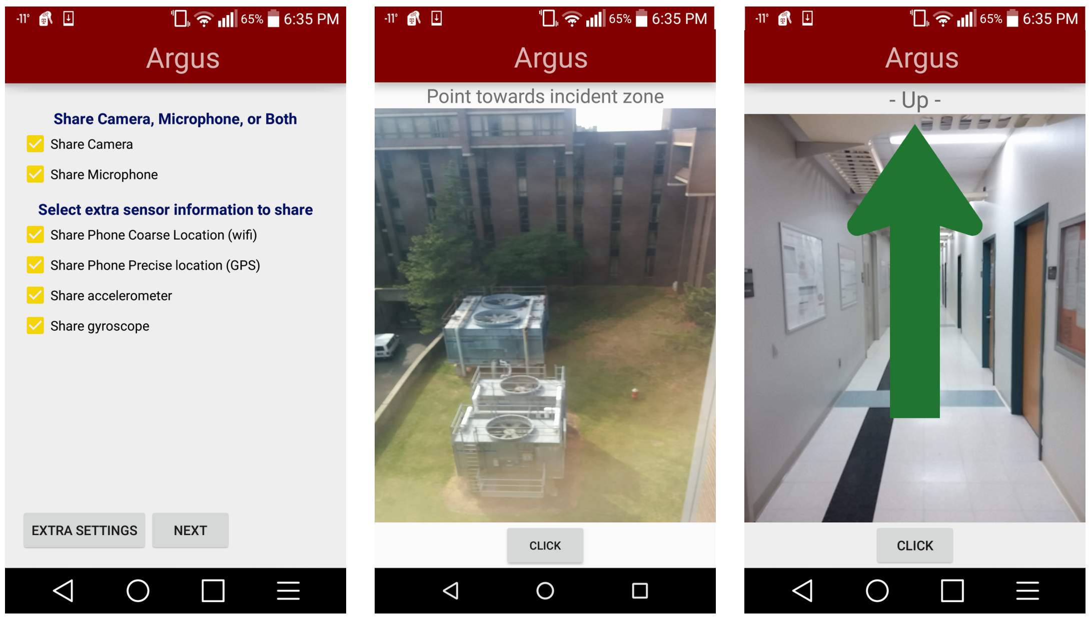


Figure 2.4: Variation of T, α to perform periodic exploration.



Share Camera, Microphone, or Both

Share Camera

Share Microphone

Select extra sensor information to share

Share Phone Coarse Location (wifi)

Share Phone Precise location (GPS)

Share accelerometer

Share gyroscope

(a) (b) (c)

Figure 2.5: Argus android app: (a) sensor options User Interface (UI); (b) initial instruction; (c) camera UI and feedback (Up/Down/Right/Left).

attack; in these cases, we reset the Q-values and the exploration parameter (Boltzmann temperature) across periods to enable new learning. On the other extreme, we have situations such as slow fire propagation where there is no significant change in the environment across periods; in this case, it is preferable to carry forward the Q-values to the next period so to leverage the learning in the previous period. The exploration parameter, T, too can be reduced quickly in the next period to enable less exploration and more exploitation. In both cases, the period will depend on the rate of dynamicity of the scene.

Smartphone Application: Fig. 2.5 shows some screen-shots of our application.

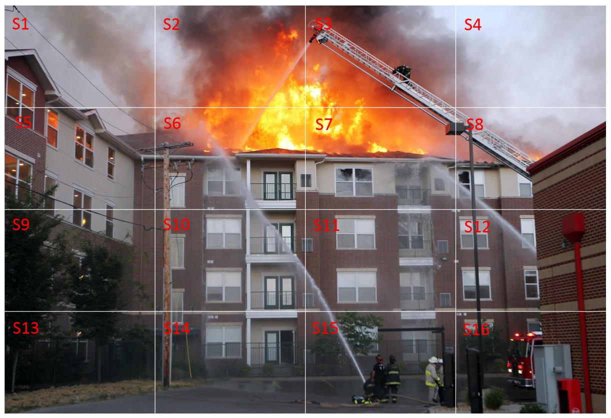


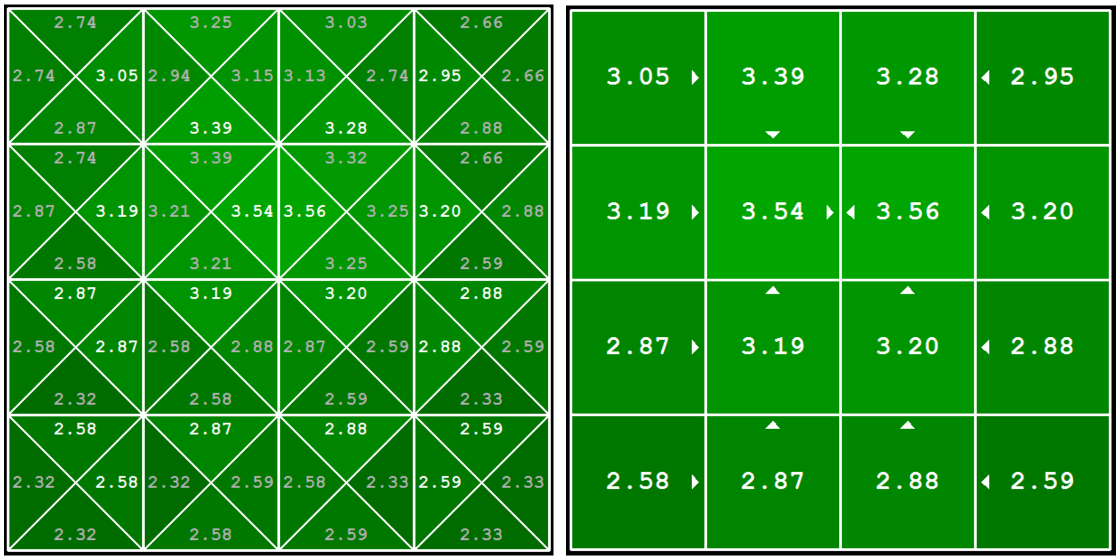
Figure 2.6: Incident scene [1] with 4x4 MDP grid shown in white.

We enable the users to share other sensor information such as camera, microphone, GPS, accelerometer, gyroscope as this will help in the 3D reconstruction (Fig. 2.5(a)); Fig. 2.5(b) shows the camera interface asking users to point towards incident zone. After capturing the image, Fig. 2.5(c) shows the action to tilt 'Up' (indicated by UP arrow) before taking the next picture. The app has an option to specify the upload server. In a real deployment, this will be the IP address of the central server in Fig. 2.2.

2.4 Performance Evaluation

We evaluated our MARL framework based on distributed Q-learning in terms of key metrics such as percentage of time spent by agents on capturing the fire states (reward-states) vs. non-fire states as the parameters of the framework including the number of agents, decay rate of Boltzmann temperature, and learning rate are varied; also, we tested how well the algorithm adapts to random incidents. We coded the MARL framework in Python using distributed Q-learning on a 4x4 MDP grid shown in Fig. 2.6 with $S13$ as the start state.

We remind the reader that the fire region will be divided into multiple MDPs where each MDP corresponds to a single surface of building, a corner of the building, or some other view of the incident scene. A rectangular grid will be overlaid on each of the views where the ensuing MDP will be solved. Fig. 2.6 shows an example of this scenario (the picture is sourced from the Internet [1]). A 4x4 grid is overlaid on the incident zone,



(a) (b)

Figure 2.7: (a) Final Q-Values calculated by our framework using distributed Q-learning; (b) Final values (which are maximum Q-values in each state) calculated by our framework with arrow heads showing the optimal actions for each state. We can notice that fire states have higher values compared to the others and the optimal policy indicated by arrow heads leads the agents to the fire states (middle two cells in the second row) where they stay most of the time.

which means we have 16 states/cells in the MDP. Each agent's action in a state will include capturing a picture of the state (centering smartphone on the state) along with the physical action of panning/tilting. Since we do not have pictures of the incident scene in Fig. 2.6, we approximated the image of each state to the grid piece obtained by dividing the picture in Fig. 2.6, as per the white grid lines. Unlike in this simplified scenario, in reality we would have multiple pictures of each state captured by different agents at different times and the reward would be calculated as the percentage of fire pixels in those images. Using these approximated images (grid pieces), we calculated the percentage of fire pixels in them and then normalized it to obtain the rewards for these states. We can observe that only states S2, S3, S6, S7 have non-zero rewards. We ran the distributed Q-learning with discount factor of 0.9 until convergence of the values

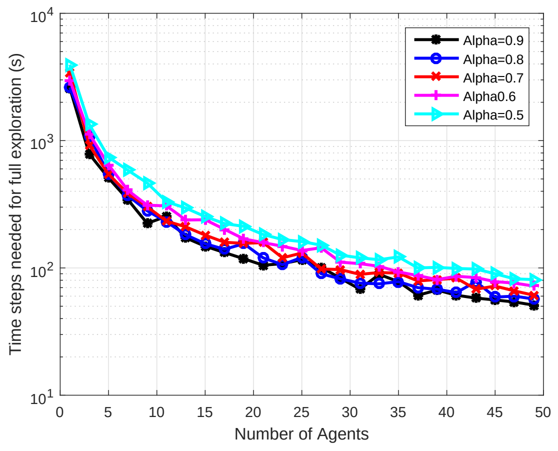
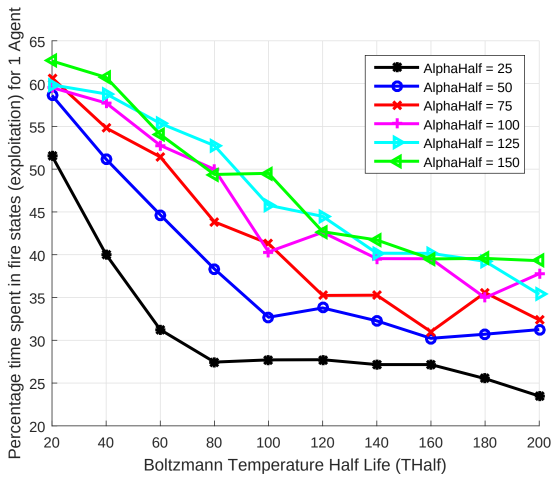


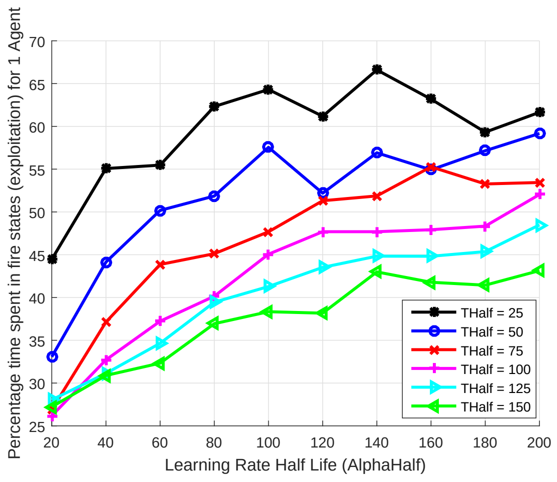
Figure 2.8: Time steps needed for convergence (i.e., full exploration) as the number of agents increases for different learning rates (we assume one second per move/step).

of the states. Note that all the agents share the same Q-values in our distributed Q-learning approach. The final Q-values and state values (which are maximum Q-values in each state) are shown in Fig. 2.7(a) and Fig. 2.7(b), respectively. We can notice from Fig. 2.7(b) that fire states have higher values compared to the others and that the rewards propagate to surrounding states. Also, the optimal policy indicated by arrow heads leads the agent to the fire states where it stays most of the time.

Exploration vs. number of agents: Fig. 2.8 shows the number of MDP time steps needed for full exploration of the 4x4 MDP grid shown in Fig. 2.6 for different values of fixed learning rate, α. We can see that the number of steps needed reduces drastically (from around 3000 to 80) as the number of agents increases, which shows the benefits of distributed Q learning and availability of multiple bystanders at the incident zone. We can also notice that $\alpha = 0.9$ is the best as the agents learn faster and take less number of steps to reach convergence. We also note that in practice there is no need to wait until full convergence of the values as policy converges faster than values (so the transition to the exploitation phase is faster). We also make an assumption that each time step in the Q-learning corresponds to about one second in wall-clock time. The decay in Boltzmann temperature T (same for α) is modeled as a radioactive decay, $T = T_{min} + (T_{max} - T_{min})e^{-\frac{\log 2}{T_{half}}t}$.



(a)



(b)

Figure 2.9: (a) Time spent in fire states [%] vs. temperature half life (1 agent); (b) Time spent in fire states [%] vs. learning rate half life (1 agent). These are for an allotted time of 180 timesteps. We can observe that the percentage of time spent on fire states reduces as T_{half} increases. This is because there is more exploration as T_{half} increases, thus reducing the time spent on fire states. We observe a similar behavior in Fig. 2.9(b).

Time spent in fire states vs. α and T: Fig. 2.9(a) shows the percentage of time spent in fire states as T_{half} is varied for different values of α_{half} for single-agent case with 180 time steps (3 minutes) (we have plotted the variation with different time steps in Fig. 2.11(b)). We know that T_{half} is inversely proportional to the decay rate. Large T_{half} values correspond to more exploration and less exploitation, and viceversa. We can observe that the percentage of time spent on fire states reduces as T_{half} increases. This is because there is more exploration as T_{half} increases, thus reducing the time spent on fire states. We observe a similar behavior in Fig. 2.9(b), which shows the percentage of time spent in fire states as α_{half} is varied for different values of T_{half} also for a single agent with three minutes of allotted time. These results are averaged over 50 runs to account for randomicity present in the action selection that is based on the values of T_{half}. The ratio of exploitation to exploration can initially be around 75:25. Later this can be adaptively changed based on fire propagation. For example, if fire is spreading more rapidly, we would want to change this ratio to 90:10. However, we note

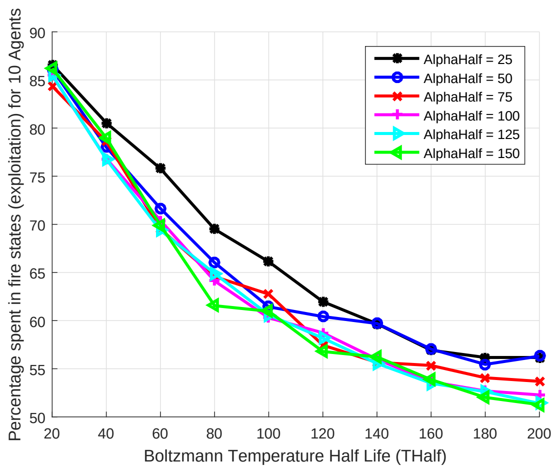
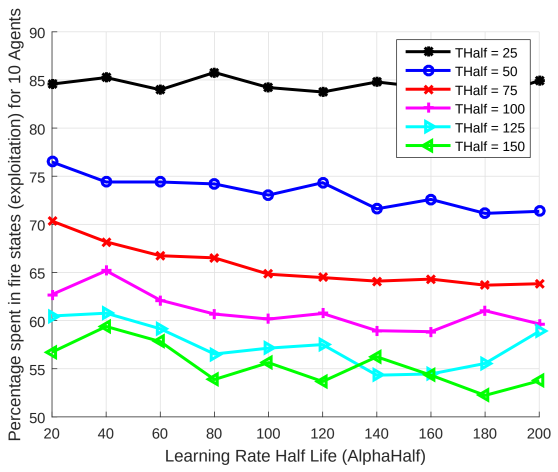


Figure 2.10: Time spent in fire states [%] vs. temperature half life (10 Agents).

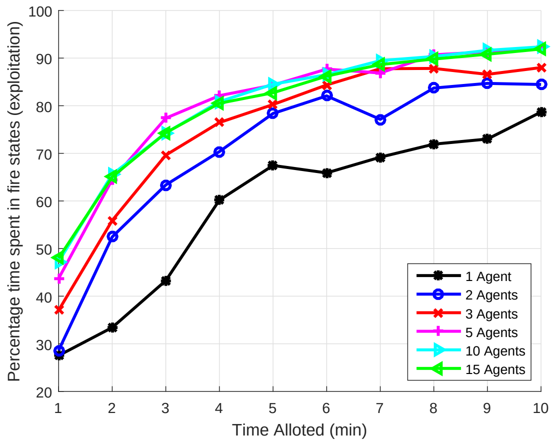
that a single agent is not able to reach 70% exploitation within the time allotted, but it is able to reach more than 75% exploitation when the time allotted is increased to five minutes. In order to show the benefits of multiple agents, we ran the same simulation with ten agents. The results are depicted in Fig. 2.10 and Fig. 2.11(a), which show that there is an improvement in the percentage of time spent in the fire states compared to the one-agent case. These results help us determine the values of the design parameters T_{half}, α_{half} based on the exploitation-exploration ratio preferences. For example, if we have ten agents and we prefer them to stay in fire regions for 75% of time (in 3-minute window), we set $\alpha_{half} = T_{half} = 50$.

Time spent in fire states vs. time allotted: Fig. 2.11(b) shows the percentage of time spent in fire states as the time allotted to the agents increases from one to ten minutes for different number of agents. We have set $\alpha_{half} = T_{half} = 50$ based on our previous observation. We can note that the percentage increases both as the time allotted increases as well as the number of agents increases. However, we note a saturation behavior in both, implying that there is a minimum amount of time needed for exploration.

Model adaptation to random fire occurrences: To evaluate our algorithm's ability to adapt to dynamic environments, we evaluated our MARL framework as the fire position changes every three minutes in a random manner, as shown in Fig. 2.12. We can

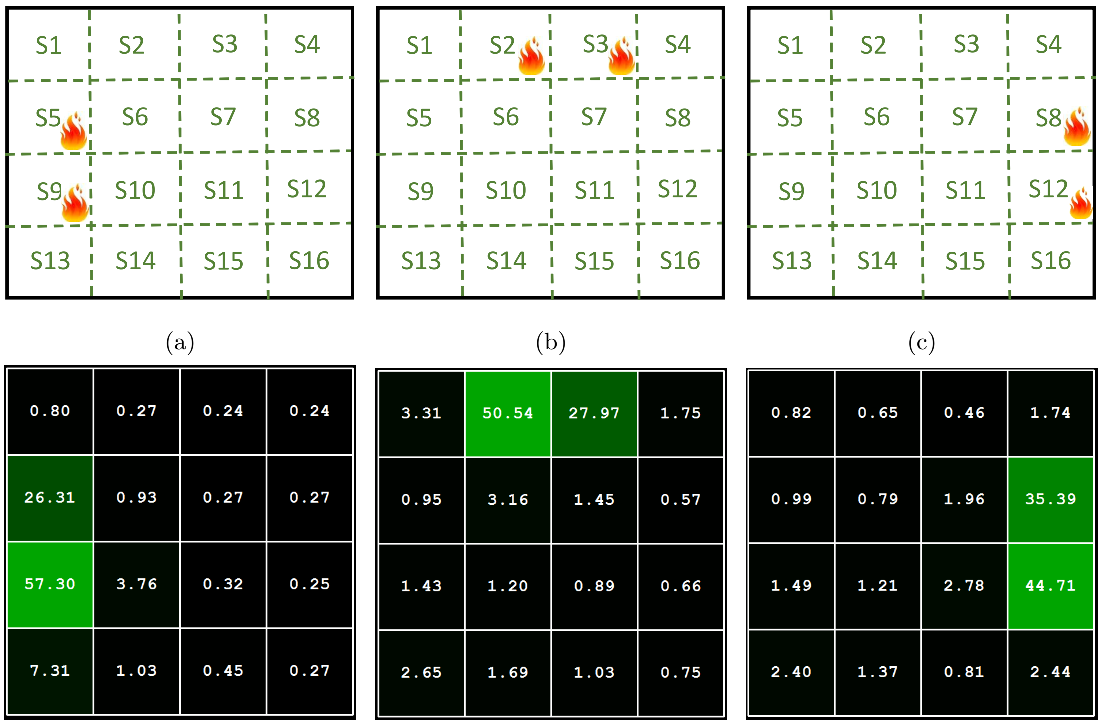


(a)



(b)

Figure 2.11: (a) Time spent in fire states [%] vs. learning rate half life (10 Agents); (b) Time spent in fire states [%] as the time allotted to the agents increases ($T, \alpha = 50$).



0.80	0.27	0.24	0.24
26.31	0.93	0.27	0.27
57.30	3.76	0.32	0.25
7.31	1.03	0.45	0.27

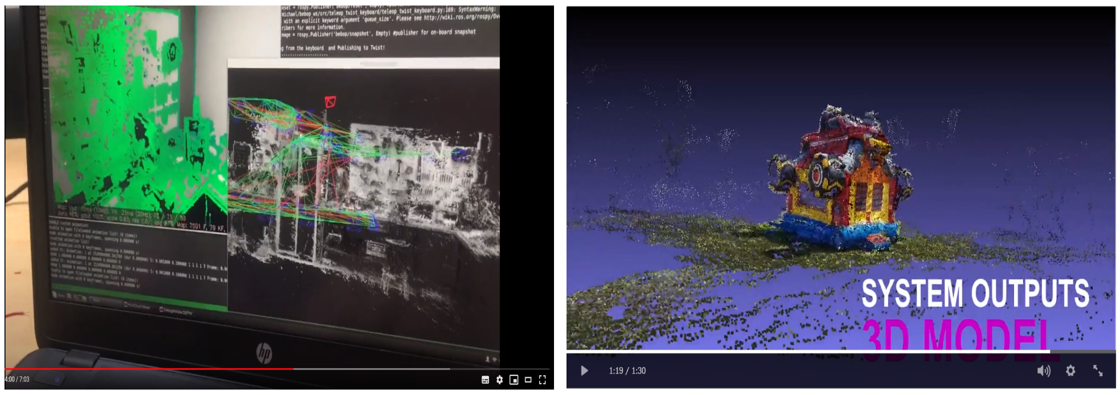
3.31	50.54	27.97	1.75
0.95	3.16	1.45	0.57
1.43	1.20	0.89	0.66
2.65	1.69	1.03	0.75

0.82	0.65	0.46	1.74
0.99	0.79	1.96	35.39
1.49	1.21	2.78	44.71
2.40	1.37	0.81	2.44

(a) (b) (c) (d) (e) (f)

Figure 2.12: Random fire occurrences and model adaptation.

see that the system is able to adapt to change in fire positions by adaptively changing the percentage of time spent in fire states. We have enabled periodic exploration with



(a) (b)

Figure 2.13: (a) 3D reconstruction in an indoor environment using single drone [2]; (b) 3D reconstruction in an outdoor environment using two drones [3].

resetting of parameters, i.e., Q-values to 0, $T_{half} = \alpha_{half} = 50$.

Video demos: A demo video (screenshot shown in Fig. 2.13(a)) showing dense 3D point cloud construction in real time using one drone in an indoor environment (controlled via a laptop) can be found in [2]. The same (screenshot shown in Fig. 2.13(b)) involving two autonomous drones in an outdoor environment implementing multi-agent reinforcement learning to accelerate the 3D reconstruction can be seen in [3]. Another informative video demo that is complimentary to the above ones—showing how 3D reconstruction can be performed using the MARL framework and the mobile application developed as part of this chapter—can be found at [40].

2.5 Summary

We presented Argus, a Multi-Agent Reinforcement Learning (MARL) framework (implemented as a mobile application plus a backend server) for cooperative data collection to build a 3D mapping of the disaster scene using agents present around the incident zone to facilitate the rescue operations.

The proposed framework leverages the first-hand information available at the incident zone so as to perform coordinated data collection using MARL. However, the proposed framework does not scale with the state and action spaces as the Q-table becomes larger and cannot hold Q-values for large number of state and action spaces. We will deal with this shortcoming in the next chapter.

Chapter 3

Aerial-DeepSearch: Distributed Multi-Agent Deep Reinforcement Learning for Search Missions

Disaster management discussed in previous chapter also involves searching for victims/objects using a group of drones. While, in the previous chapter, we have explored the possibility of using the bystanders and drones for coordinated disaster scene data collection using multi-agent reinforcement learning, in this chapter, we focus on using a group of drones to perform a coordinated search using multi-agent deep reinforcement learning that also addresses some of the weaknesses of the simple MARL framework presented in the previous chapter.

Search and Rescue (SAR) operations are very important in disaster response. SAR usually consists of the search for and provision of aid to people who are in distress or imminent danger. The *search* part can also be extended to generic targets that are of importance to the mission e.g., certain vehicles (see Fig. 3.1), people who pose threat to public safety, lost animals, or valuable equipment. The general field of SAR includes many specialty sub-fields, typically determined by the type of terrain the search is conducted over; these include SAR using ground or surface versus aerial vehicles in mountain, marine, urban, or battlefield environments.

3.1 Introduction

Motivation: The first main challenge in SAR missions is represented by the fact that the environment is typically unknown and needs to be explored. Existing formulations represent the environment as a 2-D grid map (also known as occupancy map), and use a frontier-based approach, where the frontier is defined as key points or cells on the grid that are on the boundary of the already explored area; the algorithms then

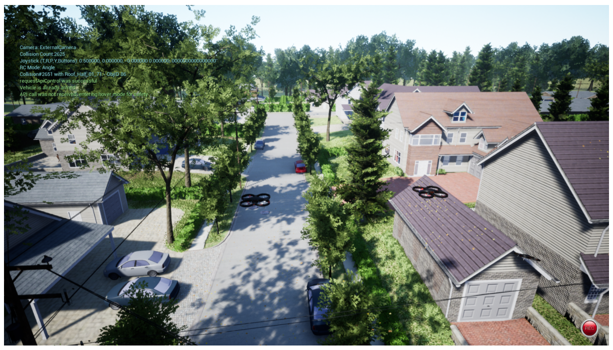


Figure 3.1: Illustrative example of a search for 'red sedan car' using two aerial drones in AirSim simulator [4].

determine a part of the frontier to explore next using utility or cost-based approaches. While in the former (utility-based approaches), importance is given to the expected new information gain, in the latter (cost-based approaches), energy expended and distance traveled are given more importance. The utility or cost functions are typically hand-crafted for specific environments using domain knowledge; for instance, the expected information gain is different in a marine environment with a small number of sparse obstacles, versus maze-like urban environments, and the movement cost depends on the terrain type or wind conditions. Such approaches can therefore show significant performance drops when presented with new, significantly different environments. To counter this problem, Learning-based and Deep-Reinforcement-Learning-(DRL) based approaches learn the utility or cost based on the environment (specific examples are discussed in the next section); however, they are mostly limited to single agents.

The second major challenge in SAR is the coordination of multiple agents (when available). Technically, this is due to the fact that the environment, from the point of view of any given agent, is non-stationary, due to the actions of other agents. Most existing multi-agent techniques (including those specializing to aerial multi-UAV SAR missions) focus on the offline planning of routes that obtain the best coverage in an apriori fully known environment; such plans are not generalizable to new environments.

Our Approach: In order to address the shortcomings of the existing SAR approaches mentioned earlier, we propose a novel Multi-Agent Deep Reinforcement Learning (MADRL) multi-UAV solution for autonomous exploration of unknown regions and target search. We consider a 2-D grid-based environment model, where each cell might arbitrarily contain a target, and we assume that each UAV has knowledge (either via direct observation, or communication) of the positions (cell coordinates) of every other UAV (full observability). The goal of our framework is to coordinate the agents to explore the environment while (i) avoiding overlaps between the UAVs' observations (which would correspond to wastage of resources); (ii) finding all the targets in the least amount of time; (iii) avoiding collisions. Our MADRL framework builds upon the actor critic architecture [41]. During training, a critic neural network that has access to the states and actions of all other UAVs is learned on each UAV. The critic on each UAV is used to provide feedback on the actions generated for the current state by the actor network on that UAV. The feedback is in terms of the long term reward (i.e., Q-value) obtained by the UAV for taking those actions in the current state. Using such feedback from critic, the parameters/weights of the actor network are updated in the direction suggested by the critic. The training is done in simulated environments, where each UAV broadcasts its actions to all other agents. During testing, when the UAVs are deployed for actual search missions, only the individual actors are used for generating the actions of each UAV, based on the current state information, i.e., the relative positions of all UAVs in a fully observable environment. Our framework also deals with partial observability and limited communication scenarios—where each UAV is able to observe the positions of other agents (or other agents may transmit their observations) only occasionally (instead of continuously) due to imperfect communication or restriction on the sensing range due to geographical constraints. For this, we propose to augment the actor and critic networks with recurrent neural networks such as Long Short Term Memory (LSTM) neural networks so that each agent can integrate its history/state information over time. Though we test our framework in a grid environment, our framework could be extended to non-grid or image-based environments where the state consists of the images observed by the UAVs, and the actions consist of continuous variables such as position and velocity. We will explore this in our future work (Sect. 3.5).

Main contributions can be summarized as follows:

- We propose Multi-Agent Distributed Actor Critic (MADAC) framework for multi-UAV coordination for aerial search missions of unknown regions.
- We realize the actor-critic framework on a distributed group of embodied agents (UAVs) instead of computing threads on a computer.
- We propose to handle imperfect communication and partial observability issues by augmenting our MADAC architecture with Recurrent Neural Networks (RNNs).
- We validate the proposed framework against other existing algorithms in a grid-world environment under both ideal and non-ideal communication settings.

Chapter Outline: In Sect. 3.2, we position our work with respect to existing techniques for SAR missions (both aerial and ground robots). In Sect. 3.3, we present our proposed approach, and in Sect. 3.4, we evaluate our model via simulations in a grid-world environment. Finally, in Sect. 3.5, we summarize the chapter.

3.2 Related Work

We now position our work with respect to the state of the art in multi-agent cooperation for search and rescue missions. We divide the existing work on multi-agent coordination according to ground/terrestrial and aerial SAR.

Terrestrial SAR: There are some traditional works where a ground robot searches the environment in Urban Search and Rescue (USAR) missions using frontier-based approaches. A frontier is defined as a key point on the boundary of an already explored area; hence further expanding the frontier will enable the robot to explore unseen regions. As mentioned before, these approaches can be either utility- or cost-based. For example, Niroui et al. [42] proposed a hierarchical Partially Observable Markov Decision Process (POMDP) for the ground robot to explore an USAR area. The authors proposed a utility-based approach that weighted the terrain type (open unvisited, unvisited climbable, unknown obstacle), neighbouring cell states and travel distance to determine an exploration frontier. Basilico et al. [43] propose a multi-criteria-based utility approach to determine the next frontier to explore based on the distance to the frontier, information gain and ability to communicate information back to a base station. Mei et al. [44] propose a cost-based approach to select frontier locations using the robot's orientation, such that repeated navigation of the same area is minimized. Most of these approaches are limited to a single agent, and also use parameters/weights tuned that are tune for specific environments, and they do not perform well in different environments.

Approaches based on Deep Reinforcement Learning (DRL) have been proposed to alleviate such problems and adapt to more general environments. They are also able to take high-dimensional state spaces such as an image as input. Tai et al. [45] propose a Deep Q Network (DQN) using Convolutional Neural Networks (CNNs) for exploring a corridor environment using depth information from an RGB-D sensor. Zhang et al. [46] designed an exploration strategy to learn representations of a global map from sensor data using the Simultaneous Localization and Mapping (SLAM) framework and LSTM neural networks. Niroui et al. [47] present a RL architecture based on Asynchronous Advantage Actor Critic (A3C) to determine an optimal exploration strategy (best frontier to explore next) using the map of the environment and the location of the robot and the frontier locations as image input. Unlike our multi-agent coordination approach, all these approaches are limited to a single agent.

Aerial SAR: Existing works that enable SAR missions using aerial robots [48] can be broadly categorized into traditional offline optimization (both single and multiple UAVs) and reinforcement learning based approaches (single UAV).

In terms of single UAV traditional optimization based approaches, Delmerico et al. [49] designed an active autonomous exploration strategy for an aerial robot to find the best path for ground robots in SAR missions. Specifically, the problem is modeled as an optimization problem that aims to minimize the number of visited waypoints for the aerial robot, and the traversal time of the ground robot. Sampedro et al. [50] designed a fully autonomous aerial robot for SAR applications in indoor environments using learning-based techniques. The proposed solution generates way points for the robot offline by applying a K-means clustering over points randomly distributed over the area to be explored. In terms of multi-UAV approaches, Yanmaz et al. [51] have designed a multi-UAV system for SAR missions. Here, a central base station generates sufficiently spaced preplanned waypoints for each UAV so as to sweep the whole area. Khan et al. [52] propose a multi-UAV cooperative search algorithm based on Multiple Traveling Salesman Problem (MTSP). The authors model the environment as a 2-D grid with search targets randomly spread across the grid. The proposed algorithm generates a set of waypoints (cells to traverse) for the UAVs so as to minimize the search time while considering sensing and communication limitations. Hu et al. [53] developed a probabilistic model for multi-UAV vision-based cooperative target search. A 2-D probability map of the environment is iteratively updated using measurement data and information shared among agents, and the control inputs/trajectories of the UAVs are obtained by maximizing the total coverage area/information gain.

In terms of reinforcement learning based approaches, Zuluaga et al. [54] proposed a Deep Q Network for autonomous SAR missions for a single UAV where the reward is based on the area of the observed image occupied by the target object. Sadhu et al. [55] present a Multi-Agent Reinforcement Learning (MARL) framework for coordinated exploration (data collection) of a disaster scene using bystanders/UAVs to build 3D maps of the incident scene. The proposed approach uses a centralized Q-table which is prone to single point of failure, and also cannot scale to large state spaces as it uses traditional table-based Q-learning.

In contrast to the above approaches, we propose a multi-UAV system based on actor-critic architecture for cooperative target search in unknown regions. In our approach, during training, a critic that considers the actions and states of all the UAVs is used to update the parameters of the actor network on each UAV. During testing/deployment, only the actor network is used to generate the actions based on the current state. Such an approach is not feasible with Multi-Agent Deep Q Networks (MADQN), as it cannot have different information during training and testing. Also, our approach is able to handle partial observability and imperfect communication scenarios and is environment agnostic (it can be readily deployed in any new environment that can be simulated) and leverages the capabilities of several UAVs to perform the search mission in the least amount of time without collision.

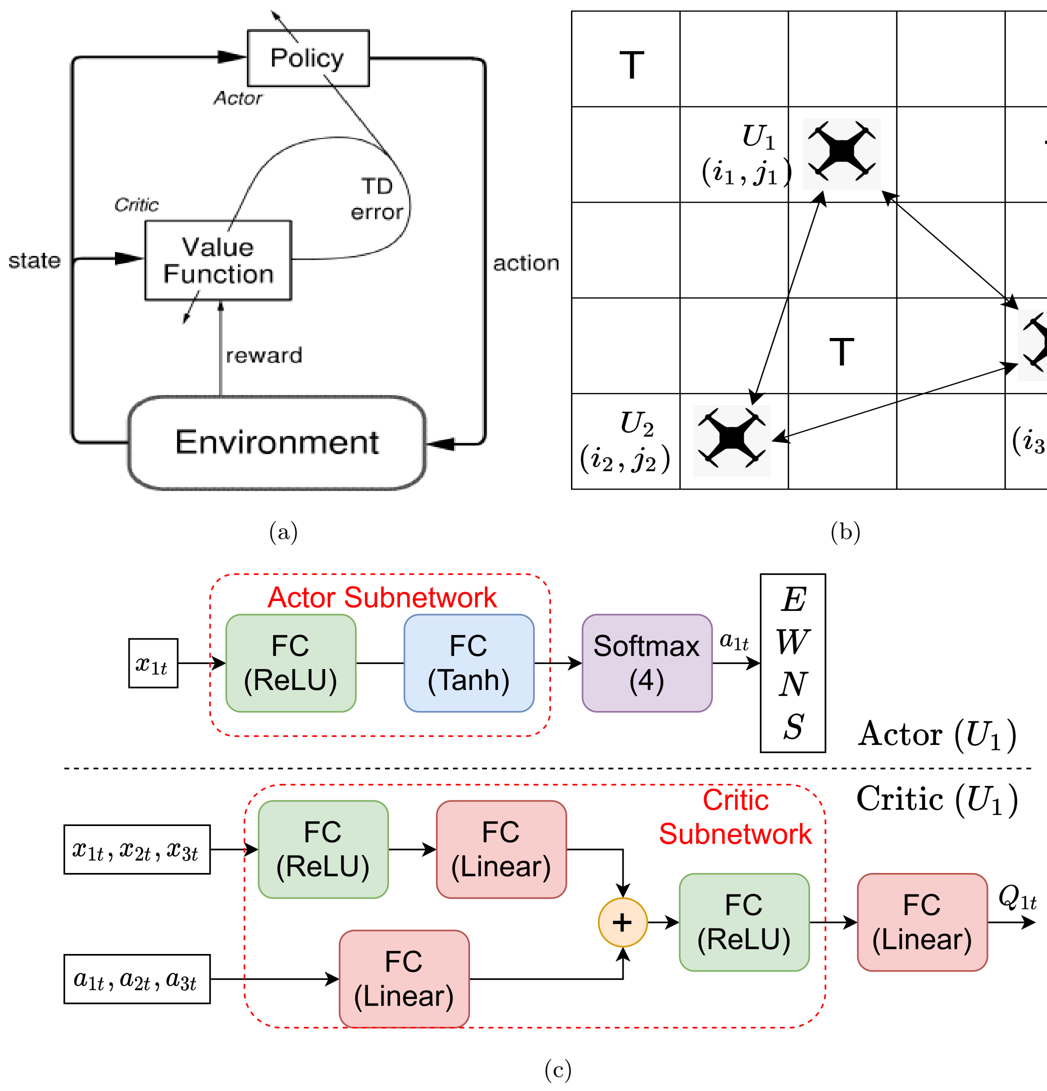


Figure 3.2: (a) Actor-Critic architecture [5], where the TD-error of critic is used to update both the critic and the actor; (b) Multi-agent Actor-Critic (MADAC) framework—with three UAVs and three targets in the grid environment—for illustration purpose; (c) (Top) Actor architecture of UAV 1; (Bottom) Critic architecture of UAV 1. Similar architectures can be assumed for other two UAVs.

3.3 Proposed Solution

We present a brief background about DRL approaches before presenting our solution.

Background on Reinforcement Learning: In the previous chapter, we have discussed about simple Q-learning. In this chapter, we go a step ahead and discuss about Deep Q-learning and advanced RL algorithm known as actor-critic. We consider a multi-agent extension of Markov Decision Processes (MDPs) called Partially Observable Markov Games (POMGs) [56, 57]. A Markov game for N agents is defined by a set of states $\mathcal{S}$ describing the possible configurations of all agents, a set of actions $\mathcal{A}_1, ..., \mathcal{A}_N$, and a set of observations $\mathcal{O}_1, ..., \mathcal{O}_N$ for each agent. To choose actions, each agent i uses a stochastic policy $\pi_{\theta_i} : \mathcal{O}_i \times \mathcal{A}_i \mapsto [0, 1]$, which produces the next state according to the state transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A}_1 \times ... \times \mathcal{A}_N \mapsto \mathcal{S}$ (here θ_i represents the parameters of the policy/actor network of agent i, which is sometimes omitted from the subscript of π for the sake of simplicity). Each agent i obtains rewards as a function of the state and its action $r_i : \mathcal{S} \times \mathcal{A}_i \mapsto \mathbb{R}$, and receives a private observation correlated with the state $\mathbf{o}_i : \mathcal{S} \mapsto \mathcal{O}_i$. The initial states are determined by a distribution $\rho : \mathcal{S} \mapsto [0, 1]$. Each agent i aims to maximize its own total expected return $\mathbb{E}[R_i = \sum_{t=0}^{T} \gamma^t r_i^t]$, where $\gamma \in [0, 1]$ is a discount factor, and T is the time horizon.

Q-Learning and Deep Q-Networks (DQN): Q-Learning and DQN [58] are popular methods in reinforcement learning and have been previously applied to multi-agent settings [59]. Q-Learning makes use of an action-value function for policy π defined as $Q^\pi(s, a) = \mathbb{E}[R|s^t = s, a^t = a]$, which denotes the expected long term return the agent receives by taking action a in state s and then following the policy π thereafter. This Q function can be recursively rewritten using Bellman expectation equation as $Q^\pi(s, a) = \mathbb{E}_{s'}[r(s, a) + \gamma \mathbb{E}_{a' \sim \pi}[Q^\pi(s', a')]]$, where $r(s, a)$ denotes the immediate reward received by the agent and s', the next state. Here, the distribution of the next state, s' depends on the current action taken, a. DQN is an off-policy algorithm that learns the optimal action-value function Q^* corresponding to the optimal policy using bootstrapping/Temporal-Difference (TD) learning [5], by minimizing the loss, $\mathcal{L}(\theta) = \mathbb{E}_{s,a,r,s'}[(Q^*(s, a|\theta) - y)^2]$, where, $y = r(s, a) + \gamma \max_{a'} \bar{Q}^*(s', a')$ and $\bar{Q}$ is a target Q function whose parameters are periodically updated with the most recent θ, to help stabilize learning. Another crucial component of stabilizing DQN training is the use of an Experience Replay (ER) buffer $\mathcal{D}$ containing tuples (s, a, r, s') from which samples are drawn randomly to train the DQN. Q-Learning can be directly applied to multi-agent settings by having each agent i learn an independently optimal function

Q_i [60]. However, because agents are independently updating their policies as learning progresses, the environment appears non-stationary from the point of view of any one agent, violating one of the assumptions on the Markov process that is required for convergence. Moreover, the ER buffer cannot be used in such a setting as state transition probabilities, $P(s'|s, a, \pi_1, ..., \pi_N)$ change when any agent's policy, π_i changes.

Policy Gradient (PG) Algorithms: Policy gradient methods are another popular choice for a variety of RL tasks, from continuous control [61] to Atari game-playing [62]. The main idea is to directly adjust the parameters θ of the policy in order to maximize the objective $J(\theta) = \mathbb{E}_{s\sim p^\pi, a\sim\pi_\theta}[R]$ (where p^π, the state visitation (occupancy measure) distribution, is the discounted sum of probabilities of visiting a given state when the policy, π_θ, is followed indefinitely) by taking steps in the direction of $\nabla_\theta J(\theta)$. Using the Q function defined above, the gradient of the policy can be written as [41],

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim p^\pi, a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q^\pi(s, a)], \tag{3.1}$$

The policy gradient theorem has given rise to several practical algorithms, which differ in how they estimate Q^π. For example, one could learn an approximation of the true action-value function $Q^\pi(s, a)$ by e.g. TD learning described above; this $Q^\pi(s, a)$ is called the *critic* and leads to a variety of *actor-critic* [5] algorithms (see Fig. 3.2(a)). In these algorithms, as can be seen from (3.1), the actor's parameters, θ are updated with the help of the critic, Q^π. Policy gradient methods are known to exhibit high variance gradient estimates as the trajectories (state-action pairs) generated by the policy can be widely different. This is exacerbated in multi-agent settings, since an agent's reward usually depends on the actions of many agents, the reward conditioned only on the agent's own actions (i.e., when the actions of other agents are not considered in the agent's optimization process) exhibits much more variability, thereby increasing the variance of its gradients. Also, the use of baselines, such as value function baselines typically used to reduce high variance in policy gradient methods, is not effective in multi-agent settings due to the non-stationarity issues mentioned above.

Deterministic Policy Gradient (DPG) Algorithms: It is also possible to extend the policy gradient framework to deterministic policies [63] $\mu_\theta : \mathcal{S} \mapsto \mathcal{A}$. In particular, under certain conditions, we can write the gradient of the objective $J(\theta) = \mathbb{E}_{s\sim p^{\mu}}[R(s, \mu(s))]$ as,

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{s\sim\mathcal{D}}[\nabla_{\theta}\mu_{\theta}(s)\nabla_{a} Q^{\mu}(s,a)|_{a=\mu_{\theta}(s)}] \tag{3.2}$$

Deep Deterministic Policy Gradient (DDPG) [61] is a variant of DPG where the policy μ and critic Q^{μ} are approximated with deep neural networks. DDPG is also an off-policy algorithm, and samples trajectories from a replay buffer of experiences (that are stored throughout training) to train the actor and critic networks. DDPG also makes use of a target network, as in DQN [58]. The Q function is calculated by minimizing the loss $\mathcal{L}(\theta) = \mathbb{E}_{s,a,r,s'}[(Q^{\mu_{\theta}}(s,a) - y)^2]$, where $y = r(s,a) + \gamma Q^{\mu_{\theta'}}(s', \mu_{\theta'}(s'))$. The parameters θ' of the target Q function $Q^{\mu_{\theta'}}(s', a')$ are frozen for multiple time steps and updated with the parameters θ periodically to ensure stability of convergence. Both standard policy gradient methods and DQN are not particularly suited to multi-agent settings, as they do not directly consider the policies of other agents (non-stationarity issue mentioned above). A better approach is one that considers the policies of other agents in its Q function during training, but only uses agents' local policy π during execution [59].

Distributed Actor-Critic for UAV Search Missions We extend DDPG [57,61] using the idea presented in [59] for multi-UAV search missions, where different actors are realized as UAVs instead of as computer threads. Unlike DQN, where a single agent is represented by a single neural network, actor-critic methods utilize multiple versions of the above to learn more efficiently. In actor-critic methods there are multiple workers/actors with their own set of network parameters. Each of these agents interacts with the environment in parallel. The reason this works better than having a single agent (beyond the speedup of getting more work done) is that the experience of each agent is independent. This way the overall experience available for training becomes more diverse. Generally the different actors are realized as threads in a computer system to speed up the training process. We call our framework Multi-Agent Distributed Actor Critic (MADAC), where we realize actor-critic on a distributed system with communication and observation uncertainties. In our framework, where actors are realized as real systems such as UAVs exhibits the following characteristics: (i) able to learn quickly by virtue of parallel training and exploration of the environment; (ii) able to converge sooner and exploit the learned information in real time (see Sect. 3.4); (iii) deal with imperfect communication and partial observability challenges.

We consider N UAVs, $U_1, ...U_N$, operating in an $M \times P$ grid space (see Fig. 3.2(b)), cooperatively executing the aerial search mission where the targets are located in random cells. The state of UAV i, at time t, x_{it} is represented as the concatenation of the agent's own cell coordinates, (i_i, j_i) and relative cell coordinates of other agents as observed from U_i. Generally the neural network's output depends on the order of concatenation (i.e., output changes when the order changes), but it is possible to reduce this dependence by training the network with all possible orders (which we will consider in future work). Denote the policies of N UAVs with $\pi = \{\pi_1, ..., \pi_N\}$ parameterized by $\boldsymbol{\theta} = \{\theta_1, ..., \theta_N\}$. Actor/policy network of UAV i, π_i, shown in Fig. 3.2(c) (top), takes the state, x_{it} as input and gives the action probabilities over the four actions (moving East, West, North, South from the current cell position). We can write the gradient, $\nabla_{\theta_i} J(\theta_i)$, of the expected return for UAV i, $J(\theta_i) = \mathbb{E}[R_i]$ as,

$$\mathbb{E}_{s\sim p^{\mu}, a_i \sim \pi_i}[\nabla_{\theta_i} \log \pi_i(a_i|o_i) Q_i^{\pi}(\mathbf{x}, a_1, \dots, a_N)]. \tag{3.3}$$

where $Q_i^{\pi}(\mathbf{x}, a_1, \dots, a_N)$ is critic for U_i—shown in Fig. 3.2(c) (bottom) for $i = 1, N = 3$. It takes as input the states of all UAVs, $[x_{1t}, ...x_{Nt}]$, actions of all UAVs, $[a_1, \dots, a_N]$ (obtained from their respective actor networks), and outputs the Q-value for agent i. In order to simplify the problem, we will consider deterministic policies, μ_{θ_i} w.r.t. parameters θ_i (abbreviated as μ_i). The policy gradient, $\nabla_{\theta_i} J(\mu_i)$, can then be written as [63],

$$\mathbb{E}_{\mathbf{x}, a\sim\mathcal{D}}[\nabla_{\theta_i} \mu_i(a_i|o_i) \nabla_{a_i} Q_i^{\mu}(\mathbf{x}, a_1, ..., a_N)|_{a_i=\mu_i(o_i)}], \tag{3.4}$$

where the experience replay buffer $\mathcal{D}$ contains the tuples $(\mathbf{x}, \mathbf{x}', a_1, \dots, a_N, r_1, \dots, r_N)$, recording experiences of all UAVs. The critic for UAV i, Q_i^{μ} is updated as:

$$\begin{aligned}\mathcal{L}(\theta_i) &= \mathbb{E}_{\mathbf{x}, a, r, \mathbf{x}'}[(Q_i^{\mu}(\mathbf{x}, a_1, \dots, a_N) - y)^2], \\ y &= r_i(o_i, a_i) + \gamma\, Q_i^{\mu'}(\mathbf{x}', a_1', \dots, a_N')|_{a_j'=\mu_j'(o_j)}\end{aligned} \tag{3.5}$$

where $\mu' = \{\mu_{\theta_1'}, ..., \mu_{\theta_N'}\}$ is the set of target policies with delayed parameters θ_i' and r_i is the reward given to UAV i. The reward r_i is calculated as the weighted combination of four components, $r_i = w_d r_{di} + w_e r_{ei} + w_c r_{ci} + w_t r_{ti}$, as follows: (i) detection reward, r_{di}: if the UAV detects the target in its current cell, a positive reward is assigned; (ii) exploration reward, r_{ei}: if the UAV visits an unexplored cell, it gets a positive reward; (iii) collision reward, r_{ci}: if the action of the current UAV causes a collision (i.e., moves to a cell where there is another UAV already positioned), it receives a negative reward (this can also be defined in terms of inter-UAV distance/spatial closeness among UAVs between two time steps); (iv) time penalty, r_{ti}: a negative reward for each time step where (i) and (ii) are not awarded to the agent. The weights for these rewards are determined based on the application scenario. If there is some prior information about the locations of the targets, then w_d, w_e are assigned a high value during training, as the learned information can be directly applicable to the testing scenario. In case, the environment is completely unknown, w_e should be given high value, while w_d should be given a low value as the there may not be any correlation between the locations of the targets during training and deployment. As the critic on each UAV is considering the actions taken by all other UAVs, the environment is stationary even as the policies of the agents are changing since $P(s'|s, a_1, ..., a_N, \pi_1, ..., \pi_N) = P(s'|s, a_1, ..., a_N) = P(s'|s, a_1, ..., a_N, \pi'_1, ..., \pi'_N)$ for any $\pi_i \neq \pi'_i$. This is not the case if we do not explicitly condition on the actions of other UAVs. From (3.5), we can note that, each UAV requires the observations and policies of other UAVs to perform an update to its critic. This may not be a restrictive assumption especially when the training is performed in simulated environments. Details on the training procedure are given in Sect. 3.4.

Actor and Critic Networks: Fig. 3.2(c) (Top) shows our actor architecture consisting mainly of fully connected layers as the input (UAV's state) is a 1-D vector. We consider *tanh* and leaky Rectified Linear Unit *(ReLU)* as our activation functions because of their well-known advantages [64]. After the fully connected layers, we have a softmax layer to generate probabilities for different actions on the grid—East (E), West (W), North (N), South (S). Fig. 3.2(c) (Bottom) shows our critic architecture which takes the states and actions of all UAVs as input. As the state vectors have more elements than the action vectors, we first pass the state vectors through fully connected layers with leaky ReLU activation functions. These outputs are then passed through fully connected layers

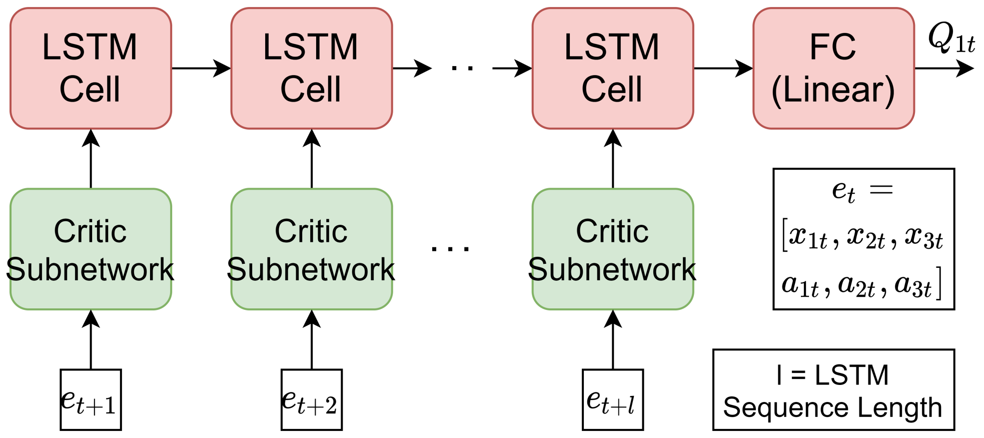


Figure 3.3: Critic augmented with LSTM layer to handle imperfect communication and partial observability scenarios. Actor has similar architecture.

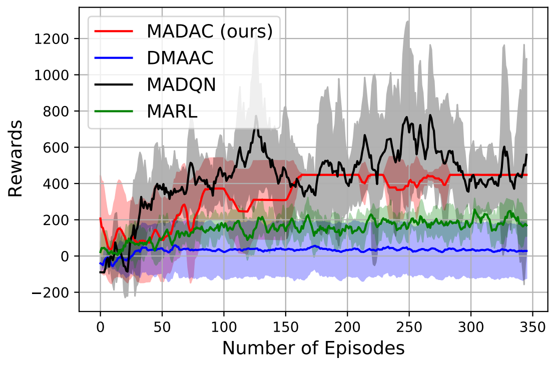
without any activation function (i.e., linear operation of weight multiplication and bias addition) to be added with similar outputs from the action vectors. The sum is then passed through leaky ReLU activation function followed by fully connected layers with single output and no (linear) activation function to obtain a single Q value. The actor and critic networks are updated using (3.4) and (3.5) respectively, using experiences sampled from ER buffer.

Imperfect Communication and Partial Observability: Full or partial observability is defined in terms of being able to observe the state information of other UAVs. In a fully observable setting, each UAV is able to observe the state information of all other UAVs, whereas in a partially-observable scenario, states of only some (probably nearby) UAVs is observable. On the other hand, imperfect communication affects the availability of action information of other UAVs as such information is only available when other UAVs transmit it. To study such scenarios, we define a parameter, p, to indicate the link failure probability between any two agents. This means that $p = 0$ corresponds to perfect communication scenario where as for example, $p = 0.1$ indicates that 10% of the communication packets (containing the action information) corresponding to a link between two UAVs are dropped. At any time step, if the communication packet containing the action information of the transmitting UAV is dropped, the last known action of that UAV is used by the receiving UAV. We now describe our enhanced architecture to improve performance in situations with imperfect communication and partial observability. Fig. 3.3 shows our proposed critic architecture, where we have augmented the critic architecture in Fig. 3.2(c) (Bottom) with Long Short Term Memory (LSTM) units. Actor network can also be modified in a similar manner. The critic network now processes experience (state and action) information from the previous timesteps to learn the underlying temporal patterns in them. The LSTM units process such temporal information (from previous sequence of experiences) to better estimate the underlying state, than just using the current experience. In Fig. 3.3, l denotes the LSTM sequence length. Using a high value for l translates to better performance but higher computational complexity and vice-versa. Due to the computational complexity introduced due to LSTMs, it may be beneficial to use them (depending on the available resources) only for $p \neq 0$ scenarios (see Sect. 3.4).

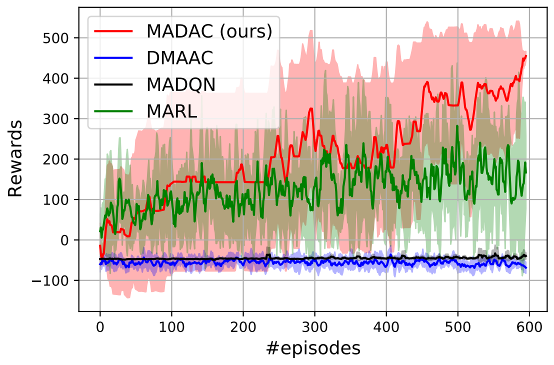
3.4 Performance Evaluation

In this section, we first describe our simulation, training, and testing setup and then evaluate the performance of our solution under two scenarios: (i) ideal communication, where each UAV has access to the state and action information of all the other agents at each time step; (ii) non-ideal communication where, depending on the link failure probability we define below, UAVs sometimes cannot receive the state/action information from other UAVs and need to rely on the last observed values.

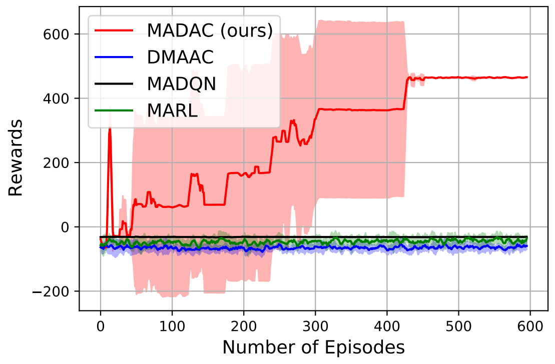
Comparison Plan: In our MADAC algorithm, we consider a critic on each UAV that has access to the state and action information of all other UAVs during training. During testing, each UAV has access only to the states of all other UAVs as we assume full observability. We compare our approach with three other algorithms: (i) Multi-Agent Deep Q Network (MADQN), where each UAV observes/receives the states (but not actions) of all other UAVs and independently updates its Deep Q Network based on its own experiences; the Q-Network does not receive the actions of other agents during training, as it must do so during testing also, which is prohibitive; (ii) Multi-Agent Q-Learning (which we simply call Multi-Agent Reinforcement Learning (MARL) to be



(a)



(b)

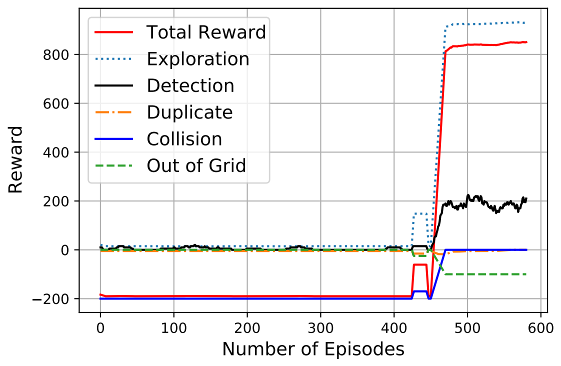


(c)

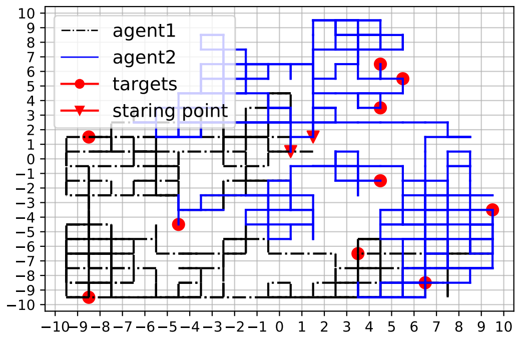
Figure 3.4: Average reward of the agents vs. number of training episodes when number of agents = (a) 2; (b) 4; (c) 6. (grid size=100 × 100; number of targets=100)

consistent with previous literature [55]), which is a simplification of the MADQN, where each agent maintains a Q-table instead of Deep Q-Network; (iii) Multi-Agent Distributed Actor Critic with Decentralized critic (DMAAC). This is an extension of MADQN where each agent instead of maintaining a single Q-network, maintains both an actor and critic network. However, the critic network is trained in a similar manner to DQN i.e., does not receive action information from other agents and hence is desynchronized from the critics of others.

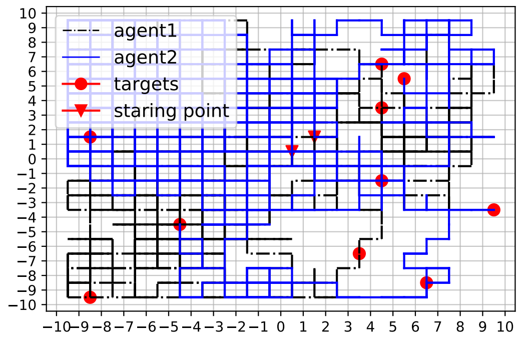
Training and Inference Setup: We used a grid-world like environment defined in the OpenAI Gym environment [65] to implement our MADAC architecture in Tensorflow (Python). As in Fig. 3.2(c) (Top), we use a simple two-layer fully connected neural



(a)



(b)



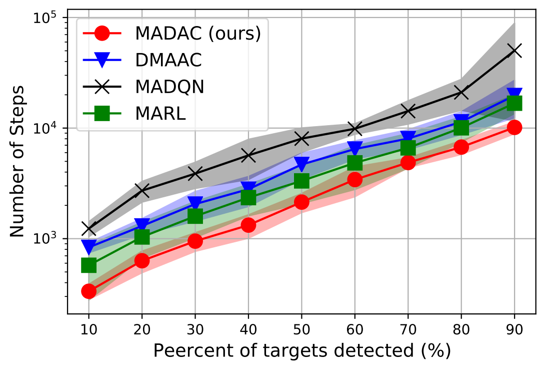
(c)

Figure 3.5: (a) Total reward split across different components as training progresses (100 × 100 gird with 100 targets and 4 agents); Trajectory followed by the UAVs when the location of targets is (b) same between training and testing; (c) different between training and testing. The targets shown are all the targets in the environment. We can notice that there are more steps in (c) than in (b). A video demo of these trajectories over time can be seen at [6].

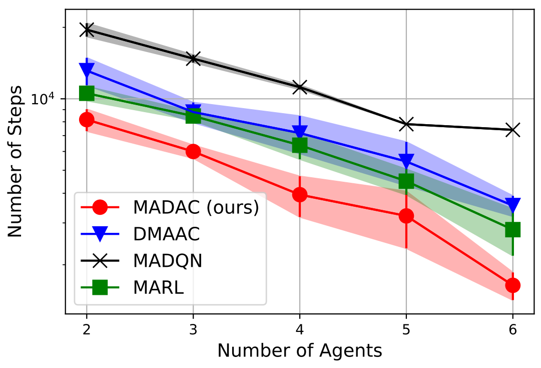
network as actor with 50 units each. For the critic (Fig. 3.2(c) (Bottom)), a two-layer fully-connected neural network with 100 and 50 units is used to process the state and a one-layer network with 50 units is used to process the action. The outputs of both are added and fed to a one-layer network with 50-units followed by a one layer network with one output unit (final Q-value). We used a learning rate of 0.001 for training the actor and critic networks. Training consists of running several episodes and updating the actor and critic networks based on the past experiences. In each episode, first we start with action generated by the actor network (we also add Ornstein-Uhlenbeck noise [61] to the output of the actor network to encourage exploration before selecting the final action). The action is executed in the environment to collect the experience at each timestep in the form of $\langle \texttt{state}, \texttt{action}, \texttt{reward}, \texttt{next\_state}, \texttt{fail} \rangle$. Each UAV stores such experience tuples to its own Experience Replay (ER) buffer. Here state, action, next-state consists of states, actions and next-states of all the UAVs, while reward is specific to the agent; `fail` indicates when the UAV either collides with another UAV, or goes out of the boundaries of the grid. If the ER buffer is full, a random entry is removed and filled with the new experience (random replacement is better so as to not be biased towards older or newer experiences). Also, at each time step, the agent randomly samples a *batch-size* number (we used 32) of experiences from this ER buffer to train first the critic network followed by the actor network (as actor is updated based on critic according to (3.4)). Using ER buffer is known to stabilize convergence and also learn from a variety of experiences instead of the immediate experiences produced by the current policy. Once the training is finished, we use (i.e., perform inference with) the trained models to evaluate the performance of our approach as the number of targets, UAVs and cell size is varied. As the agents might collide or go out of grid during inference (in which case, the episode ends), we average the results over several episodes and plot the corresponding confidence intervals with the mean.

Ideal Communication Scenario: In this subsection, we consider the ideal communication scenario where all the communication links among the agents are working perfectly (i.e., $p = 0$).

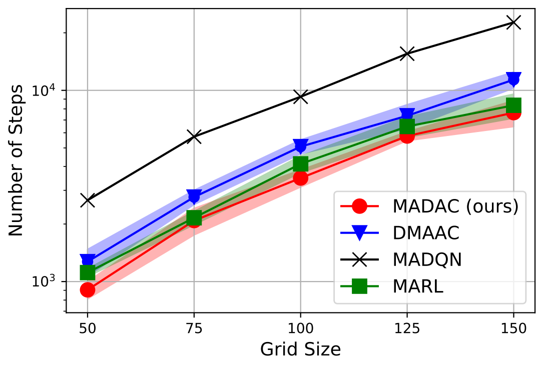
Episode reward vs. No. of training episodes: To evaluate how different models learn as the training progresses, we have plotted the average reward received by the agents versus the training episodes for different number of agents in Fig. 3.4. The simulation is conducted in a 100×100 grid world with 100 targets randomly deployed. When the number of agents is small (2 in Fig. 3.4(a)), it is clear that both MADQN and MADAC converge to a similar level of reward after certain number of episodes. However, as the number of agents increase (4 in Fig. 3.4(b), 6 in Fig. 3.4(c)), we can find



(a)



(b)



(c)

Figure 3.6: (a) Number of steps vs. number of agents (grid size=100 $\times$ 100; number of targets=100); (b) Number of steps vs. number of targets (grid size=100 $\times$ 100; number of agents=4); (c) Number of steps vs. grid size (number of agents=4; number of targets=100).

MADQN, MARL and DMAAC have difficulty converging owing to the non-stationarity issue mentioned in Sect. 3.3. From these figures, we can observe that MADAC has better performance on multi-agent tasks, especially when the number of agents is high.

Analyzing the learning process and the learned model: Fig. 3.5(a) shows the different components of total reward (100 $\times$ 100 grid with 100 targets and 4 agents) viz., exploration (+5), collision (-100), detection (+5), going out of grid (-100) and duplicate exploration (-5) (i.e., revisiting already explored cells), as the training progresses. We can notice that for several episodes, the positive rewards stay close to zero, while negative rewards stay negative. This means that agents are mostly colliding or going out of the grid. The cause of high collisions in the beginning is because the starting positions of the agents are very close and they can easily collide with another one even in one single (wrong) step. However, after about 400 episodes, we can notice that the agents experienced the exploration reward, which encourages them to subsequently explore more often. At the same time (as the agents are exploring new regions), we can notice that the collision and detection rewards show improvement. Fig. 3.5(b) and Fig. 3.5(c) show a sample trajectory followed by two agents in a 10×10 grid with 10 targets when the location of the targets is fixed (exploration reward is $+1$, detection reward is $+10$) and varying (exploration reward is $+5$, detection reward is $+5$) between training and testing sessions, respectively. From the trajectories we can notice that, in the former, agents do less exploration and more detection, and vice-versa in the latter.

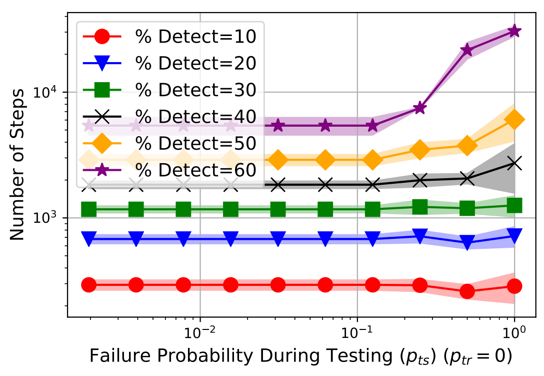
Time taken vs. Percentage of target detection: While testing the performance of models, it is important to observe how much time is needed to detect a certain percentage of targets We considered four UAVs and 100 targets randomly deployed in a 100×100 grid to study this behavior. Fig. 3.6(a) shows this result, where, we can notice that MADAC is the fastest one to detect a certain percentage of targets. Even though the curves for different algorithms seem close, the y-axis is in log-scale, which translates to considerable difference. Moreover, it is to be noted that—even though from Figs. 3.4(c), 3.4(b), DMAAC, MADQN and MARL seem to have difficulty in learning compared to MADAC—random actions can also result in detection of targets due to the nature of the problem. From Fig. 3.6(a), we can also notice an exponential relationship (linear in logarithmic scale) between steps required vs. detection percentage. In particular, we can note that, MADAC requires about 10000 steps to achieve 90% detection.

Time taken vs. No. of UAVs: To verify the assumption that more UAVs can accelerate the search mission, we evaluate the time taken to detect 60% of targets against the number of UAVs for a fixed grid size of 100×100 with 100 targets randomly placed in it. As shown in Fig. 3.6(b), when the number of UAVs increase, we can notice that the number of steps needed to search reduces, as UAVs can simultaneously search different regions when trained properly. We can again notice that MADAC achieves the best performance as it takes the least number of steps to complete the search.

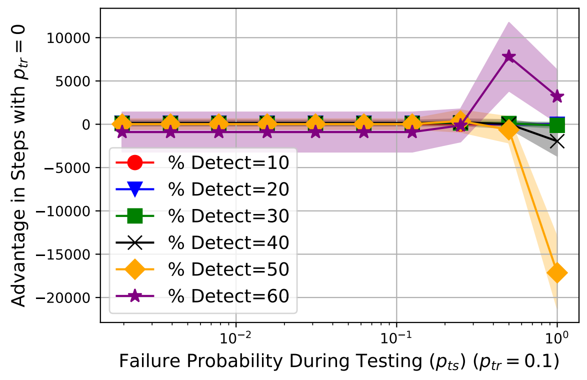
Time taken for search vs. Cell size of the grid: In a common SAR task, the area to search significantly influences the detection performance of the UAVs. Also, the search area is very large compared to the field of view of the UAV (considering near ground flight required for small object detection e.g., humans or animals). Hence, it is better to consider large grid sizes such as from 50×50 to 150×150. We compared the detection performance (in terms of number of steps required by four UAVs to detect 60% of targets out of 100 randomly deployed in a 100×100 grid scenario) vs. different grid sizes in Fig. 3.6(c). As the size of the grid increases, the density of targets decreases forcing the UAVs to take more number of steps. It is also clear that MADAC outperforms others.

Non-ideal Communication Scenario: In this subsection, we evaluate the performance of our MADAC model under non-ideal communication and partial observability settings (in both training and testing phases). Both imperfect communication and partial observability is achieved by setting $p \neq 0$. This is because, the communication packet sent by one UAV (A) to other UAV (B) contains both state and action information. When this packet is dropped due to $p \neq 0$, B has no access to A's action and is also not able to observe its state. We consider two types of p—p_{tr} (p used to train the models), p_{ts} (p used during testing). Given this, we study three aspects: (i) when trained with $p_{tr} = 0$ (i.e., assuming perfect communication), how the performance degrades during testing when $p_{ts} \neq 0$; (ii) train the models assuming $p_{tr} \neq 0$, to study if that imparts some robustness during testing when p_{ts} is non-zero; (iii) study the benefits of LSTM-enhanced models for both, $p_{tr} \neq 0$ and $p_{ts} \neq 0$.

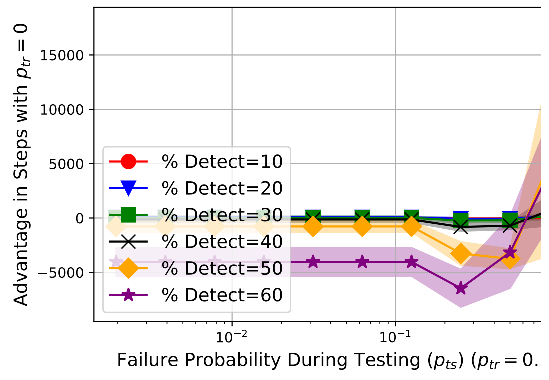
Ideal Training and Non-Ideal Testing: In real search missions, communications among UAVs are not reliable (e.g., in disaster scenes). Therefore, we test how our MADAC model—that was trained under ideal communication scenarios (i.e., $p = 0$)—performs when deployed in non-ideal communication scenarios (i.e., $p \neq 0$). In this scenario, $p_{tr} = 0$ but p_{ts} is varied from 0.002 till 1 by doubling every step in 10 steps. Fig. 3.7(a) shows this scenario for 100×100 grid, 100 targets and 4 UAVs. We can notice that as



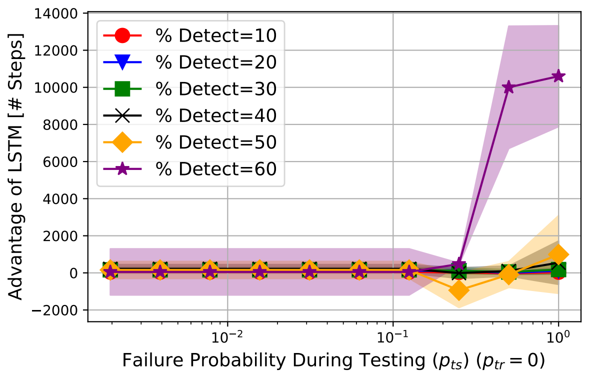
(a)



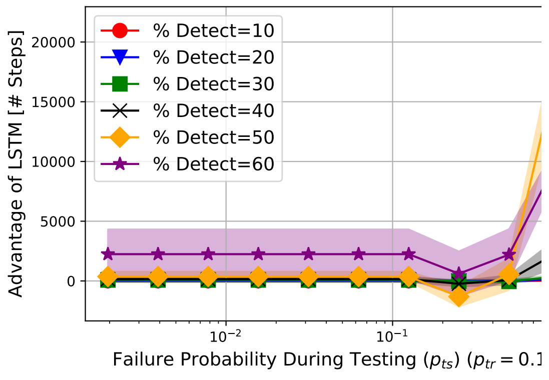
(b)



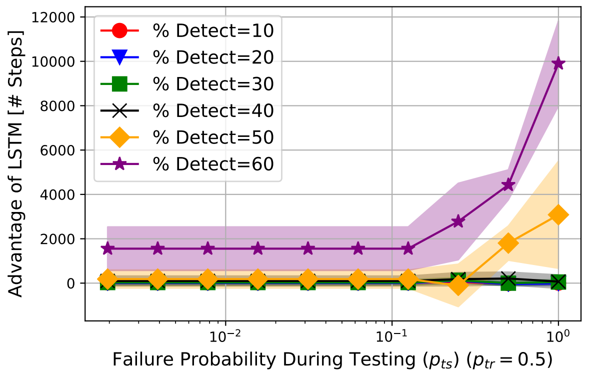
(c)



(d)



(e)



(f)

Figure 3.7: (a) Number of steps vs. p_{ts} for different detection percentages when $p_{tr} = 0$; Advantage in number of steps with respect to $p_{tr} = 0$ when (b) $p_{tr} = 0.1$; (c) $p_{tr} = 0.5$. Advantage of LSTM in number of steps with respect of not using LSTM for different values of p_{ts} and detection percentages, when (d) $p_{tr} = 0$; (e) $p_{tr} = 0.1$; (f) $p_{tr} = 0.5$; (number of agents=4; grid size=100 $\times$ 100, number of targets=100).

p_{ts} increases, more number of steps are needed to achieve a given detection percentage.

Non-ideal Training and Non-ideal Testing: We would like to study whether training the models with $p_{tr} \neq 0$ imparts any robustness to handle $p_{tr} \neq 0$. Fig. 3.7(b) considers this scenario, where we plot the advantage in steps (i.e., less number of steps) with respect to $p_{tr} = 0$ (i.e., the number of steps are relative to Fig. 3.7(a)), when we train the models with $p_{tr} = 0.1$. Fig. 3.7(b) plots similar metric for $p_{tr} = 0.5$. In the following discussion, we will categorize p into three levels: low ($p < 0.1$), medium ($0.1 \leq p \leq 0.5$), high ($p > 0.5$). From these two figures, we can notice that: (i) for low p_{ts}, it is better to train with low p_{tr} ($= 0$); (ii) for medium p_{ts}, it is better to train with medium p_{tr} ($= 0.1$); (iii) for high p_{ts}, it is better to train with high p_{tr} ($= 0.5$). This shows that there is a correspondence between p_{tr} and p_{ts}. Hence it is advantageous to store models trained with various p_{tr} values onboard (if possible) and choose the best model based on real-time p_{ts} estimated during deployment using the communication link.

Advantage of LSTM: We now study the benefit of including LSTM layer in our actor and critic networks for different p_{tr} and p_{ts} values. For this we considered $l = 5$ (LSTM sequence length) and 32 as the LSTM unit size. We store sequential experience data corresponding to 20 episodes. Whenever a new episode is generated, the data corresponding to one of the episodes is randomly replaced with the new episode data. Each entry in the training batch is sampled as follows: we randomly select an episode out of the 20 available ones; we then randomly sample $l = 5$ sequential experiences from this episode. We have used Tensorflow's *tf.dynamic_ rnn* [66] to deal with scenarios where the episode length is less than l. Fig. 3.7(d) plots the advantage in steps (i.e., less number of steps) compared to without LSTM (i.e., Fig. 3.7(a)) for $p_{tr} = 0$ scenario. Similarly, Figs. 3.7(e), 3.7(f) plot the advantage in steps for using the LSTM models with respect to not using LSTM, for $p_{tr} = 0.1$ and $p_{tr} = 0.5$ respectively. From these figures, we can notice that, adding LSTM does not provide significant benefits for low p_{ts}, but is definitely advantageous for medium and high p_{ts} values. Similarly, we can notice from Fig. 3.8 that the advantage of using LSTM is more evident with high p_{tr} values—we can observe that the blue curve corresponding to LSTM models trained with $p_{tr} = 0.5$ gives the best performance of all.

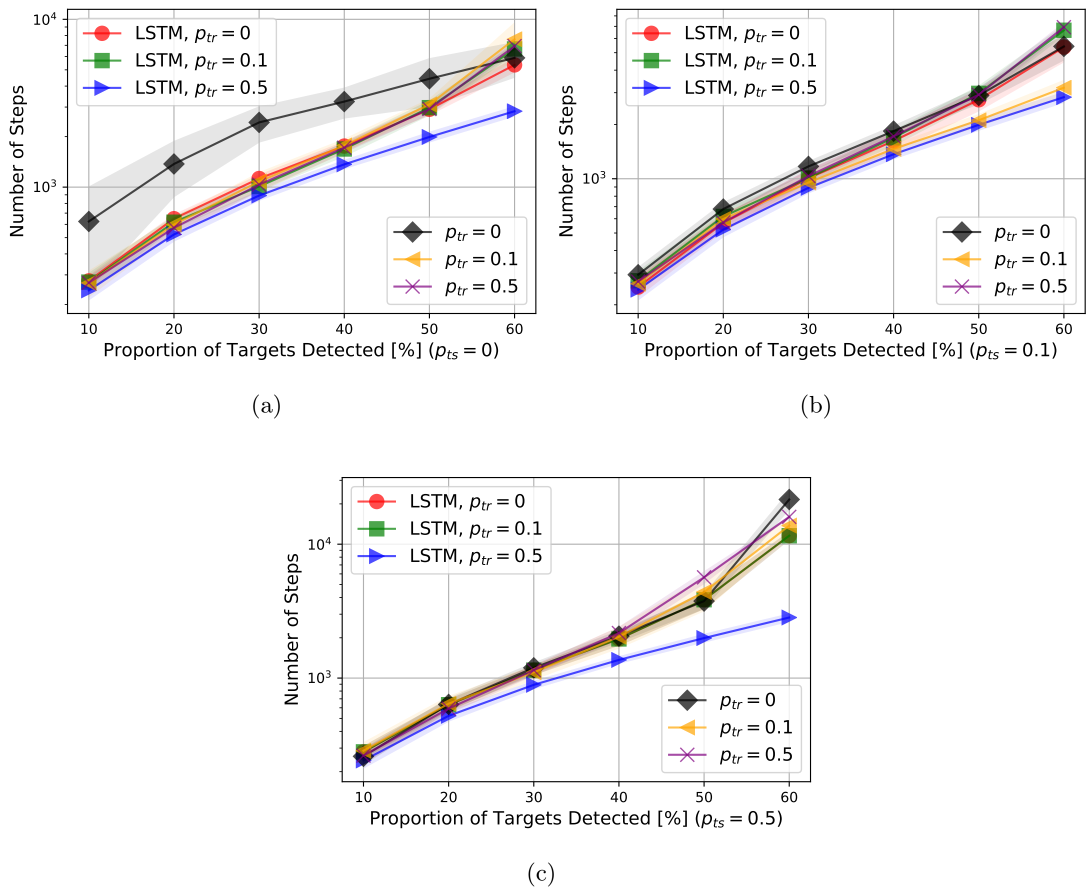


Figure 3.8: Comparison of performance in terms of number of steps vs. detection percentage across with- and without-LSTM models and different p_{tr} values when (a) $p_{ts} = 0$; (b) $p_{ts} = 0.1$; (c) $p_{ts} = 0.5$; (no. of agents=4; grid size=100 $\times$ 100, no. of targets=100).

In summary, we can note that adding LSTM is beneficial in both $p_{tr} \neq 0$ and $p_{ts} \neq 0$ cases. Specifically, more is the benefit with higher p_{ts} and higher p_{tr} scenarios.

3.5 Summary

We have presented a multi-agent deep reinforcement learning algorithm based on multi-agent distributed actor-critic framework for multi-UAV search missions. Specifically, during training, the critic on each UAV is augmented with the states (i.e., full observability) and actions of all UAVs to solve the non-stationarity problem inherent to multi-agent settings. The proposed framework is evaluated both under ideal and non-ideal communication settings and shown to perform better than other MARL algorithms.

Chapter 4

HCFContext: UAV Context Inference via Sequential History-based Collaborative Filtering

In this chapter, we mainly propose techniques for context (e.g., current location, battery level, etc.) modeling and prediction for mobile phones. The same techniques (i.e., context modeling and prediction)are equally applicable for a group of UAVs executing a mission/task, as we explain later in Chapter 7. In case of UAVs, the context can be a combination of location, remaining operational time, etc. Henceforth in this chapter, we will focus mainly on the context attributes of a mobile phone only.

Mobile device applications provide an increasing number of features customized to match users' needs. These needs are very often inferred from specific features such as the user location, activity (e.g., running, walking, driving), surrounding people, interacting people, the current app usage on the device, etc. These features collectively define a specific user (mobile) *context*. Mobile applications are increasingly making use of these contexts such as location-based services (e.g., Foursquare, Google Now, Weather updates, etc.), enhanced reality applications (Pokemon GO [67]), continuous authentication, etc. However, to enable these services, context inference is a much needed and important step.

4.1 Introduction

Most of the existing work focuses on obtaining mobile context instantaneously from sensors which could possibly be hacked, noisy or insufficient and as such cannot be relied in certain security applications. For example, see our previous work on privacy-preserving, distributed, smartphone localization framework [68]. Hence we take a different approach to that problem in this chapter by modeling mobile context based on past context data.

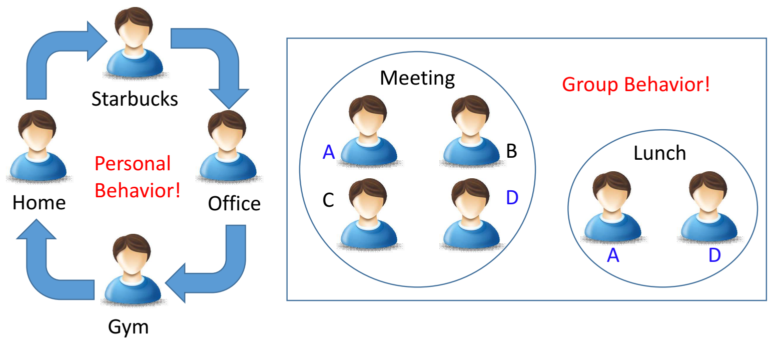


Figure 4.1: A user's personal (left) and group/collaborative filtering (right) behavior that can provide some clues about user's context at any given time.

There are many advantages of modeling the user context by leveraging the sequential nature of context information in a user's history as it can be used to predict the current or future contexts. The former can be used to validate and/or enhance the possibly hacked/noisy/insufficient sensor context, while the latter can provide some information ahead of time to the benefit of the user [69]. For example, a user's general routine during weekdays could be to head first to Starbucks near his home, then to his work and then to Gym and back to home as shown in Fig. 4.1(left). *The learned model will capture this behavior and can be used to validate the location of the user obtained via GPS at 5:30 pm to be at Gym (current context prediction) or display coupons related to Starbucks on his phone in advance (future context prediction).* The latter can be leveraged by mobile personal assistant technologies such as Apple Siri/Google Now for user's benefit.

Motivation: One of the context-aware services is the enterprise data access control as in [70], where policies are defined for enterprise data access based on the phone's context (e.g., connected Wi-Fi, Cell ID, time, etc.). For example, a policy may be defined to allow the phone to be used to attend a confidential meeting or open a confidential document only when the phone's context is found to be within a given location and time. In these secure scenarios, it is not suggested to rely solely on the context obtained from the phone's sensors (e.g., GPS/WiFi and System clock to give location and time) because *they could be hacked unknowingly to the user.* For example, a virus might change the system clock to show different time or spoof the GPS [71] to show different location.

However it is hard to hack a model (more so, a collaborative one) that is learned over a long period of time. Hence our solution can be used to *validate* the context directly obtained from sensors at that instant. Secondly, it is possible that context from sensors is *noisy (due to malfunction) or does not contain enough information.* For example, in the case of a tablet or old mobile phone, it may not be able to acquire (accurate) GPS signal. As such it will be helpful if there is another way of obtaining this information such that it complements the context from sensors. Thirdly, as mentioned earlier, future context prediction is useful in certain context-aware services such as mobile personal assistant technologies (Google Now, Apple Siri, etc.), which can help *pre-fetch information/pre-plan based on the predicted context.*

Our Approach: In order to address the above issues, we first propose a personalized model (*HPContext*) (where *HP* stands for History-based Personalized) that predicts the user's context based on its past sequential history of contexts. Obtaining context through two approaches—sensors and personalized model—adds an extra layer of confidence to the obtained context. However, the following situations are possible: contexts obtained from both approaches are very different, contexts from one of the approaches is not available (e.g., GPS may not be available from phone sensors indoors, etc.) or insufficient leading to uncertainty. In such situations, assuming the user is closely connected to a group of people (e.g., same team colleagues in the company as shown in Fig. 4.1(right), gym friends, family members, etc.), *it is possible that the context of other members in that group of people can provide additional information about his/her context.* For example, assume users A and D often go to lunch together (learned via model). Now somehow if it is known that D is going to "Restaurant1" tomorrow for lunch, it is most likely that the context of user A tomorrow around 1 pm is "having lunch with D at Restaurant1" without having to rely on A's phone sensors at that instant. The chapter explores this aspect of context to provide a second layer of confidence to the context (over and above the personalized model). For this purpose, we propose to use such context obtained through *collaborative filtering* of the contexts of users closely related to the primary user (*HCFContext*, where *HCF* stands for History-based Collaborative Filtering). To the authors' best knowledge, this is the first work to explore collaborative filtering for mobile contexts that can be used to *validate* and/or *enhance* the *current* context obtained from sensors or predict the *future* context for mobile personal assistant technologies. Additionally we present a privacy-preserving method for parameter estimation (training) of *HCFContext*, as users may not be willing to share their private data with each other for the same.

Main Contributions can be summarized as follows.

- We propose a personalized (*HPContext*) and collaborative filtering (*HCFContext*) model to *predict* the users context at any given instant (including future) based on the sequential history of past contexts and based on the theory of Hidden Markov Models (HMMs). We design a novel emission model for these HMMs by considering the unique features and the practicalities of a mobile context (e.g., GPS from sensors may not be always available).
- We present a homomorphic encryption based privacy-preserving approach for training the *HCFContext*.
- We validate the efficacy of the proposed models by testing them on a real-life data set belonging to five graduate students collected over two months. We also evaluate our privacy-preserving approach to study its trade-offs.

Chapter Outline: In Sect. 4.2, we present the related work and position our chapter. In Sect. 4.3, we discuss the proposed models, (*HPContext*, *HCFContext*) and the privacy-preserving approach for the parameter estimation of *HCFContext*. In Sect. 4.4, we present the results of our proposed approaches. Finally, in Sect. 4.5, we summarize the chapter.

4.2 Related Work

In this section, we position our work with respect to previous works (i) that obtain context with and without users sequential history, (ii) via local collaborative sensing, and (iii) related to privacy-preserving collaborative filtering.

Without Sequential History: There is existing work on modeling mobile contexts without considering the sequential nature of context information. For example, Bao et al. [72] propose an unsupervised approach to model mobile context from raw contextual data using Latent Dirichlet Allocation (LDA). Srinivasan et al. [73] mine the co-occurrences of certain context attributes; frequently and simultaneously occurring context attributes are formulated as association rules to predict what else the user will do (e.g., read comics) given a current context attribute (e.g., listen to jazz). However, unlike ours these approaches do not exploit temporal dependencies among contexts but only consider the behavior at a given instant.

With Sequential History: There is also work that exploits the sequential/temporal dependencies between contexts. For example, Mukherji et al. [74] present Mobile Sequence Miner (MSM) framework that mines frequent sequences occurring in app usage patterns, location visits, and call logs using a frequency-based approach. Farrahi et al. [75] present a probabilistic approach to mine mobile phone data (e.g., location) sequences using Distant N-Gram Topic Model (DNTM) where they model the sequence to be dependent on the starting element of the sequence. There are works that model the user activity using HMM based on sensor measurements [76–79]. For example, Trabelsi et al. [80] propose an unsupervised approach for activity recognition based on HMM regression where they segment the sensor data into multiple activities. Cilla et al. [81] use HMM for human activity recognition, with features selected using genetic algorithms that maximize the accuracy of the HMM. Mannini et al. [76] use HMMs for human physical activity classification from on-body accelerometers where they have modeled the HMM emission probabilities as Gaussian Mixture Models (GMMs). Brand et al. [78] and Oliver et al. [79] use HMM for detecting human activities and interactions in video. Lin et al. [77] use feature-guided HMM to segment automatically and identify human motion for physical rehabilitation exercises. Even though these approaches exploit the sequential nature of contextual information, they neither consider collaborative filtering nor privacy-preserving aspects like we do. We claim that collaborative filtering context has additional context information than context obtained from personal history alone.

Local Collaborative Sensing: There are also works on local collaborative sensing; however, these works do not consider the sequential nature of past information into the collaboration process [82]. For example, Honicky et al. [83], while monitoring air quality by integrating air-quality sensors into mobile phones, use collaboration to improve accuracy in over-sampled regions by averaging the readings from several nearby sensors. Castro et al. [82] use collaborative sensing to decide which mobile phone should capture audio when two or more devices are potentially recording a similar audio signal to reduce energy consumption. Mantyjarvi et al. [84] present a collaborative sensing approach where a device, upon noticing a change in its local context beyond a threshold value, requests contexts from its surrounding devices so as to increase the accuracy of its context vector. Miluzzo et al. [85] use collaboration to increase the confidence of the sensed context through consensus of contexts sensed at surrounding devices. These approaches however do not consider past sequential nature of context information into collaboration. Here, each device carries out its own sensing independently of others, and then consensus is applied to increase the confidence. In all these works, collaboration is mostly used as a form of “averaging"; also, these approaches do not consider the past sequential nature of context information in the collaboration process.

Privacy-preserving Collaborative Filtering: There is existing literature in the domain of privacy-preserving collaborative filtering and HMM techniques, which can be broadly classified into two categories—*data perturbation/randomization* to hide the original data albeit with accuracy loss and *data encryption* with typically no accuracy loss albeit with higher computational complexity. On the former, Polat et al. [86] and Parameswaran et al. [87] present privacy-preserving collaborative filtering techniques based on randomized perturbation and data obfuscation respectively. On the latter, Guo et al. [88] present a privacy-preserving Markov model for sequence classification using homomorphic and ElGamal cryptographic systems. Nguyen et al. [89] present a homomorphic encryption based technique for a multi-party HMM. Renckes et al. [90] present a homomorphic-encryption-based method for HMM parameter estimation and forecasting for vertically and horizontally distributed data. More works in this category can be found in [91–94]. We present an approach, designed specifically for our scenario,

Table 4.1: Example context observations of user u for times $t = 1, ..., T$.

Time (t)	**Context observation of user u at time t ($\boldsymbol{O}_{tu}$)**
t_1	WiFi: *wifi1*, CellID: *cid1*, LAC: *lac1*, Battery Level: *high*, Battery Status: *discharging*, Day Period: *morning*, Day of week: *Monday*, Holiday: *No*
t_2	WiFi: *wifi2*, CellID: *cid2*, LAC: *lac2*, Battery Level: *medium*, Battery Status: *discharging*, Day Period: *noon*, Day of week: *Monday*, Holiday: *No*
....	
t_T	WiFi: *wifi1*, CellID: *cid1*, LAC: *lac1*, Battery Level: *low*, Battery Status: *charging*, Day Period: *night*, Day of week: *Sunday*, Holiday: *Yes*

that extends the ideas in this category for privacy preserving multi-party parameter estimation of our *HCFContext* model.

4.3 Proposed Solution

In this section, we describe *HPContext* and *HCFContext* models and a privacy-preserving approach for parameter estimation of *HCFContext*.

Problem Formulation: We present here the proposed *HCFContext* model *by designing a novel emission model of a HMM taking into account the multi-user collaborative filtering aspects, as well as the unique features of the mobile context and its practical issues such as feature unavailability.* We first start with notation—capital letters denote random variables, whereas their small equivalents are their realizations. A vector variable will be indicated in bold. We model context (C_t) as the latent variable of the HMM. For a given user, the observation corresponding to a context state at time t will be called context observations ($\boldsymbol{O}_t$). An example of a user's context observations from time $t = 1...T$ is shown in Table 4.1. Each observation, $\boldsymbol{O}_t$, consists of a set of contextual feature-value pairs. These observations are obtained at regular time intervals (e.g., a minute to four hours). It can be seen as an example from the table that the context observation at $t = t_1$ corresponds to morning when the user is at home (battery is high, probably because the user charges her phone the previous night). The observation at

$t = t_2$, say after 4 hours, can be interpreted as being at office or workplace (change of WiFi, Cell ID, etc.) with battery level being in medium range. Finally, the observation at $t = t_T$ (after several days) can be taken to be again at home in the night. We assume that K number of latent context states spans across these T observations. Considering users $u = 1, ..., M$, each user has a similar set of T observations. Plate notation [95] for our *HCFContext* model for M users is shown in Fig. 4.2. In plate notation, the number of different categorical values a random variable can take is shown inside the circle or rectangle. A circle is used for a random variable while a rectangle is generally used for hyperparameters. Observable variables are shaded. The number of repetitions of a rectangular block is shown at its bottom right corner. For a given user u, the observation at time t, $\boldsymbol{O}_{tu}$ is a set of feature-value pairs (as in a row of Table 4.1). We can write $\boldsymbol{O}_{tu} = (\boldsymbol{f}_{tu}, \boldsymbol{v}_{tu}) = (f_{t,u,i}, v_{t,u,i})_{i=1}^{|f_{tu}|}$, where $|f_{tu}|$ is the number of available features of user u, at time t. Generation of each variable in Fig. 4.2 is described next.

<u>Initial State Model:</u> A prior distribution of *contexts*, π is generated from prior Dirichlet distribution, η. C_1 is then generated from π. We will assume a total of K possible context states for *HCFContext* over all M users.

<u>Transition Model:</u> A prior *transition* distribution of contexts, $\rho_{c_{t-1}} = \rho_k$, is generated from a prior Dirichlet distribution, ω_k. C_t is then generated from $\rho_{c_{t-1}}$ for a given C_{t-1}. Note that ρ_k and ω_k can take a total of K categorical values (C_t, current state) for each k (C_{t-1}, previous state).

<u>Novel Emission Model:</u> For $\boldsymbol{O}_t$ generation under each C_t, since features are not always available (e.g., GPS is not available when indoor or underground, etc.), we will define a separate distribution for features (F_t) to account for their availability and then another distribution to obtain the values (V_t) for those features at time t, as illustrated in Fig. 4.2. F_t is dependent on C_t, whereas V_t is dependent on both C_t and F_t. An initial *feature* distribution, $\theta_{c_t,f_t} = \theta_{k,f}$, is generated from a prior Dirichlet distribution, $\delta_{k,f}$. Feature F_t is then generated from $\theta_{k,f}$, which can take two categorical values—whether the feature is present or not for the given context, c_k. We will assume a total of F possible features over all observations of all users. A prior *value* distribution, $\phi_{c_t,f_t} = \phi_{k,f}$, is generated from a prior Dirichlet distribution, $\lambda_{k,f}$. Value V_t is then generated from ϕ_{c_t,f_t}

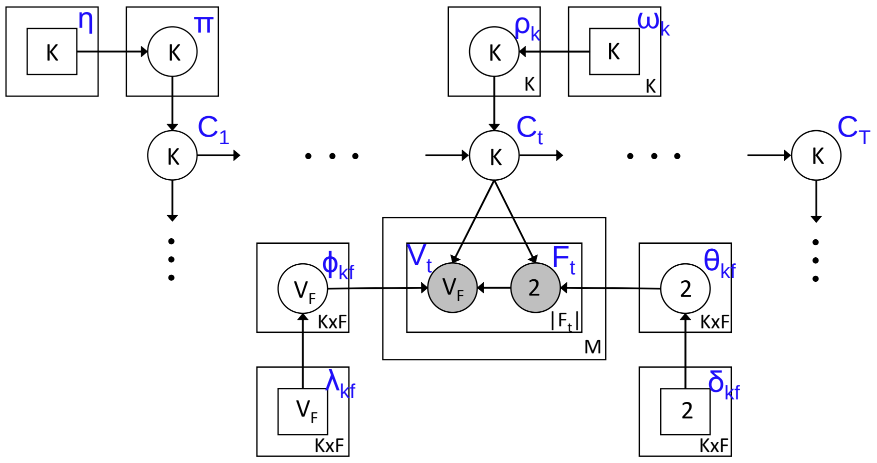


Figure 4.2: Plate notation of *HCFContext*. F is the total number of features, V_F denotes the total number of possible values for feature F, $|F_t| = |V_t|$ denotes the number of features observed at time t, M is the number of users, and K is the number of hidden context states.

for a given context c_t and feature $f_{t,i}$. For a given feature f, we will assume V can take V_f possible values. The priors will be chosen such that the summation and non-negativity constraints on the parameters are satisfied and also to encode prior information. For example, if it is known that a user frequently moves between home to work, the prior parameter for this transition, (ω_k), is given a high value. Priors are important to obtain Maximum A Posteriori (MAP) estimates as Maximum Likelihood (ML) estimates fare poorly with limited training data.

Parameter Estimation (Training): Given these parameters, $\boldsymbol{\Psi} = \{\Pi, P, \Theta, \Phi\}$, the parameter space of $\pi, \rho_k, \theta_k, \phi_{k,f}$ and their hyperparameters, $\{\eta, \omega, \delta, \lambda\}$, we can write down the joint probability of all latent contexts $\boldsymbol{C} = \{C_1, ...C_T\}$ and all context observations, $\boldsymbol{O} = \{O_1, ...O_T\}$, as $P(\boldsymbol{C, O} \mid \boldsymbol{\Psi}, \eta, \omega, \delta, \lambda)$ as follows:

$$P(\boldsymbol{C, O} \mid \boldsymbol{\Psi}, \eta, \omega, \delta, \lambda) = \int P(\pi \mid \eta) P(c_1 \mid \pi) d\pi$$
$$\times \int \prod_{k=1}^{K} P(\rho_k \mid \omega_k) \prod_{t=2}^{T} P(c_t \mid c_{t-1}, \rho_{c_{t-1}}) d\rho$$
$$\times \int \prod_{k=1}^{K} \prod_{f=1}^{F} P(\theta_{k,f} \mid \delta_{k,f}) \prod_{u=1}^{M} \prod_{t=1}^{T} \prod_{i=1}^{2} P(f_{t,i,u} \mid c_t, \theta_{c_t}) d\theta$$

$$\times \int \prod_{k=1}^{K} \prod_{f=1}^{F} P(\phi_{k,f} \mid \lambda_{k,f})$$

$$\prod_{u=1}^{M} \prod_{t=1}^{T} \prod_{i=1}^{V_f} P(v_{t,i,u} \mid c_t, f_{t,i,u}, \phi_{c_t,f_t}) d\phi. \tag{4.1}$$

In the above equation, note that the integration is carried out over the entire space of the respective parameters (Π, P, Θ, Φ).

Likelihood of the observations $\boldsymbol{O}$ is then,

$$L(\boldsymbol{O}) = \sum_{\boldsymbol{c}} P(\boldsymbol{C}, \boldsymbol{O} \mid \boldsymbol{\Psi}, \eta, \omega, \delta, \lambda). \tag{4.2}$$

Training the HMM involves finding the parameters $\boldsymbol{\Psi}$ that maximize the likelihood in (4.2). Given the complex nature of (4.2), it is very difficult to derive a closed-form solution of $\boldsymbol{\Psi}$. Hence, we make use of an iterative approach called Expectation Maximization (EM). The EM algorithm consists of two steps, Expectation Step (E-Step) and Maximization Step (M-Step), that need to be iterated until the convergence of the likelihood.

In the **E-Step**, we assume the parameters are fixed (from the previous iteration, $\boldsymbol{\Psi}_{prev}$) and calculate the posterior distribution of the latent variables $p(\boldsymbol{C} \mid O, \boldsymbol{\Psi}_{prev})$. Using this posterior distribution, we calculate the expectation of the logarithm of the complete-data likelihood (including the latent context variables), $Q(\boldsymbol{\Psi}, \boldsymbol{\Psi}_{prev})$, as follows [96],

$$Q(\boldsymbol{\Psi}, \boldsymbol{\Psi}_{prev}) = \sum_{c} p(\boldsymbol{C} \mid \boldsymbol{O}, \boldsymbol{\Psi}_{prev}) \ \mathrm{ln} p(\boldsymbol{C}, \boldsymbol{O} \mid \boldsymbol{\Psi}) \tag{4.3}$$

We will introduce two new variables, $\gamma(C_t)$ *and* $\xi(C_{t-1}, C_t)$, to simplify the expression in (4.3),

$$\gamma(C_t) = p(C_t \mid \boldsymbol{O}, \boldsymbol{\Psi}_{prev}), \tag{4.4}$$

$$\xi(C_{t-1}, C_t) = p(C_{t-1}, C_t \mid \boldsymbol{O}, \boldsymbol{\Psi}_{prev}). \tag{4.5}$$

To simplify notation, we will denote $\gamma(C_t = c_{tk}) = \gamma(c_{tk})$ and similarly for $\xi(c_{t-1,j}, c_{tk})$. After plugging (4.1) in (4.3), and using the definitions of $\gamma(C_t)$ and $\xi(C_{t-1}, C_t)$, we can rewrite (4.3) as follows,

$$Q(\boldsymbol{\Psi}, \boldsymbol{\Psi}_{prev}) = \sum_{k=1}^{K} \gamma(c_{1k}) \ln \pi_k$$

$$+\sum_{t=2}^{T}\sum_{j=1}^{K}\sum_{k=1}^{K}\xi(c_{t-1,j},c_{tk})\ln\rho_{jk}$$
$$+\sum_{u=1}^{M}\sum_{t=1}^{T}\sum_{k=1}^{K}\sum_{i=1}^{\boldsymbol{f}_t}\gamma(c_{tk})\ln p(f_{t,i,u}\mid c_{tk},\theta_{k,f})$$
$$+\sum_{u=1}^{M}\sum_{t=1}^{T}\sum_{k=1}^{K}\sum_{i=1}^{\boldsymbol{f}_t}\gamma(c_{tk})\ln p(v_{t,i,u}\mid f_{t,i,u},c_{tk},\phi_{k,f}). \tag{4.6}$$

In the **M-Step**, we treat $\gamma(C_t)$ *and* $\xi(C_{t-1},C_t)$ as constants and find the parameters $\boldsymbol{\Psi}_{curr}$ that maximize $Q(\boldsymbol{\Psi},\boldsymbol{\Psi}_{prev})$ i.e.,

$$\boldsymbol{\Psi}_{curr}=\underset{\boldsymbol{\Psi}}{\operatorname{argmax}}\,Q(\boldsymbol{\Psi},\boldsymbol{\Psi}_{prev}). \tag{4.7}$$

This maximization can be carried out using Lagrange multipliers [96] with $\boldsymbol{\Psi}_{curr}$ as (note M for *collaborative filtering*),

$$\pi_k=\frac{\gamma(c_{1k})+\eta_k}{\sum_{k'=1}^{K}(\gamma(c_{1k'})+\eta'_k)}, \tag{4.8}$$
$$\rho_{kj}=\frac{\sum_{t=2}^{T}\xi(c_{t-1,k},c_{tj})+\omega_{kj}}{\sum_{j'=1}^{K}\sum_{t=2}^{T}\xi(c_{t-1,k},c_{tj'})+\sum_{j'=1}^{K}\omega_{kj'}}, \tag{4.9}$$
$$\theta_{k,f}=\frac{\sum_{t=1}^{T}\gamma(c_{tk})\sum_{u=1}^{M}\boldsymbol{I}(f\in\boldsymbol{f}_{tu})+\delta_{k,f}}{M\sum_{t=1}^{T}\gamma(c_{tk})+\sum_{f'=1}^{F}\delta_{k,f'}}, \tag{4.10}$$
$$\phi_{k,f,v}=\frac{\sum_{u=1}^{M}\sum_{t:f\in\boldsymbol{f}_{tu}}\gamma(c_{tk})\boldsymbol{I}(v_{t,f,u}=v)+\lambda_{k,f,v}}{\sum_{u=1}^{M}\sum_{t:f\in\boldsymbol{f}_{tu}}\gamma(c_{tk})+\sum_{v'=1}^{V_f}\lambda_{k,f,v'}}, \tag{4.11}$$

where $\boldsymbol{I}(x)$ is the indicator function with $\boldsymbol{I}(x)=1$ if x is true and 0 otherwise. In order to evaluate efficiently $\gamma(C_t)$ and $\xi(C_{t-1},C_t)$ of the E-Step, we will follow the forward-backward algorithm [96], for which we will define two new variables,

$$\alpha(C_t)\doteq p(\boldsymbol{o}_{1:t},C_t), \tag{4.12}$$
$$\beta(C_t)\doteq p(\boldsymbol{o}_{t+1:T}\mid C_t). \tag{4.13}$$

We can now write $\gamma(C_t)$ and $\xi(C_{t-1},C_t)$ in terms of $\alpha(C_t)$ and $\beta(C_t)$ as follows:

$$\gamma(C_t)=p(C_t\mid\boldsymbol{O})=\frac{p(\boldsymbol{O}\mid C_t)}{p(\boldsymbol{O})} \tag{4.14}$$
$$=\frac{p(\boldsymbol{o}_{1:t}\mid C_t)p(\boldsymbol{o}_{t+1:T}\mid C_t)p(C_t)}{p(\boldsymbol{O})} \tag{4.15}$$
$$=\frac{\alpha(C_t)\beta(C_t)}{p(\boldsymbol{O})}, \tag{4.16}$$

where (4.15) is a result of independence (d-separation). Upon summing (4.16) on both sides over C_t, we find that the left-hand side sums up to one, giving us the relation between the likelihood function, $p(\boldsymbol{O})$, and $\alpha(C_t)$ and $\beta(C_t)$,

$$p(\boldsymbol{O}) = \sum_{C_t} \alpha(C_t)\beta(C_t) \tag{4.17}$$

We can express $\xi(C_{t-1}, C_t)$ in a similar manner,

$$\xi(C_{t-1}, C_t) = p(C_{t-1}, C_t \mid \boldsymbol{O}) = \frac{p(\boldsymbol{O} \mid C_{t-1}, p(C_t)C_{t-1}, C_t)}{p(\boldsymbol{O})} \tag{4.18}$$

$$= \frac{p(\boldsymbol{o}_{1:t-1} \mid C_{t-1})p(\boldsymbol{o}_t \mid C_t)p(\boldsymbol{o}_{t+1:T} \mid C_t)p(C_t \mid C_{t-1})p(C_{t-1})}{p(\boldsymbol{O})} \tag{4.19}$$

$$= \frac{\alpha(C_{t-1})p(\boldsymbol{o}_t \mid C_t)p(C_t \mid C_{t-1})\beta(C_t)}{p(\boldsymbol{O})}, \tag{4.20}$$

where (4.19) is again a result of d-separation rule.

Hence, we have:

$$\gamma(C_t) = \frac{\alpha(C_t)\beta(C_t)}{p(\boldsymbol{O})}, \tag{4.21}$$

$$\xi(C_{t-1}, C_t) = \frac{\alpha(C_{t-1})p(\boldsymbol{o}_t \mid C_t)p(C_t \mid C_{t-1})\beta(C_t)}{p(\boldsymbol{O})} \tag{4.22}$$

Given this, the variables, $\alpha(C_t), \beta(C_t)$ can now be recursively computed (forward and backward relations) respectively as follows,

$$\alpha(C_t) = p(\boldsymbol{o}_t \mid C_t) \sum_{C_{t-1}} \alpha(C_{t-1})p(C_t \mid C_{t-1}), \tag{4.23}$$

$$\beta(C_t) = \sum_{C_{t+1}} \beta(C_{t+1})p(\boldsymbol{o}_{t+1} \mid C_{t+1})p(C_{t+1} \mid C_t), \tag{4.24}$$

where $\beta(c_{Tk}) = 1$ for $k = 1, ..., K$. Using these definitions, we can rewrite the likelihood, $p(\boldsymbol{O})$, as follows,

$$p(\boldsymbol{O}) = \sum_{k=1}^{K} \alpha(c_{Tk}). \tag{4.25}$$

We will now express $\alpha(C_t)$ and $\beta(C_t)$ in terms of the parameters $\boldsymbol{\Psi}_{prev}$. Before that, we will compute the full observation probability $p(\boldsymbol{o}_t \mid c_{tk}) = \mu_{tk}$,

$$\mu_{tk} = p(\boldsymbol{o}_t \mid c_{tk}) = \prod_{u=1}^{M} \mu_{tk_u} = \prod_{u=1}^{M} \prod_{\substack{f \in \boldsymbol{f}_{tu} \\ v \in \boldsymbol{v}_{tu}}} \theta_{k,f} \ \phi_{k,f,v}. \tag{4.26}$$

We can now express the forward relation in (4.23) in terms of the parameters as follows,

$$\alpha(c_{1k}) = \pi_k \mu_{1k} \text{ (for t = 1)}, \tag{4.27}$$

$$\alpha(c_{tk}) = \mu_{tk} \sum_{j=1}^{K} \alpha(c_{t-1,j}) \rho_{jk} \text{ (for t = 2 to T)}. \tag{4.28}$$

Similarly, we can express the backward relation in (4.24) as follows,

$$\beta(c_{tk}) = \sum_{j=1}^{K} \beta(c_{t+1,j}) \mu_{t+1,j} \rho_{kj}. \tag{4.29}$$

Finally, we can express $\xi(C_{t-1}, C_t)$ *and* $\gamma(C_t)$ in (4.20) and then $\gamma(C_t)$ as follows,

$$\xi(c_{t-1,j}, _{tk}) = \frac{\alpha(c_{t-1,j}) \mu_{tk} \rho_{jk} \beta(c_{tk})}{\sum_{k=1}^{K} \alpha(c_{Tk})}, \tag{4.30}$$

$$\gamma(c_{tk}) = \sum_{j=1}^{K} \xi(c_{t-1,j}, c_{tk}). \tag{4.31}$$

In (4.30) and (4.31), we have expressed $\gamma(C_t)$ and $\xi(C_{t-1}, C_t)$ fully in terms of parameters $\boldsymbol{\Psi}$ (we used (4.27), (4.28), and (4.29) to substitute for $\alpha(C_t)$ and $\beta(C_t)$). This constitutes the main computation in the E-Step, which in turn involves computing the likelihood as in (4.25). On the other hand, we have expressed in (4.8), (4.9), (4.10), and (4.11) $\boldsymbol{\Psi}$ in terms of $\gamma(C_t)$ and $\xi(C_{t-1}, C_t)$, which constitutes the main computation in the M-Step. Hence, by iterating between the two steps until the likelihood in (4.25) converges, we will be able to obtain stable parameters. This completes the training of *HCFContext* to obtain its parameters from the training data.

Prediction: We will now use the learned parameters to predict the future observations given past observations. This will be done by first finding the distribution over future states and then by multiplying the distribution over the observations given the future state. In our case, since the observations are feature-value pairs, we will first calculate the distribution over features and then the distribution over values given features. Given that the user has made a sequence of past 't' observations, $\boldsymbol{o}_{1:t} = \{\boldsymbol{o}_1, ..., \boldsymbol{o}_t\}$, the probability that a feature f, and then a value v for that feature, will be observed at time $t+1$ can be computed, respectively, as follows,

$$p(f \in \boldsymbol{f}_{t+1} \mid \boldsymbol{o}_{1:t}) = \sum_{k=1}^{K} p(c_{t+1,k} \mid \boldsymbol{o}_{1:t}) \cdot \theta_{k,f}, \tag{4.32}$$

$$p(v_{t+1,f} = v \mid \boldsymbol{o}_{1:t}) = \sum_{k=1}^{K} p(c_{t+1,k} \mid \boldsymbol{o}_{1:t}) \cdot \phi_{k,f,v}, \tag{4.33}$$

Note that $p(c_{t+1,k} \mid \boldsymbol{o}_{1:t})$ in (4.32), (4.33) is computed as,

$$p(c_{tk} \mid \boldsymbol{o}_{1:t}) = \sum_{j=1}^{K} p(c_{t+1,k} \mid c_{tj}) p(c_{tj} \mid \boldsymbol{o}_{1:t}), \tag{4.34}$$

where $p(c_{t+1,k} \mid c_{tj}) = \rho_{jk}$ and $p(c_{tj} \mid \boldsymbol{o}_{1:t})$ can be recursively computed using a procedure similar to (4.28) (forward algorithm),

$$\alpha'(c_{tk}) = p(c_{tk} \mid \boldsymbol{o}_{1:t}) \tag{4.35}$$

$$= \frac{p(\boldsymbol{o}_t \mid c_{tk}) \sum_{j=1}^{K} p(c_{tk} \mid c_{t-1,j}) p(c_{t-1,j} \mid \boldsymbol{o}_{1:t-1})}{\sum_{k=1}^{K} p(\boldsymbol{o}_t \mid c_{tk}) \sum_{j=1}^{K} p(c_{tk} \mid c_{t-1,j}) p(c_{t-1,j} \mid \boldsymbol{o}_{1:t-1})} \tag{4.36}$$

$$= \frac{\mu_{tk} \sum_{j=1}^{K} \rho_{jk} \alpha'(c_{t-1,j})}{\sum_{k=1}^{K} \mu_{tk} \sum_{j=1}^{K} \rho_{jk} \alpha'(c_{t-1,j})} \tag{4.37}$$

We compute (4.32), (4.33) over all features, $f_i : i = \{1, ..., F\}$, all corresponding values $v_j : j = \{1, ..., V_{f_i}\}$ and pick the most probable ones to get feature-value pairs at $t + 1$.

Determining the Number of Hidden Contexts: So far we have assumed that the number of hidden contexts K is given; however, in general this number needs to be determined automatically from the data. We will now detail an original approach to determine the best K assuming it lies in the range $\mathcal{K}_{range} = [K_{min}, K_{max}]$ and that the extremes can be approximately obtained from prior information about the data. To determine the best $K \in \mathcal{K}_{range}$, we will define a metric called *Perplexity* that determines how well the chosen K fits for prediction tasks over the testing set. Perplexity [97], usually used in machine learning perplexity is a measurement of how well a probability distribution or probability model predicts a sample. It can be used to compare probability models. A low perplexity indicates the probability distribution is good at predicting the sample. In our scenario, we define the perplexity to be the prediction probability over a sequence of observations given a sequence of past observations from the testing set. It can be written as follows,

$$\text{Perplexity} = \exp\left(- \frac{\log p(\boldsymbol{o}_{t+1:T} \mid \boldsymbol{o}_{1:t})}{\sum_{u=1}^{M} \sum_{t=t+1}^{T} |\boldsymbol{o}_{tu}|} \right), \tag{4.38}$$

where $|\boldsymbol{o}_{tu}|$ is the number of features observed at time t for user u and $p(\boldsymbol{o}_{t+1:T} \mid \boldsymbol{o}_{1:t})$ can be determined as follows,

$$p(\boldsymbol{o}_{t+1:T} \mid \boldsymbol{o}_{1:t}) = \sum_{k=1}^{K} p(c_{tk} \mid \boldsymbol{o}_{1:t}) p(\boldsymbol{o}_{t+1:T} \mid c_{tk}), \tag{4.39}$$

where $p(c_{tk} \mid \boldsymbol{o}_{1:t}) = \alpha'(c_{tk})$ (can be computed similar to (4.37) or (4.23)) and $p(\boldsymbol{o}_{t+1:T} \mid c_{tk}) = \beta(c_{tk})$ (can be computed using (4.29)). Intuitively, a small perplexity is desired as explained above. Although in general the perplexity reduces as K increases, a large K is not preferred due to the risk of overfitting. Hence, we make use of the rate of decrease in perplexity to determine when to stop increasing K, e.g., if it falls below a threshold (say 10%), we stop K at that value.

***HPContext* Model:** Personalized model is just a special case of the above collaborative filtering model when the number of users is one (i.e., $M = 1$). The main difference is in the observation probability. Specifically, for user u,

$$\mu_{tk} = p(\boldsymbol{o}_{tu} \mid c_{tk}) = \mu_{tk_u} = \prod_{\substack{f \in \boldsymbol{f}_{tu} \\ v \in \boldsymbol{v}_{tu}}} \theta_{k,f} \ \phi_{k,f,v}. \tag{4.40}$$

Remaining equations remain the same with setting $M = 1$.

Privacy-preserving Multi-party Computing: *Since the users could possibly be in different contexts while training it is important to preserve their privacy.* In this section, we develop algorithms for multi-party parameter estimation of *HCFContext* while preserving each party's privacy. Hence model parameters need to be jointly estimated when the individual observations are encrypted. Since the model parameters $\boldsymbol{\Psi}$ are known to everyone, the prediction can be carried out by each party on their own. Before we present the algorithms, we provide the threat/adversary model.

Threat/Adversary Model: We use a semi-honest setting, where parties keep all their intermediate computations private, and we assume that P_m will not collude with P_1 and disclose encrypted values received. The key generation in our security model will be following a standard key exchange mechanism [98] without the need of a third party entity. Consequently, through underlying guarantees by the cryptographic multi-party computation primitives, any malicious party (either compromised by an external adversary or playing as an insider threat) cannot get access to private observations of other parties without authorization. Compared to past similar solutions that extract context based on phone sensors, for secure policy enforcement (e.g., Swirls [70]), our proposed collaborative filtering-based solution uses sensor information from multiple parties. Past techniques (e.g., Swirls [70]), which rely on sensor measurements from only one device, assume that device's operating system as the Trusted Computing Base (TCB) [99]. Hence, if the device is compromised, no security properties can be guaranteed because the sensor measurements may be corrupted. However, in our case, the use of measurements from multiple devices raises the security bar against such adversaries. One may argue that the models themselves may be unreliable as they are based on historical sensor data. Our approach provides protection against this point by comparing sensor context with that obtained from *HPContext* and *HCFContext*, and alerting the user in case of significant differences and also adaptively learning to ignore false positives based on similar such observances in the past.

Algorithms: The algorithms we developed for this (extended from [89]) are based on the following well-known primitives—*homomorphic encryption* [100], *secure logsum*, and *secure negation* [101] which are briefly introduced below.

Homomorphic Encryption: It relies on a public-private crypto-system that has the property of mirroring plaintext operations on ciphertexts while not revealing either the parameters of the operation or the result. In practice, it allows us to perform computations on the encrypted text. As such, the following homomorphic properties hold for the Paillier crypto-system: $D[E[s_1] \times E[s_2]] \sim s_1 + s_2$ and also $D[E[s_1]^{s_2}] \sim s_1 \times s_2$, for some plain-texts, s_1, s_2 (here E and D stand for encryption and decryption respectively; E not to be confused with 'expectation'). Accordingly, we can add or multiply numbers by respectively multiplying or taking the exponent of their ciphertext. For further details on the Paillier crypto-system, please see [100].

Secure Logsum: Secure logsum is a simple protocol that uses the homomorphic properties of the Paillier crypto-system. Consider two parties a and b where a has a vector of encrypted values $(E[\log x_1], E[\log x_2], \dots E[\log x_n])$, and b holds the decryption key. a and b want to jointly compute $E[\log \sum_{i=1}^{n} a_i x_i]$, for a public vector $(a_1, ..., a_n)$. This operation relies on computing for both parties the vector $(E[x_1], \dots, E[x_n])$ using the

Algorithm 4.1 Secure multi-party computation of $P(\boldsymbol{O} \mid \boldsymbol{\Psi})$.

```
Input: Parties $P_1, P_2, \ldots, P_M$ know the model $\boldsymbol{\Psi}$. Each party has a set of private observations
    $\boldsymbol{O}_{tu}=(\boldsymbol{f}_{1u}, \boldsymbol{v}_{1u}), (\boldsymbol{f}_{2u}, \boldsymbol{v}_{2u}), \ldots, (\boldsymbol{f}_{Tu}, \boldsymbol{v}_{Tu})$.
Output: $P(\boldsymbol{O} \mid \boldsymbol{\Psi}, \eta, \omega, \delta, \lambda) = \sum_{k=1}^{K} \alpha(c_{Tk})$
        Initialization (t=1):
 1: for $k = 1, \ldots, K$ do
 2:     for all $P_{q \neq m}$ do
 3:         $P_q$ sends $E[\log(\mu_{1k_x})]$ to $P_m$,
 4:     end for
 5:     $P_m$ computes $E[\log(\prod_{u=1}^{M} \mu_{1k_u})]$ using (4.26)
 6:     $P_m$ computes $E[\log(\alpha(c_{1k}))]$ using (4.27)
 7: end for
        Induction:
 8: for $t = 2, \ldots, T$ do
 9:     Repeat steps 2-6, replacing time index with $t$, so that $P_m$ obtains $E[\log(\prod_{u=1}^{M} \mu_{tj_u})]$ for $j =$
    $1, \ldots, K$.
10:     for $k = 1, \ldots, K$ do
11:         $P_m$ and $P_1$ use the secure logsum protocol to compute $E[\log \sum_{j=1}^{K} \alpha(c_{t-1,j})\rho_{jk}]$,
12:         $P_m$ computes $E[\log(\alpha(c_{tk}))]$ using (4.28)
13:     end for
14: end for
        Termination:
15: $P_m$ and $P_1$ use the secure logsum protocol to compute $E[\log P(\boldsymbol{O} \mid \boldsymbol{\Psi})] = E[\log \sum_{k=1}^{K} \alpha(c_{Tk})]$,
16: $P_1$ decrypts the result and sends the value to $P_m$.
```

secure exponent protocol. Details of these protocols can be found in [101].

Secure Negation: The presented algorithm also relies on the ability of two parties a and b to compute the encrypted negation $E[-x]$ of an encrypted value $E[x]$. This can be easily done using the basic application of homomorphic encryption outlined above.

The proposed algorithm for secure multi-party computation of data likelihood, $P(\boldsymbol{O} \mid \boldsymbol{\Psi})$ is shown in self-explanatory Algorithm 4.1 (E in the algorithm refers to encrypted value and not expectation). It details the steps the M parties need to undertake to jointly compute $\alpha(\cdot)$ (as per (4.27), (4.28)) and the log likelihood of their observation data given the model parameters, $P(\boldsymbol{O} \mid \boldsymbol{\Psi})$. We assume only one party (P_1 in Algo. 4.1) has both the private and public keys, while the remaining $M - 1$ parties ($P_m \mid m = 2, \ldots, M$) have only the public key. Hence all parties encrypt their private

Algorithm 4.2 Secure multi-party estimation of $\boldsymbol{\Psi}$.

```
Input: $E[\log \alpha(c_t)]$, $E[\log \beta(c_t)]$, and $E[\log P(\boldsymbol{O} \mid \boldsymbol{\Psi})]$.
Output: The updated model parameters $\boldsymbol{\Psi} = \{\Pi, P, \Theta, \Phi\}$
 1: for $t = 2, \ldots, T$ do
 2:     for $k = 1, \ldots, K$ do
 3:         for $j = 1, \ldots, K$ do
 4:             $P_m$ computes $E[\log \xi(c_{t-1,j}, c_{tk})]$ by taking the log of (4.30)
 5:         end for
 6:         $P_m$ and $P_1$ use the secure logsum to compute $E[\log \gamma(c_{tk})]$ from (4.31)
 7:     end for
 8: end for
 9: $P_m$ uses (4.8), (4.9) to update $E[\log \pi_k], E[\log \rho_{kj}]$.
10: $P_m$ then updates $E[\log \theta_{k,f}]$ and $E[\log \phi_{k,f,v}]$ as in (4.10) and (4.11), for the $M$ parties using the
    secure logsum and negation protocols.
```

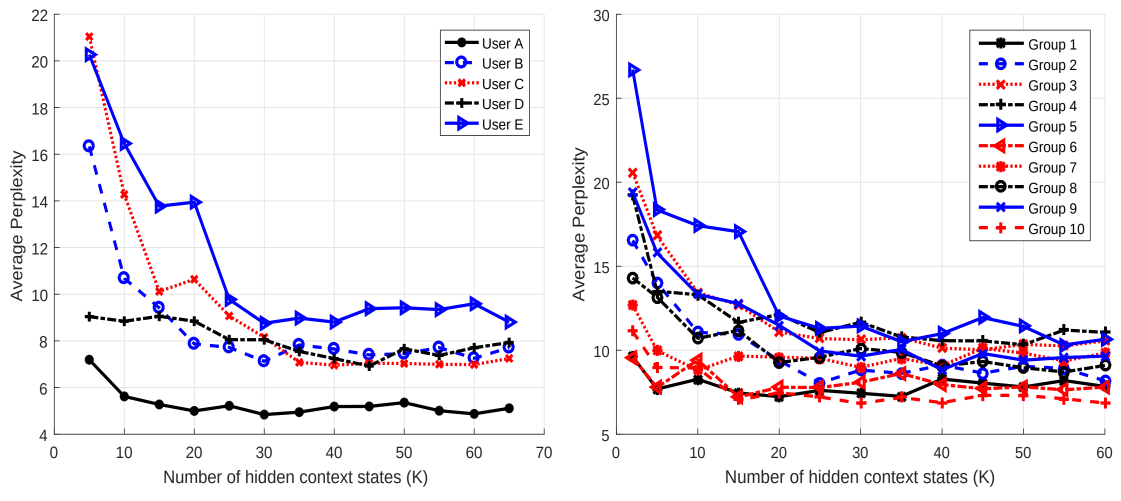
observation data and send it to P_m, where m can be any one of $2, \ldots, M$, which does the computations on this private encrypted data (as it cannot decrypt it since it has only the public key). Whenever it needs to compute secure logsum and secure negation protocols, it consults P_1 which has the private key. Algo. 4.1 can be similarly used to compute $\beta(\cdot)$ as per (4.29). The proposed algorithm for secure multi-party estimation of model parameters, $\boldsymbol{\Psi}$ is shown in Algo. 4.2. It details the steps taken by the M parties to estimate $\boldsymbol{\Psi}$ using $P(\boldsymbol{O} \mid \boldsymbol{\Psi}), \alpha(\cdot), \beta(\cdot)$ computed from Algo. 4.1. In Algo. 4.2, P_m first computes $\xi(c_{t-1,j}, c_{tk}), \gamma(c_{tk})$ as per (4.30) and (4.31) respectively. It then computes the model parameters, $\boldsymbol{\Psi}$ as per eqs. (4.8) to (4.11).

Computational Complexity: The computational complexity of Algo. 4.1 can be seen as $O(MK^2T)$ due to twice-nested for-loop operating for T timesteps for each party, while it is $O(MK^3T)$ for Algo. 4.2 due to triple-nested for-loop, (lines $1-3$).

Floating Point and Negative Numbers: Our algorithms need to encrypt the HMM parameters, which are real numbers. We translate between floating-point numbers and non-negative integers by scaling and rounding off the values. Let c be the scaling factor. A real number r is translated to integer $\bar{r} = \lfloor cr \rfloor$, where $\lfloor x \rfloor$ is the largest integer $\leq x$. We incorporate this operation into the encryption and decryption as $E'[r] = E[\bar{r}] = E[\lfloor cr \rfloor],\ and\ D'[E'[r]] = \bar{r}/c \approx r$. For negative numbers, we use modulo n arithmetic, i.e., negative numbers are represented by their modular additive inverse.

Table 4.2: Lifemap dataset analysis (9 weeks, 5 users, 1 hour sampling).

Feature	**Values**	V_F
WiFi	MAC values of max dBm Access Points	440
Place Name	User defined place name values	168
Cell ID	Cell ID values	316
LAC	Location Area Code values	33
Batt. Level	Low ($< 35\%$), Medium ($35\% - 65\%$), High ($65\% - 85\%$), Full ($> 85\%$)	4
Batt. Status	Charging, Discharging, Full	4
Day Period	Morning: {7 to 11 am}; Noon: {11 am to 2 pm}; Afternoon: {2 to 6 pm}; Evening: {6 to 9 pm}; Night: {9 pm to 7 am}	5
Day Name	Mon, Tue, Wed, Thu, Fri, Sat, Sun	7
Holiday	Yes, No	2



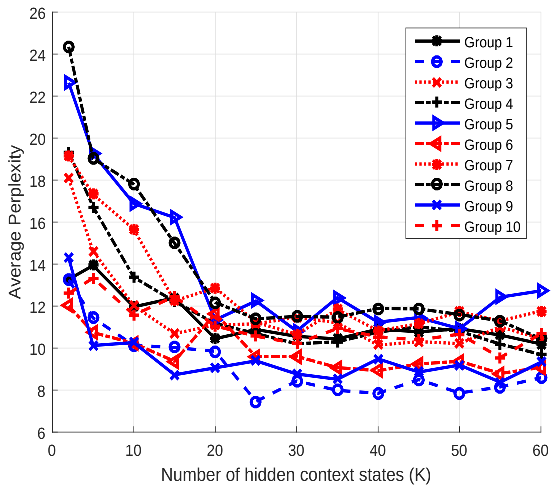
(a) (b)

Figure 4.3: Perplexity vs. K: (a) 1 user case; (b) 2 user case. These figures help identify the best 2 user groups (most related two users).

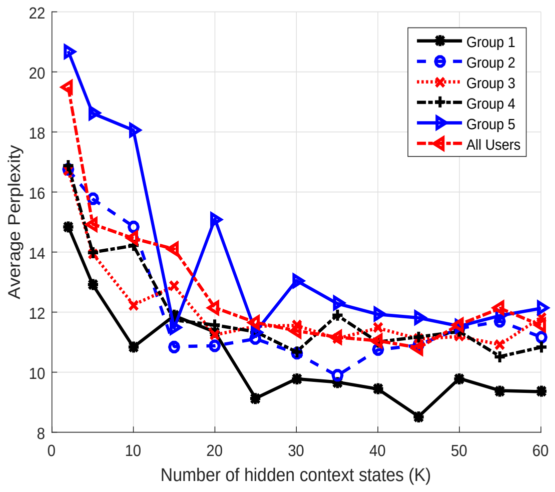
For $r < 0$, $E'[\bar{r}] = E'[\bar{r}+n]$. This means our r is limited to range $[-n/(2c), (n-1)/(2c)]$.

4.4 Performance Evaluation

We describe the experimental setup and results, and evaluate the performance of the privacy-preserving algorithms.

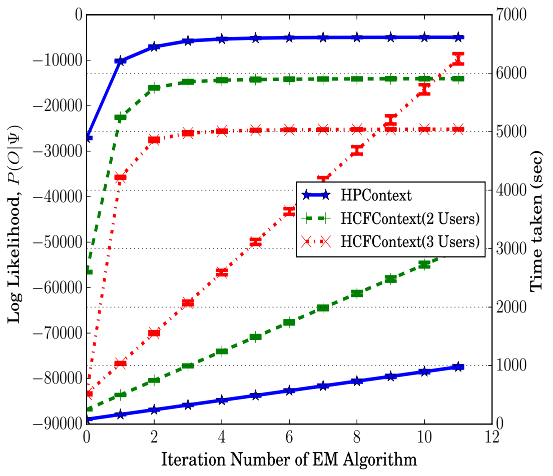


(a)

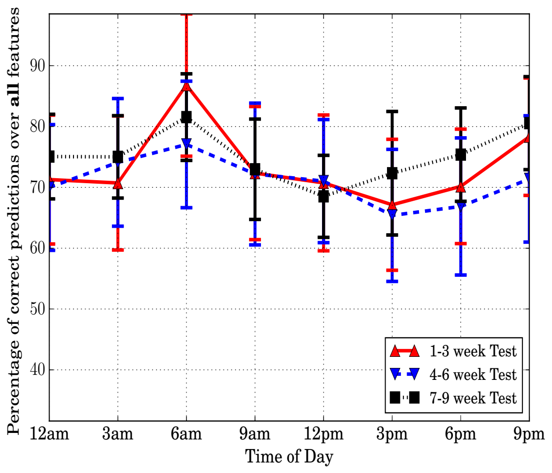


(b)

Figure 4.4: Perplexity vs. K: (a) 3 user case; (b) 4 and all user case.



(a)



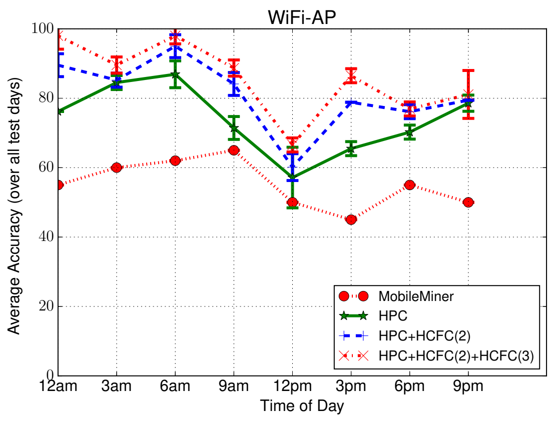
(b)

Figure 4.5: (a) Log likelihood of the training data and time taken vs. number of iterations of EM algorithm for different models; (b) Comparison of performance (*HPContext* + *HCFContext* (2) + *HCFContext* (3)) for different choices of test data - first, mid, and last three weeks.

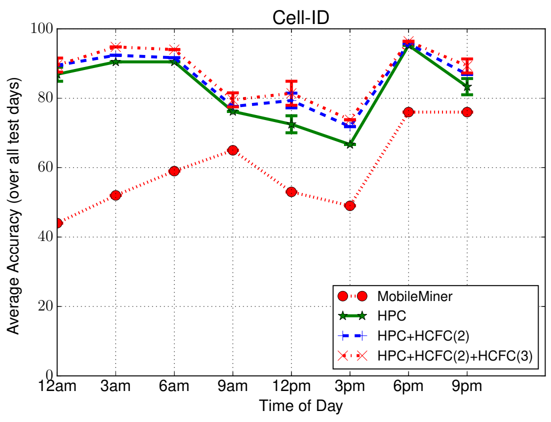
Dataset and Experiment Description: To validate our models, we have used the LifeMap dataset [102], which is freely available online. This dataset consists of fine-grained mobility data such as WiFi fingerprints (MAC address and signal strengths of surrounding Wi-Fi APs), user-defined types of places (workplace, cafeteria, etc.), cell tower ID, etc. These details of 10 users are logged every 2 to 5 minutes for about two months (which is the overlap time among all users) in Seoul, Korea. The users are graduate students of the same lab in the university and as such the data suits our application. We have chosen data corresponding to five users for a period of nine weeks for our experiments. Username, gender and the number of places visited in total for these five users are as follows—*GS2 (A)*, M, 163; *GS3 (B)*, M, 297; *GS4 (C)*, F, 209; *GS7 (D)*, M, 289; *GS12 (E)*, M, 376. The average number of places visited per user is about 270. However, the number of frequently visited places for each user ranges from 8 to 35 (median 20). We also down sampled the data to 1 hour period to ease the computations i.e., the time instants t and $t+1$ are separated by 1 hour. This also means the trained models will be able to capture mostly only those contexts with stay duration of the order of an hour or more. Six weeks of data is used for training and the remaining three week data is used for testing. The list of features we have used from this dataset is shown in Table 4.2. Here V_F corresponds to the total number of values taken by that feature. In case of 'Holiday', Saturday, Sunday and any public holidays are considered as holidays. In cases where a certain feature's value is missing, we model it as if the feature is not available at that time using $\theta_{k,f}$. We have empirically set $\eta = \{1/K, ..1/K\}, \omega_k = \{50/K, ...50/K\}, \delta = \{1, 10\}, \lambda = \{0.01, ...0.01\}$ in our experiments [103] (K is the number of hidden context states). All our results (implemented in Python) are generated on an Intel 4-core i7-2600 CPU @ 3.40GHz, with 8 GB of RAM. We did not consider a larger dataset as our main idea is to apply collaborative filtering across the user's *closely* related users such as labmates, roommates, etc. Also, we could not find larger datasets with similar features.

Perplexity vs. K and Optimal Group Selection: Fig. 4.3(a) shows the averaged perplexity values for different number of hidden states, K, for one user case. Since this is a one user case it corresponds to *HPContext*. The perplexity is calculated by predicting the observations at the next 12 time instants in the test data given the preceding 12 observations on the same day. This has been done over all the days in the three week test data and the results are averaged. From Fig. 4.3(a), we can observe that perplexity reduces as K increases with a pattern of diminishing returns. We can observe that user A has the best perplexity and $K = 10$ provides a good balance between perplexity and complexity introduced due to higher K values. Fig. 4.3(b) shows similar result for 2 users. Since we have 2 users, we have a total of 10-user groups (5C2 combinations). We can observe that Group 10 consisting of users A, C has the best perplexity and $K = 15$ provides a good tradeoff. This means that the two users in Group 10 have more similar patterns than the users in other groups. Fig. 4.4(a) shows similar result for 3 users. We can observe that Group 2 (users A, C, D) has the best perplexity with $K = 25$ providing good tradeoff. Similarly, for 4 user case, we can observe from Fig. 4.4(b), that *Group* 1 (users A, B, C, D) has best perplexity values and $K = 25$ provides a good balance. For all user case in the same figure, we can see that $K = 25$ provides a good tradeoff. *Hence these results can be used to select most related users in a group of 2/3/4, etc. Even though it takes time to find the optimal group via this approach, we feel it is acceptable as it is done only once offline.* For the results below, whenever a user or a group of users is mentioned, we considered the above *best* groups and optimal K values. To illustrate the benefits of collaborative filtering contexts, we considered only 2 and 3 user groups as benefits diminish with increase in group size.

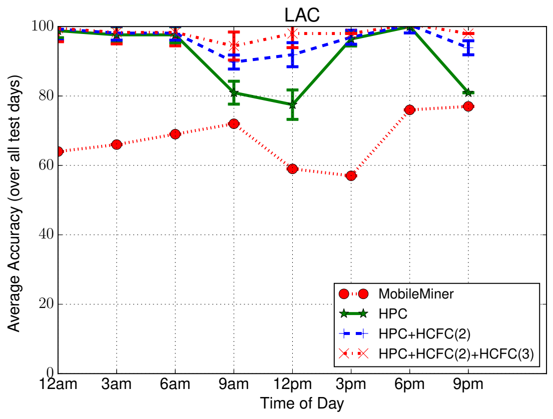
Log Likelihood Convergence, Training Times: Fig. 4.5(a) shows the log likelihood (LL) of the training data, $P(\boldsymbol{O} \mid \boldsymbol{\Psi})$, versus the iteration number in the EM algorithm (which is used to estimate parameters of *HCFContext*) for different models. We have considered six weeks of training data so, $T = 6 \times 7 \times 24$. We can see that the LLs have converged and the model significantly improves the initial LL values (up to 30%). We can notice that LLs have approximately converged after $n_{con} = 3$ iterations. Even then, the converged LLs have lower values due to large number of values for certain features (such as WiFi APs which has about 440 unique values) which makes the value probabilities, $\phi_{k,f,v}$, very small. In fact, we encountered underflow problem due to multiplication of several small probabilities and then using such a value in the denominator resulting in *nan* values. In order to solve this problem, we have used scaling approach [104] for α, β values. Fig. 4.5(a) also shows the time taken in seconds vs. iterations of EM algorithm. The corresponding time for $n_{con} = 3$ is about $200,\ 800,\ 1500$ seconds respectively for the three models. *These durations are reasonable considering that the training is run offline and very less often.* Both these results are averaged over 4 runs and we can



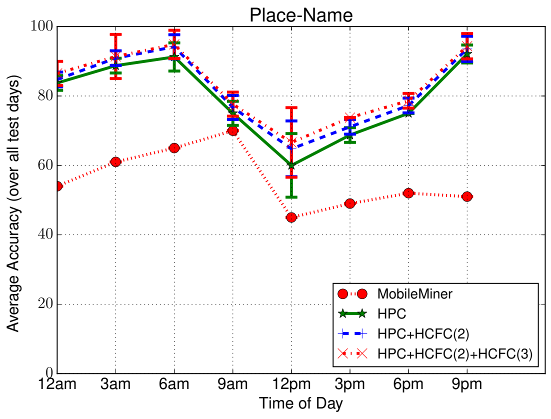
(a)



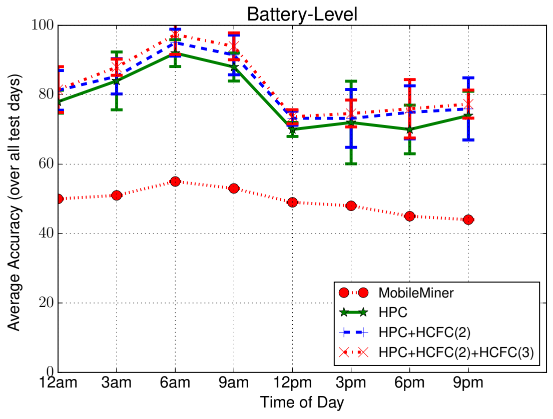
(b)



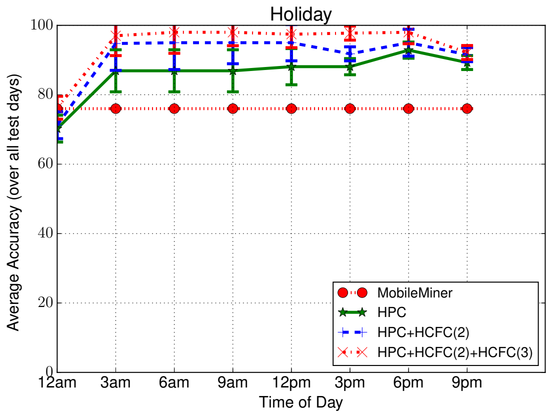
(c)



(d)



(e)



(f)

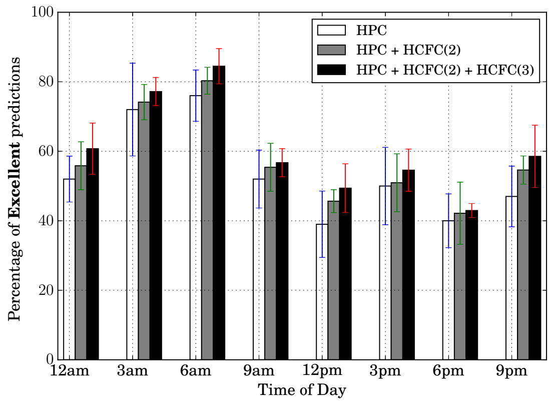
Figure 4.6: Average accuracy of the proposed models—*HPContext*, *HPContext* + *HCFContext* (2), *HPContext* + *HCFContext* (2) + *HCFContext* (3) in predicting the respective features at different times in a day (averaged over all 21 test days/4 runs). notice that the 95% confidence intervals are too minute to be noticed.

Table 4.3: Use case illustrating the predictions (with corresponding maximum probabilities) at 1 pm on one Tuesday in the test period.

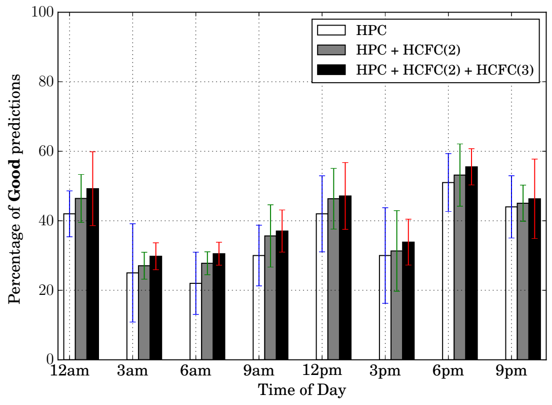
Features	**Ground Truth(A)**	**HPC(A)**	**HCFC(A,C)**	**HCFC(A,C,D)**	**HPC(A) + HCFC(A,C)**	**HPC(A) + HCFC(A,C) + HCFC(A,C,D)**
Wi-Fi AP	00:bl:f2:9b:05:76	**00:og:1f:2e:4n:6c (0.32)**	00:bl:f2:9b:05:76 (0.43)	00:bl:f2:9b:05:76 (0.41)	00:bl:f2:9b:05:76 (0.27)	00:bl:f2:9b:05:76 (0.32)
Place Name	F007	F007 (0.84)	**B003 (0.48)**	**J023 (0.31)**	F007 (0.57)	F007 (0.41)
Cell ID	42534164	42534164 (0.62)	42534164 (0.44)	**48759836 (0.3)**	42534164 (0.53)	42534164 (0.39)
LAC	8513	**9353 (0.32)**	8513 (0.47)	8513 (0.57)	**9353 (0.36)**	8513 (0.41)
Battery Level	High	High (0.89)	**Medium (0.41)**	**Medium (0.32)**	High (0.57)	High (0.42)
Battery Status	Discharging	Discharging (0.95)	Discharging (0.93)	Discharging (0.81)	Discharging (0.94)	Discharging (0.89)
Day Period	Afternoon	Afternoon (0.72)	Afternoon (0.82)	Afternoon (0.73)	Afternoon (0.77)	Afternoon (0.75)
Day Name	Tuesday	Tuesday (0.59)	Tuesday (0.68)	Tuesday (0.73)	Tuesday (0.63)	Tuesday (0.67)
Holiday	No	**Yes (0.6)**	No (0.7)	No (0.75)	No (0.55)	No (0.62)

Prediction Performance—A Use Case: We relate to the "lunch" use case mentioned in Sect. 4.1. We considered user A for this purpose and his most related 2 and 3 user groups as found from above—(A, C) and (A, C, D)—and the corresponding models, *HP-Context*, *HCFContext* (2), and *HCFContext* (3), respectively (please note this notation to be used in rest of the chapter). We first illustrate the prediction results using a known use case as follows. We manually observed from the data that users A, C, D usually go to lunch together on weekdays around $12-2$ pm. We wanted to check whether our models are able to capture this group behavior. Hence we predicted the contextual feature-values of user A at 1 pm on one randomly selected weekday (Tuesday) in the test period given the observations of previous one-day duration using the above models. Table 4.3 shows the results of different models along with ground truth (column 2). All entries correspond to maximum probability values with those probabilities shown in brackets. Incorrect predictions are depicted in bold. We can notice that *HPContext* (col. 3) does a good job in predicting the personalized features such as *Place Name*, *Battery Level* but makes 3 incorrect predictions for more general features such as *LAC*, *Holiday*, etc. Note that each feature is predicted independently, e.g., predicting *Place Name* correctly does not necessarily mean *Wi-Fi AP* prediction is correct. Interestingly, *HCFContext* (cols. 4,5) makes correct predictions for those general features but fares badly for personalized features. Hence we combined the two models to obtain better predictions (as can be seen in last two columns) as follows. In case of *HPContext* + *HCFContext* (2), we first average the probabilities of values predicted by both models and then take the feature value with the highest average probability. We do similarly for $HPC + HCFC(2) + HCFC(3)$ (here *HPC* refers to *HPContext* and *HCFC* to *HCFContext* for simplicity).

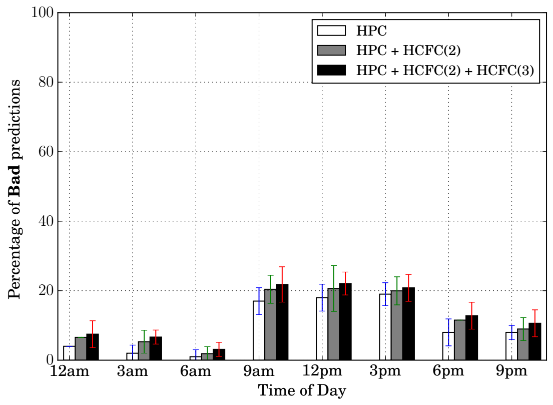
Prediction Performance—Overall: To evaluate the overall performance, we predicted all the contextual feature value pairs of user A at 3 hour increments in the entire 3-week testing period (i.e., $3 \times 7 \times 8$ in total) given the past observations of one-day duration. In order to compare with other closest approaches, we considered Mobile Miner [73], which is the current state-of-the-art machine learning algorithm to mine contextual co-occurrences. Each feature is considered to be independently co-occurring with the time of day and day of week (as opposed to sequentially occurring in our case),



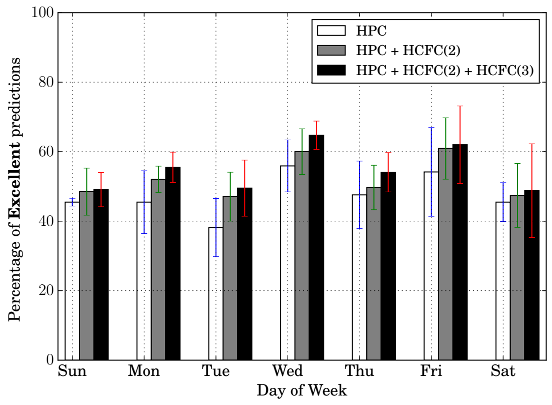
(a)



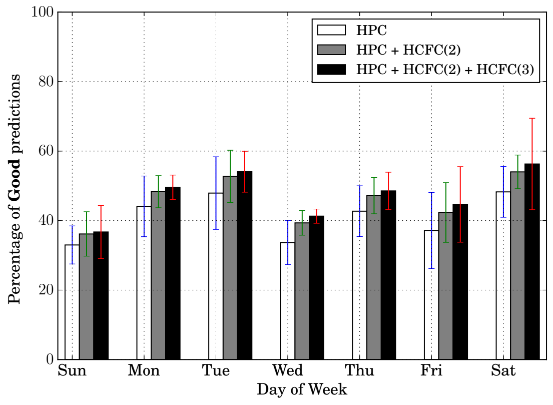
(b)



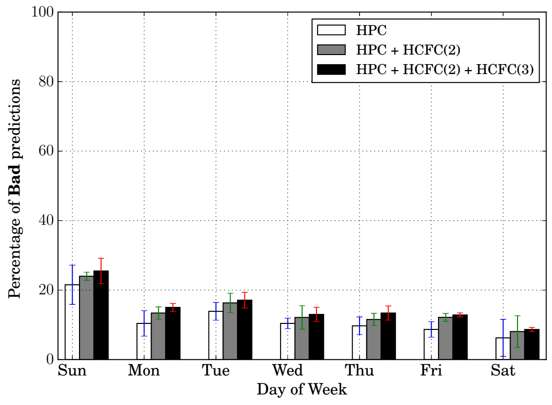
(c)



(d)



(e)



(f)

Figure 4.7: Percentage of excellent/good/bad predictions—(a)-(c) at different times of day; (d)-(f) for different days of week.

and is modeled using Multinomial Logistic Regression. Fig. 4.6 shows the average prediction accuracy (percentage of correct predictions) for each feature at different times in a day (averaged over all the 21 days). Even though we are able to predict at the hourly level (due to the upsampling mentioned earlier), we show only at 3-hour intervals for clarity. These simulations are also run for 4 runs (to account for randomicities such as random initialization of the model parameters) and the confidence intervals are shown. We have shown results only for six features consisting of four location features—Wi-Fi AP, Place Name, Cell ID, Location Area Code (LAC), one device feature—Battery Level and one time feature—Holiday. Others follow similar pattern.

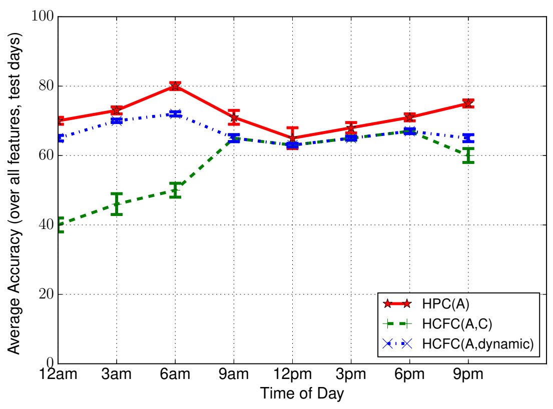
First of all we can notice that the proposed models perform better than Mobile Miner. Second, in case of proposed models, we can notice that the accuracy slightly drops during mid-day compared to other times. The reasons for this drop as follows—(1) the user is more mobile during those times introducing randomness into the data making the model hard to learn; (2) the average number of places visited by each user is 270 (as indicated above) and the K value chosen (as a tradeoff between complexity and accuracy) is less than that; (3) the data is down-sampled to one hour interval, meaning, any places with stay duration less than one hour will not be captured well. However it is important to note that the accuracy in such cases is still on an average about 75% which is reasonable to validate/complement the context from sensors. During other times of the day, we can notice that the accuracy is about 90%, which means the models are able to predict the values of those contextual features correctly 90% of the time. Third, we can notice that $HPC + HCFC(2)$ model improves accuracy over HPC to a maximum of 15% especially in more general features such as *LAC*, *Holiday*. In case of personalized features such as *Place Name*, *Battery Level*, the improvement is minor. We also notice that the additional improvement from $HCFC(3)$ is again minor, about 5%. In total, $HCFC$ contributes upto 20% improvement in accuracy. We have also plotted similar results for each day of the week but could not include them due to space limitations. In addition to above trends in generic/personalized features vs. models, we have noticed a drop in accuracy over weekends (to around 75% on average) again owing to increased mobility with less sequential behavior. The average accuracy is about 80% to 90% for the rest of the days ($HPC + HCFC(2) + HCFC(3)$ model).

Prediction Quality: Next, we tested our models' prediction quality by testing how many of the features (all in Table 4.2) the models are able to predict correctly at a given time of day. For this purpose, we created three categories—*Excellent*, *Good*, *Bad*. If the model is able to predict at least 7 features out of 9 correctly at a given instant, we call it *Excellent* prediction. Similarly we call 4/5/6 features prediction, a *Good* prediction and 1/2/3 (0 is not included) features prediction, a *Bad* prediction. For example, the prediction corresponding to $HPC(A)$ in Table 4.3 is considered a *Good* prediction, while that belonging to $HPC(A) + HCFC(A, C)$, an *Excellent* prediction. Fig. 4.7 shows these results (averaged) for different times of day and days of week. In both sets of figures, we can notice that the percentage of *Excellent* cases is at least 50%. Secondly, the percentage of *Excellent* cases is more than *Good* cases which in turn is more than the *Bad* cases (in particular the *Bad* cases are very less, comparatively speaking).

Test Data Rotation: The performance of the models when the test data chosen is the first (1-3), middle (4-6) and last (7-9) three weeks is shown in Fig. 4.5(b), which shows the percentage of correct predictions across all features and test days for different choices of test data (results shown only for *HPC* + *HCFC*(2) + *HCFC*(3) for clarity). We can notice that the performance is roughly the same showing robustness of the proposed models to choice of test data and their ability to fully learn users' sequential patterns using 6 weeks train data.

Contextual Optimal User Group Selection: So far, for a given user, A, we have found the optimal user groups considering all times of the day. However, it is more beneficial to find the optimal user group based on the time of the day which is more realistic. For example, for a given user, the closely related users during the office hours may be different from the closely related users during home hours. Taking this idea into account, we have found the most closely related user for user A (i.e., optimal 2-user group) at different times of the day using the perplexity method mentioned earlier. Instead of averaging across all times of the day, we find the user group with minimum average perplexity at each time instant of the day. The results are as follows—$(12am, B)$, $(3am, B)$, $(6am, B)$, $(9am, C)$, $(12pm, C)$, $(3pm, C)$, $(6pm, C)$, $(9pm, D)$.

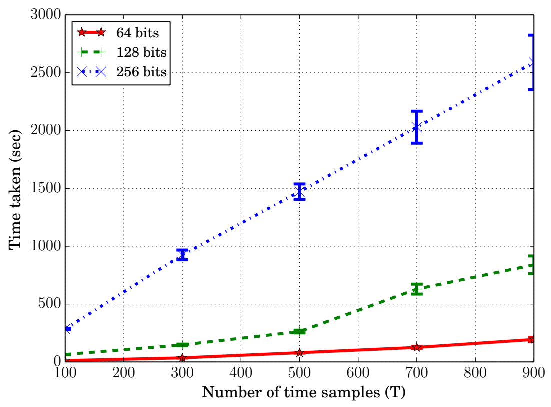
Prediction Using Only *HCFContext*: We now predict the feature-value pairs of user A at all time instants in the test period using only the collaborative filtering model, *HCFC*(2), with dynamic optimal 2-user group obtained from above. Fig. 4.8(a) shows



(a)

n (bits)	c	$P(O \mid \Psi)$	π	ρ	θ
64	10^3	5.70	6.56	9.10	14.95
128	10^3	5.10	4.76	8.26	11.84
256	10^3	4.01	3.48	6.89	10.02
512	10^3	1.83	2.95	6.05	7.23
64	10^6	0.15	1.21	4.04	7.12
128	10^6	0.13	1.05	3.09	6.04
256	10^6	0.11	0.58	1.98	5.14
512	10^6	0.11	0.45	0.97	2.50

(b)



(c)

Figure 4.8: (a) Comparison of performance among *HPContext*, *HCFContext* (with fixed optimal user group), *HCFContext* (with dynamic optimal user group). (b) Worst-case errors of parameters [10 runs] (%). (c) Time taken to run Algorithm 4.1 for different number of training (time) samples vs. key length (bits).

the performance of that model compared against two other models—the personalized model *HPC* and *HCFC*(2) with fixed optimal 2-user group (A, C). We can notice that dynamic *HCFC* performs close to *HPC* and better than fixed *HCFC*. This indicates that personalized context can be obtained from collaborative filtering of contexts corresponding to user's closely related users with appropriate dynamic (i.e., time of day/activity the user is performing) selection of closely related users.

Privacy-preserving Algorithms: To evaluate the performance of Algorithms 4.1, 4.2, we tested them on a simple HMM with $M = 2$ parties, $K = 2$ hidden states and $|\boldsymbol{f}_{tu} = 6|$ observation states per hidden state. We evaluated both the amount of error introduced (due to scaling as mentioned previously) as well as the time taken to train the HMM and run the predictions. For the former, we calculated the error by comparing with the non-privacy preserving case. We varied the key length (bits) n and the scaling factor c. The worst-case errors over 10 runs with $T = 1000$ samples, as percentages, for different parameters is shown in Fig. 4.8(b). We notice that the error reduces as scaling factor increases (as expected). Similarly, as the key length increases, the error reduces and also security increases. We can notice that for large keys and reasonable scaling factors, the error due to integer approximation and consequent over- or underflow is insignificant (about 2% in the worst case over all parameters). This shows that the effect on the prediction accuracy results above will not be drastic. However, the price is in terms of run-time. Fig.. 4.8(c) shows the average run-times of Algorithm 4.1 vs. the number of time samples as well as the key length (with $c = 10^6$). We can see that the time taken varies linearly with the number of samples used. However, the relation seems to be approximately quadratic with key-length. This result shows the tradeoff between security and run-time. As the key-length is increased, more is the security, less is the amount of error introduced, however the run-times are more. Hence a suitable key length should be chosen that is a compromise between security/error and run-time. The run-times for Algorithm 4.2 are on average five times more. *However these algorithms need to be run only for training the HCFContext which is run very less frequently and offline.* Moreover, we will leverage recent advances in encryption and multi-party computation algorithms such as [105] to help further reduce the runtimes of these algorithms, as part of future work.

Real-time Inference and Cold-start: Once the training is complete, model parameters are known to each device. Prediction, then, just involves evaluating (4.32), (4.33) by plugging in the learned parameters. These are just a few arithmetic operations and do not involve any compute-intensive encryption/decryption algorithms unlike training which happens offline. *Hence, our approach will not have any problems in practical implementations i.e., making predictions in real time.* Furthermore, to combat the cold-start problem akin to collaborative filtering based approaches, we suggest to—(i) use sensor context and also obtain user validation for additional security in case of security based applications such as Swirls [70]; (ii) use sensor context + *HPContext* until *HCFContext* is learnt well.

Energy Considerations: Note that all of the features we worked with are passive, i.e., do not require active probing that consumes energy. One exception is the Wi-Fi, which needs to be turned ON in case it is not ON. In all other cases, we piggyback on the sensor data already available on the phone, reducing energy consumption.

4.5 Summary

We proposed and evaluated (on a real-life dataset with over 80% accuracy) privacy-preserving, sequential history-based personalized and collaborative-filtering models, for current and future mobile context prediction to validate and/or enhance the sensor context. Their feasibility for practical deployment in security applications and/or mobile personal assistant technologies is shown.

Chapter 5

Deep Multi-Task Learning for Fault/Anomaly Detection Using Scalar Sensor Data

We have seen in previous chapters how to design coordinated target search and coordinated data collection in a post-disaster scenario using a group of IPSs (UAVs) via reinforcement learning based techniques. As these IPSs are operating in a highly dynamic and uncertain environment such as a post-disaster scenario, it is paramount to continuously monitor their operation and proactively detect and diagnose any anomalous behavior. As such, in this chapter, we will focus on developing anomaly detection techniques using scalar sensor data of the IPS. Specifically, we will develop techniques in the context of autonomous ground vehicles, which we will then extend to autonomous UAVs in the next chapter.

Autonomous Driving (AD) aka self-driving vehicles are cars or trucks in which human drivers are never required to take control to safely operate the vehicle. They combine sensors and software to control, navigate, and drive the vehicle. Though still in its infancy, self-driving technology is becoming increasingly common and could radically transform our transportation system economy and society. Many thousands of people die in motor vehicle crashes every year in the United States (more than 30,000 in 2015); self-driving vehicles could, hypothetically, reduce that number—software could prove to be less error-prone than humans. Based on automaker and technology company estimates, level 4 (the car is fully-autonomous in some driving scenarios, though not all) self-driving cars could be for sale in the next several years [106].

5.1 Introduction

Motivation: Currently, safety is an overarching concern in AD technology, that is preventing its full deployment in real world. The question to answer is whether the AD system can handle potentially dangerous and anomalous driving situations. Before AD systems can be fully deployed, they need to know how to handle such scenarios, which in turn calls for heavy training of an AD system on such scenarios. The challenge lies in that, these scenarios are very rare. They constitute the 'long tail of rare events' and comprise less than 0.01% of all events in a given driving dataset. Hence Artificial Intelligence (AI) technology need to be used to both mine these "gems" in a given dataset and then to train the AD system to handle such 'special' situations. Detecting such anomalous driving scenarios that is crucial to building fail-safe AD systems offers two advantages—both offline and online. In the former, given a dataset, the identified anomalous driving scenarios can be used to train an AD system to better handle such scenarios. This can be achieved, for example, via weighted training—give more weight to learning anomalous scenarios than normal scenarios. For online purposes, detecting anomalous driving scenarios ahead of time can help prevent accidents in some cases, by taking a corrective action so as to steer the system in a safe direction (e.g., apply appropriate control signals or if possible, handing over the control to a human driver). We specifically consider only the Controller Area Network (CAN) bus sensor data such as pedal pressure, steer angle, etc. (multi-modal time-series data) due to its simplicity, while still providing valuable (though not complete) information about the driving profile; augmenting with video data will be our future work. We consider any unusual pattern (such as abruptness, rarity, etc.) among different modalities as a sign of an anomaly. Such a pattern could have happened due to unusual reaction of driver on the pedal, accelerator, steering wheel, etc. which in turn implies the driver has gone through a challenging (anomalous) driving situation. Though model-based (rule-based) approaches can be used to detect anomalies in multi-modal time-series data, they work reliably only for simple cases such as threshold based anomaly detection (speed/deceleration greater than a threshold, etc.). It is difficult as well as tedious

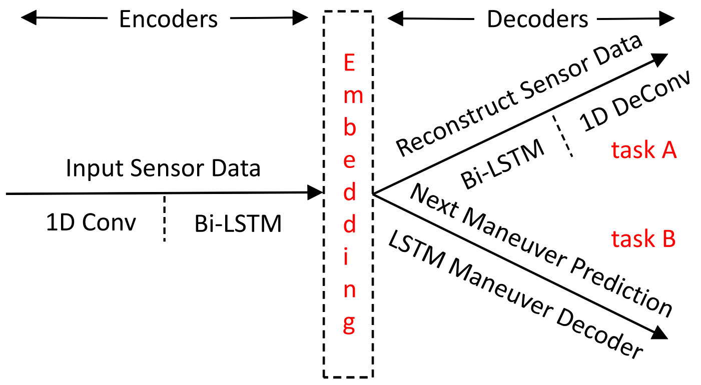


Figure 5.1: A high-level overview of the proposed multi-task deep learning based approach for anomaly detection in multi-modal time-series driving dataset.

to compose rules for complex and even unknown (apriori) situations. On the other hand, data-driven approaches, can learn representations directly from the data, and use them to detect anomalies. This gives them the ability to detect complex and unknown anomalies directly from data. Although, the performance of data-driven approaches is only as good as the data, this limitation can be addressed to some extent, when large amount of data is considered for training. In data-driven approaches, deep-learning based approaches (as opposed to classical machine learning techniques) are especially interesting due to their ability to learn the features on their own without the need for expert domain-expertise. Existing deep learning approaches for anomaly detection in multi-modal time series data include reconstruction-error based approaches such as Long Short Term Memory (LSTM) autoencoders. These approaches do not perform well in the case where multiple "normal" situations (multiple positive classes) exist with "class imbalance" problem. In case of driving data, these classes correspond to right-turn, going-straight, U-turn, etc. where the data for U-turn is far lesser than the data for going-straight. There is a higher chance that the classifier in these approaches over-fits smaller (less-frequent) classes resulting in poor performance. Since reconstruction error is used as a measure of anomaly by these approaches, they classify the less frequent normal classes (e.g., U-turn) also as anomalous, further degrading the algorithm performance for anomaly detection.

Our Approach: We make the observation that, while reconstruction-error based approaches perform poorly with rare but non-anomalous events, their performance can be greatly improved with the help of simple domain-knowledge (availability of maneuver labels in our case). Leveraging these maneuver labels, we add a symbol predictor to the autoencoder system (creating a multi-task learning system) which acts as a regularizer to the autoencoder, thereby achieving better performance than a standalone autoencoder system. The proposed deep multi-task learning based anomaly detection system is shown in Fig. 5.1. The two tasks in the proposed approach are a convolutional bi-directional LSTM (bi-LSTM) autoencoder and a convolutional bi-LSTM sequence-to-sequence (seq2seq) symbol predictor (in contrast to simple LSTM predictor that predicts raw sensor data rather than symbols). In the seq2seq predictor, the predicted symbols/labels correspond to automobile's next series of maneuvers (e.g., going-straight, left-turn, etc.). These labels are obtained from manually annotated driving data. We show that the proposed multi-task learning approach performs better than existing deep learning based anomaly detection approaches such as LSTM autoencoder and LSTM predictor as one task acts as a regularizer for the other. In addition to reconstructing the input data (via autoencoder), the network is also constrained to predict the next series of maneuvers (via symbol predictor) and as such the chance of overfitting is reduced. This is because, in a usual autoencoder system, there is a possibility of not learning good representations of the input data and blindly reproduce the input. In our system, such risk is reduced (i.e., the system learns good representations) because it is forced to also predict the future, hence it cannot blindly memorize the input. Such a regularizer system also helps solve the problem of overfitting to smaller classes mentioned above. Secondly, the proposed multi-task learning approach leverages these maneuver labels to define a custom anomaly metric (rather than simple reconstruction error) that weighs down detection of rare but non-anomalous patterns such as U-turns as anomalies. The approach has been tested on 150 hours of raw driving data [7] in and around Mountain View, California, USA and is shown to perform better than state-of-the-art approach, LSTM-based autoencoder [107].

Main contributions can be summarized as follows.

- We propose a novel multi-task learning (convolutional BiLSTM autoencoder and symbol predictor) approach for detecting anomalous driving with multiple "normal" classes and a class imbalance problem. Our approach leverages simple domain-knowledge (manuever labels) to build a regularizer system that reduces overfitting and enhances overall reconstruction performance.

- We propose an anomaly scoring metric that leverages the frequency of maneuver labels from the training data to reduce the cases where rare, but non-anomalous, events are classified as anomalies.
- We evaluated our approach both quantitatively and qualitatively on 150 hours of real driving data and compare it with state-of-the-art LSTM autoencoder and multi-class LSTM autoencoder approaches to show its advantages over them.

Chapter Outline: In Sect. 5.2, we position our work with respect to state of the art; in Sect. 5.3, we present our approach, and in Sect. 5.4, we evaluate it both quantitatively and qualitatively on real driving data. Finally, in Sect. 5.5, we summarize the chapter.

5.2 Related Work

In this section, we describe important related work in the domain of anomaly detection for multi-modal/multi-variate time series data. Anomaly detection is generally an unsupervised machine learning (ML) technique due to lack of sufficient examples for the anomalous class. Within unsupervised learning, it can be broadly classified into the following categories—contextual anomaly detection, ensemble based methods and finally deep learning approaches. These methods internally use statistical/regression/rule based approaches, dimensionality reduction, distribution based approaches. In statistical/rule-based approaches, features are generally hand-made from the data such as mean, variance, entropy, energy, etc. Certain statistical tests/formal rule checking actions are performed on these features to determine if the data is anomalous. In dimensionality reduction, the data is projected onto a low-dimensional representation (such as princial components in Principal Component Analysis (PCA)). The idea is that, this low-dimensional representation captures the most important features of the input data. Then clustering techniques such as k-means or Gaussian Mixture

Models (GMMs) are used to cluster these low-dimensional features to identify anomalies. In distribution-based approaches, the training data is fit to a distribution (such as multi-variate gaussian distribution or a mixture of them). Then given a test point, distance is calculated of this test point from the fitted distribution, for example, using Mahalanobis distance that represents the measure of anomaly.

Contextual anomaly detection: An anomaly may not be considered anomaly when the context under which it happens is well-known. For example, the CANbus sensor data of a car may look anomalous when the car is taking a U-turn, which is not considered an anomaly. This is also called seasonal anomaly detection in other domains such as building energy consumption, retail sales, etc. Hayes et al. [108] and Capozzoli et al. [109] present a two-step approach for contextual anomaly detection. In the former, in step 1, only the sensor's past data is used to identify anomalous behavior. For this it uses univariate Gaussian function. Later in step 2, if the output of step 1 is found to be anomalous, then it passes to step 2 to check if it is contextually anomalous or not. Twitter [110] recently published a seasonal anomaly detection framework based on Seasonal Hybrid Extreme Studentized Deviate test (S-H-ESD). Netflix [111] recently released an approach for anomaly detection in big data using Robust PCA (RPCA). Even though Netflix's approach seemed successful, their statistical approach relies on high dimensionality datasets to compute a low rank approximation which limits its applicability. This is because PCA makes the assumption that the lower-level representation has a linear relation with the input while there is a possibility of non-linear relationships as well. Finally, Toledano et al. [112] propose a bank filter and fast autocorrelation based approach for large scale time-series data considering seasonal variations.

Ensemble based methods: In ensemble learning, different models are trained on the same data (or random sets of samples from the original data) and a majority voting (or another fusion technique) is used to decide the final output. Another advantage of ensemble learning is that the member models are chosen such that they are complementary to each other in terms of their strengths/weaknesses, i.e., the weaknesses of one are compensated by the strengths of the other. For example, Araya et al. [113], proposed an ensemble based collective and contextual anomaly detection framework. The ensemble consisted of pattern recognition algorithms such as Autoencoder and PCA, as well as prediction based anomaly detectors such as Support Vector Regression (SVR) and Random Forest. They showed that the ensemble classifier is able to perform well compared to the base classifiers.

Deep learning methods: In deep learning techniques, the features are generally learned by the classifier itself, so there is no need to hand-engineer these features. The techniques within this can be broadly classified into two categories: (i) Representation learning for reconstruction: Here the input data is mapped to a latent space (generally lower dimension than input data) using an encoder and then the latent space is remapped to input space using a decoder. The latent space captures a representation of the input data similar to PCA. The reconstruction error at the end of this process is a measure of anomaly. Autoencoders [114] are prime examples in this category. For example, Malhotra et al. [107] present an LSTM based encoder-decoder approach for multi-sensor time-series anomaly detection. The approach has been tested on multiple datasets including power demand, space shuttle valve, medical cardiac data and a proprietary engine data and showed promising results. (ii) Predictive modeling: Here the current/future data is predicted from past data using LSTM modules that capture long term temporal behavior. The prediction error is a measure of anomaly. LSTM sequence predictors are examples in this category. For example, Taylor et al. [115] proposed an LSTM predictor based anomaly detection framework for automobiles based on Controller Area Network (CAN) bus data of an automobile similar to ours. Hallac et al. [116] present an embedding approach for drive2vec which can be used to encode the identity of the driver. However this approach only complements ours, as our approach can work both with raw data as well as embedded data. Malhotra et al. [117] proposed an LSTM based predictor for anomaly in time series data that is shown to perform well on four kinds of datasets mentioned above.

In contrast to these approaches, we propose a multi-task deep learning based approach, that overcomes the shortcomings on (i) and (ii) by— incorporating a built-in regularizer (as one task acts as regularizer for the other) and leveraging domain knowledge (such that rare but non-anomalous maneuvers such as U-turns are not classified

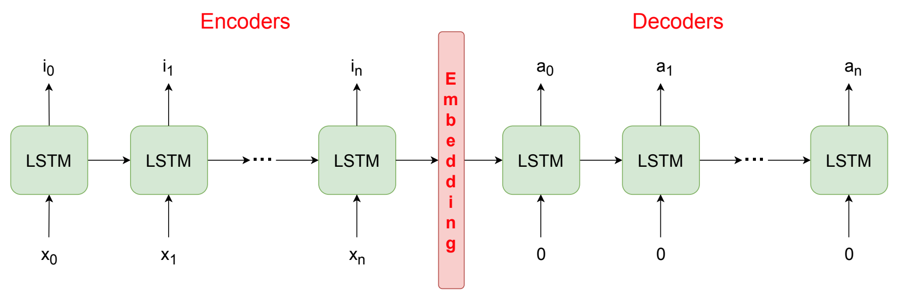


Figure 5.2: LSTM Autoencoder—the encoder cells encode the input data into a representation that is stored in the cell state of the last encoder LSTM cell. The decoder cells take it as input and try to generate the time series data.

as anomalies).

5.3 Proposed Solution

In this section, we first explain the LSTM autoencoder (reconstruction-error) based approach [107] which is currently the best performing (unsupervised) anomaly detection framework for multi-modal time-series data. We then present our semi-supervised approach for anomaly detection in driving data which leverages the maneuver labels to improve the performance. Anomaly detection using unsupervised learning consists of two steps. In step 1, the system is trained with several normal examples to learn representations of the input data e.g., GMM clustering. Because we are dealing with temporal data, a sliding window approach needs to be adopted to learn these representations. In step 2, given a test data point, we define an anomaly score based on the learned representations, e.g., distance from the mean of the cluster.

LSTM Autoencoder (Existing Approach): Fig. 5.2 shows the LSTM autoencoder high-level architecture. Input time series data, $\{x_0, x_1, ...x_n\}$, of size $n+1$ (corresponding to one window of data segmented from full data) is fed to the encoders which consist of $n+1$ LSTM cells. Each LSTM cell encodes its input and the cell state from previous cell into its own cell state, which is passed onto the next LSTM cell. Finally, the cell state of the last LSTM cell has the encoded representation—which we call *embedding*—of all the input data $\mathbf{x}$. The size of this embedding is equal to the number of units (also called hidden size) in the last LSTM cell. The decoders similarly consist of

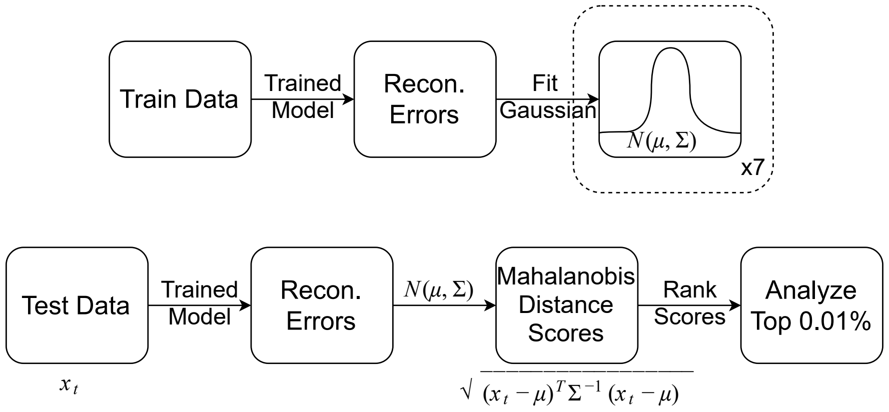
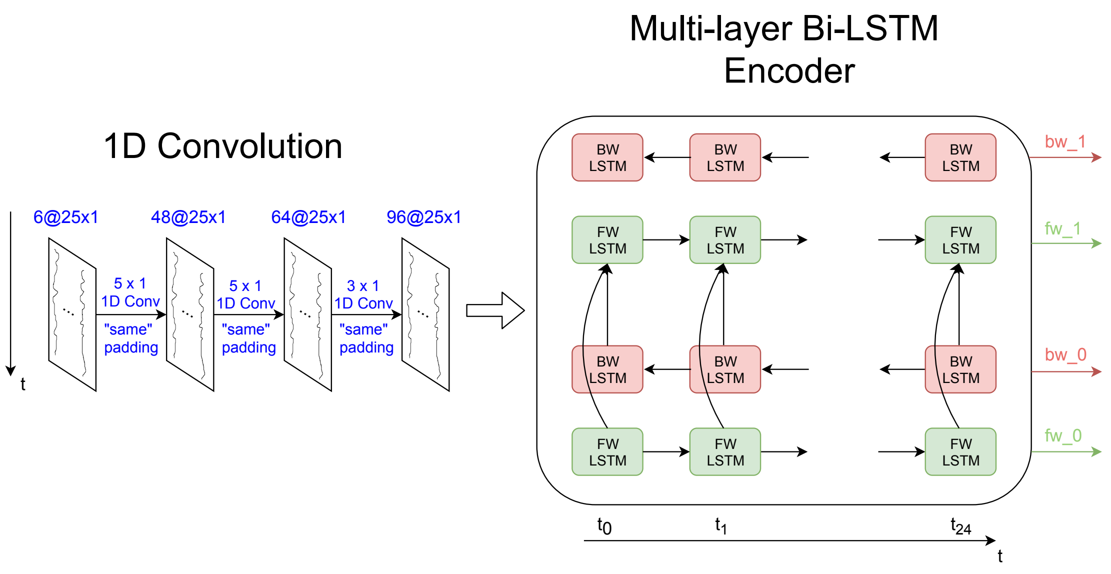


Figure 5.3: (Top) After the model is trained, we fit the reconstruction errors to a multivariate gaussian model; (Bottom) Given a test data point, we first find the reconstruction error and then find the Mahalanobis distance/score of this point with respect to the fitted gaussian distribution. Top scores can then be analyzed as per requirements.

a series of LSTM cells, however the input to these decoders is given as zero as the goal is to regenerate the input data. Another approach of feeding the output of the previous cell (a_i) as input to the next cell is also possible. The first decoder LSTM cell takes the embedding as one of the inputs (the other input being zero) and passes on its cell state to the next decoder cell. The process is repeated for $n+1$ time steps. During each step, the LSTM cell generates an output a_i, finally resulting in $\{a_0, a_1, ...a_n\}$ after $n+1$ steps. The network is trained by minimizing the difference, $|\mathbf{x}-\mathbf{a}|^2$ using stochastic gradient descent and backpropagation. After sufficient (successful) training, the network is able to learn good representations of the input data stored in its embedding, which completes step 1. The network is then able to reconstruct new data very well i.e., with lower reconstruction error, as long as it has seen similar pattern data during training. However, when the network is fed with data that has completely different pattern than is used during training, there will be a large reconstruction error.

Though reconstruction error can directly be used as a measure of anomaly for step 2, better results can be achieved, with further processing. The method currently adopted [107] is shown in Fig. 5.3. After the network is trained, the train data is again fed to the trained network to capture the reconstruction errors. These errors are then fit to a multivariate Gaussian distribution as shown in Fig. 5.3. The intuition behind this is that the reconstruction errors comprise a mixture of multiple Gaussians each corresponding to one of the maneuver classes. Hence, better results can be obtained by clustering these reconstruction errors using a multivariate Gaussian or a Gaussian Mixture Model (GMM), so the anomalous points can be easily identified using a distance metric. Therefore, given a test data point, the reconstruction error is first calculated using the trained model. Mahalanobis distance of the error is then calculated with respect to the fitted gaussian model using the formula shown in Fig. 5.3. These distances, which are considered anomaly scores are then sorted in decreasing order and analyzed as per requirements e.g., analyze top 0.01%.

Multi-task Learning (Proposed Approach): As mentioned earlier, fully unsupervised reconstruction error based approaches such as LSTM autoencoder fare poorly when there are rare occurring positive (non-anomalous) classes in the data. For such relatively rare cases, the network is unable to extract representations as it has not seen such data sufficiently during training, thereby producing large reconstruction error. We solve this problem, by designing a semi-supervised multi-task learning framework that leverages driving maneuver labels as shown in Fig. 5.1. Here task A is the autoencoder, while task B is a symbol/maneuver predictor. Task B acts as a regularizer to the autoencoder as the overall network is also constrained to predict the next series of maneuvers apart from reconstructing the input data. For this to be possible, better representations need to learned by the network that can help in both reconstruction and prediction. This combined (multi-task) system performs better reconstruction than a standalone autoencoder. Similarly, autoencoder system (task A) acts as a regularizer for symbol predictor (task B). This is because autoencoder helps in learning good representations of the input data. These representations can then be used by the symbol predictor to predict next series of maneuvers. Hence, in a similar way as mentioned above, the combined system produces a better symbol predictor than a standalone symbol predictor. Both these are possible as both tasks mutually help each other. Our approach is semi-supervised as we make use of maneuver labels to design a regularizer in Task B, but is not supervised as we do not have anomaly and non-anomaly labels. We will now explain the encoder and the decoders of both tasks in detail.



1D Convolution
Multi-layer Bi-LSTM Encoder
6@25x1
48@25x1
64@25x1
96@25x1
5 x 1 1D Conv "same" padding
5 x 1 1D Conv "same" padding
3 x 1 1D Conv "same" padding
t
BW LSTM
FW LSTM
bw_1
fw_1
bw_0
fw_0
t0
t1
t24
t

Figure 5.4: Convolutional and bi-LSTM encoder of the proposed multi-tasking learning framework in Fig. 5.2.

Convolutional bi-LSTM Encoder: The basic encoder in an LSTM autoencoder (Fig. 5.2) does not perform sufficiently well as it not does take into account: (i) inter channel correlations (ii) directionality of data. We design an encoder that addresses these issues as shown in Fig. 5.4. It consists of a series of 1-dimensional (1D) convolutional layers followed by bi-directional LSTM layers. The convolutional layers help in capturing inter-channel spatial correlations, while the LSTM layers help in capturing inter- and intra-channel temporal correlations. Unidirectional LSTM layers capture temporal patterns only in one-direction, while the data might exhibit interesting patterns in both directions. Hence to capture these patterns, we have a second set of LSTM cells for which the data is fed in the reverse order. Further, we have multiple layers of these bi-directional LSTM (bi-LSTM) layers to extract more hierarchical information. All the data that has been processed through multiple convolutional and bi-LSTM layers is available in the cell states of final LSTM cells. This is the output of the encoder which will be fed as input to the decoder tasks.

Decoder (Autoencoder, Task A): The decoder in autoencoder (task A) performs encoder operations in reverse order so as to reconstruct the input data (Fig. 5.5). It first consists of bi-LSTM layers which take the final cell states from encoder as one of the

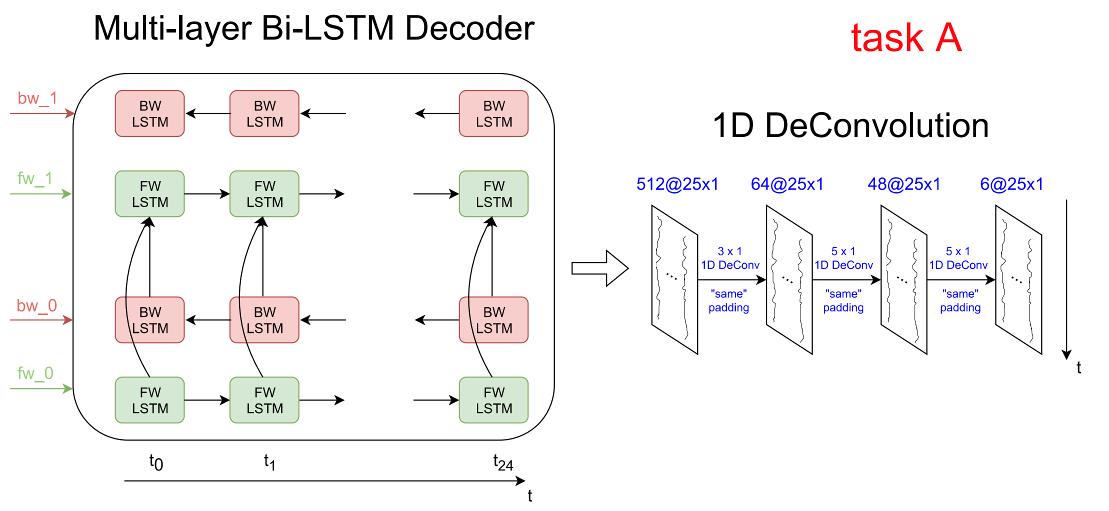


Figure 5.5: Convolutional bi-LSTM decoder for task A (autoencoder) in Fig. 5.1. Here *fw* and *bw* refer to forward and backward respectively

inputs (the other input being zero). As mentioned previously, the other input (other than the previous cell state) can be either zero or the output of the previous LSTM cell. The outputs of LSTM layers are fed as input to a series of 1D de-convolutional layers which approximate the reverse of convolution (also called *transposed convolution*) to generate data with same shape as that of input data to encoder.

Decoder (Predictor, Task B): The decoder of the symbol predictor (task B) is shown in Fig. 5.6. It takes only forward cell states from encoder as it has only uni-directional (forward) LSTM layers. It adopts a greedy decoder, where the most probable symbol output of the previous LSTM cell is fed as input to the next LSTM cell. The first LSTM cell takes a special symbol <SOS>, denoting start of sequence, as input. Likewise, the last LSTM cell generates <EOS> symbol, denoting end of sequence. The output symbols of all the LSTM cells is the predicted series of next maneuvers (Left Turn, Left Turn, ... Left Turn in Fig. 5.6).

Training (Step 1): The loss function for Task A is the Mean Square Error (MSE) between the input data to encoder and the output of decoder. The loss function for Task B is weighted cross-entropy loss with weights being the inverse of the frequency of maneuvers in the train data. That is, the weight for symbol, $s = w_s = 1/f_s^k$, where f_s is

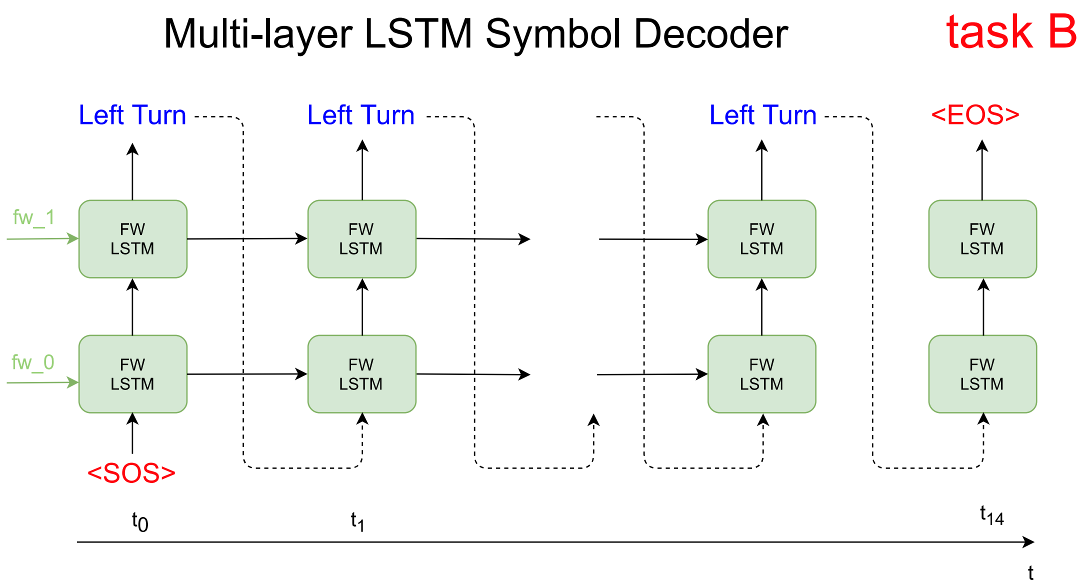


Figure 5.6: Greedy symbol decoder for task B (maneuver predictor) in Fig. 5.1.

the frequency ratio of maneuver s in the train data and k is determined empirically for best results. The overall network is trained by minimizing the weighted losses of task A, task B and regularization losses. That is, the overall loss is $L_O = w_A L_A + w_B L_B + w_R L_R$, where w_A, w_B , w_R are the weights for L_A, L_B, L_R, the task A, task B and regularization loss respectively. The weights can be optimized in an empirical manner by searching the parameter space exhaustively. To limit the search time, quantization can be adopted.

Inference (Step 2): During inference, given a test data point, an anomaly score is calculated as mentioned in Fig. 5.3. This anomaly score (say a_i), however fares

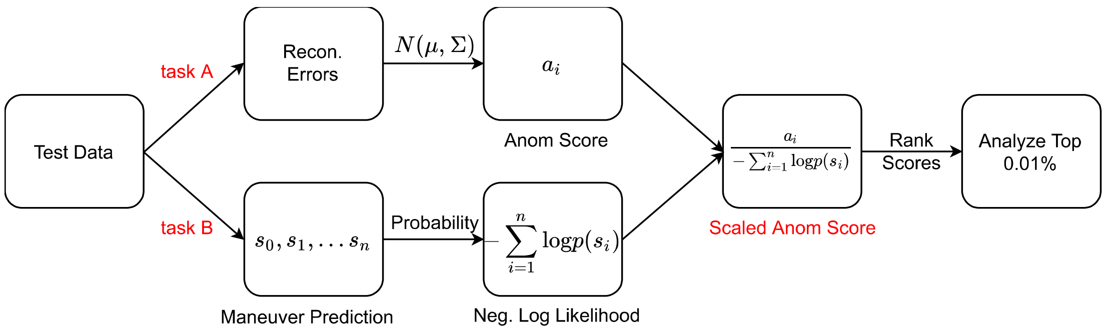


Figure 5.7: Scaled anomaly scores that leverages the maneuver predictions of task B to reduce number of false positives compared to the scores in Fig. 5.3.

poorly with rare positive classes leading to multiple false positives. In order to address this problem, we define a new anomaly score leveraging the predicted maneuvers from task B as shown in Fig. 5.7. Assume $s_0, s_1, ...s_n$ correspond to the maneuvers predicted by task B. We then calculate the negative log-likelihood of such a sequence using $-\sum_{i=1}^{n}\log[p(s_i)]$ (we assume independence for simplicity). Note that the i in the above expression is a running variable and is not related to a_i in Fig. 5.3. This calculated value is low for more frequent maneuvers (e.g., going-straight) and high for rare maneuvers (e.g., U-turns). We divide a_i with this value to obtain the scaled anomaly score. This is high for more-frequent maneuvers and low for less-frequent maneuvers such as U-turns. This can also be interpreted in a Bayesian manner as follows: the inverse of the negative log-likelihood can be considered the prior, while a_i can be considered as the conditional/likelihood, making the scaled anomaly score the posterior. In this way, rare but non-anomalous situations are weighed down leading to lesser false positives.

5.4 Performance Evaluation

In this section, we first explain the experimental setup (data and training) then present quantitative and qualitative results for two scenarios—comparison with unsupervised LSTM autoencoder (without using the information of maneuver labels) and semi-supervised multi-class LSTM autoencoder (that uses the information of maneuver labels).

Dataset description: We evaluated our approach on a 150 hours HDD driving dataset [7], which is collected from February 2017 to March 2018, predominantly during day-time. A screenshot of the driving scene as taken from the driving vehicle can be seen in Fig. 5.8. The data consists of Controller Area Network (CAN) bus data that has information about six driving modalities—steer angle, steer speed, speed, yaw, pedal angle and pedal pressure. The data has been downsampled to 5 Hz from the original 100 Hz as we observed better results with lower sampled data. Since this is time-series data, we adopted a sliding-window approach as follows. For both autoencoder and symbol predictor, the size of the input window is 5 secs, with a stride length of 0.5 secs. For symbol predictor, the size of prediction window is 3 secs. In order to obtain meaningful results (e.g., anomalous results corresponding to when the car is parked are

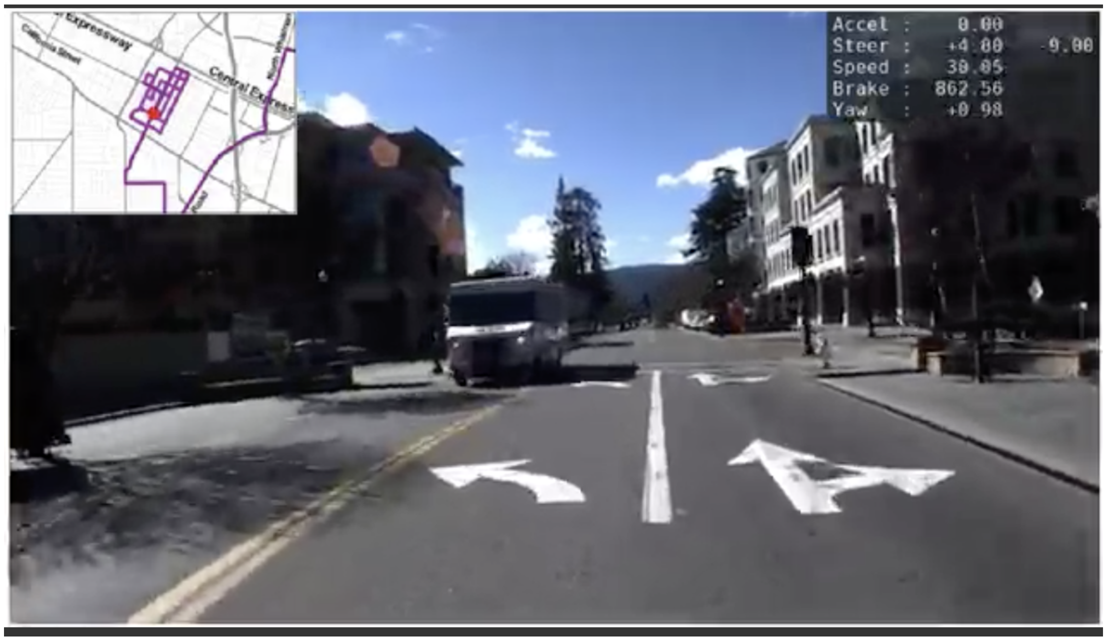


Figure 5.8: A screenshot of the driving scene (as taken from the driving vehicle) from the HDD dataset [7].

not useful), we filtered out those windows where the maximum speed of the vehicle is less than 15 mph. This results in a total of 762671 datapoints (windows). We then scaled this data between 0 and 1 in order to make the network invariant to scales of data. Of this data, 70% is used for training the models and rest for evaluating the performance (i.e., 533869 windows for train and 228802 windows for test). Table 5.1 shows the annotated maneuvers/labels present in the HDD dataset ('Background' indicates going-straight) with corresponding percentage of occurrence.

Training: We used tensorflow to build, train and test the models with a minibatch size of 512 windows. Weights for reconstruction loss (task A), cross-entropy loss (task B) and regularization loss have been set empirically as follows—$w_A = 1$, $w_B = 0.001$ and $w_R = 0.0001$. We used $k = 0.5$ to scale the weights in cross-entropy loss as mentioned previously. We used two-layers of bi-LSTMs with a hidden size 256 units for each LSTM cell. We trained the overall network for about 300 epochs using Adam optimizer [118] with a learning rate of 0.01 and epsilon value of 0.01.

Comparison with LSTM autoencoder: We compare our approach with fully

Table 5.1: Distribution of maneuvers/labels in the Honda Driving Dataset (HDD).

Label	**Percent [%]**
Background	87.15
Intersection Passing	6.00
Left turn	2.58
Right turn	2.31
Left lane change	0.54
Right lane change	0.50
Crosswalk passing	0.27
U-turn	0.23
Left lane branch	0.20
Right lane branch	0.08
Merge	0.14

unsupervised LSTM autoencoder. The network architecture, training method and parameters are similar to that of the LSTM autoencoder part of our multi-task network.

Quantitative results: After the network has been trained, we tested it on evaluation/test data. Fig. 5.9 compares the reconstruction MSE loss between our approach and LSTM autoencoder vs. the number of epochs on test data. We can notice that our approach converges to a lower loss. Table 5.2 shows the average normalized reconstruction loss on test data for different modalities between our approach (multi-task learning) and LSTM autoencoder. We can notice that, our approach results in lower reconstruction loss with 33% lower error (0.2) compared to the standalone autoencoder (0.3) in the 'combined' category. This shows that the combined system does a better job of learning representations than the standalone autoencoder, resulting in lower loss. Fig. 5.10(a) compares the weighted cross-entropy loss between our approach and a standalone symbol predictor. We can notice that our approach achieves lower loss than the symbol predictor. Also, we can observe that by coupling an autoencoder to a symbol predictor, the zig-zag behavior of the latter has been smoothed out. We can observe similar behavior in Fig. 5.10(b) for symbol prediction accuracy (as our data is annotated with maneuvers, we are able to calculate the maneuver prediction accuracy

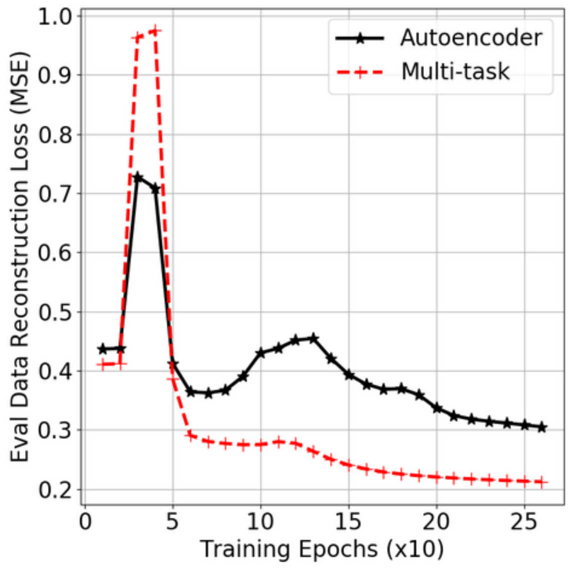
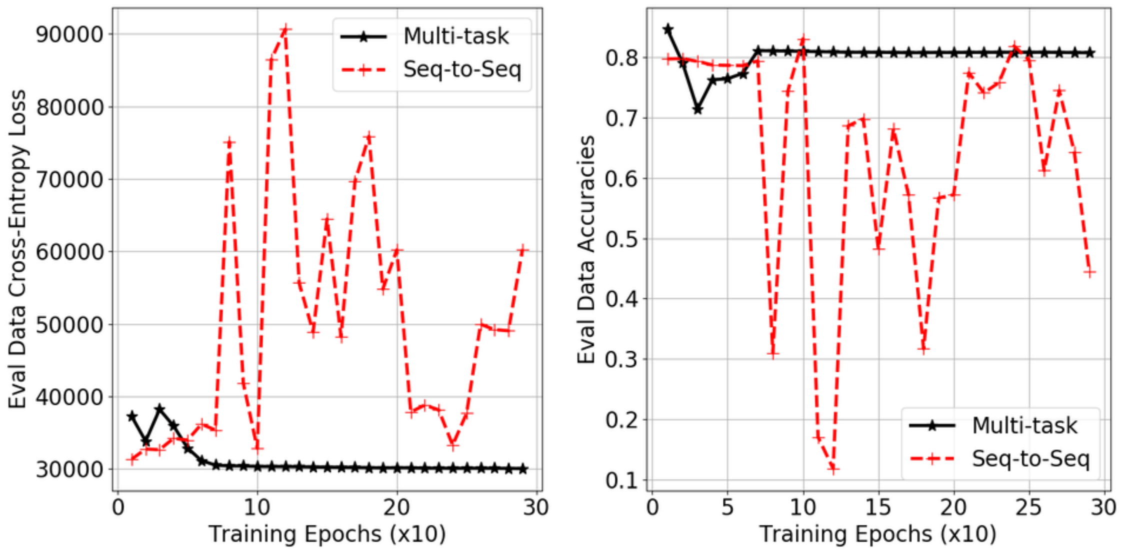
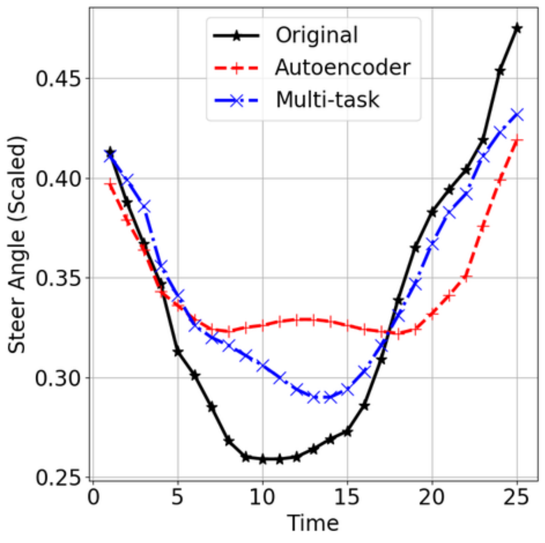


Figure 5.9: Comparison of reconstruction MSE loss performance on test data between multi-task learning (our approach) and a standalone autoencoder.

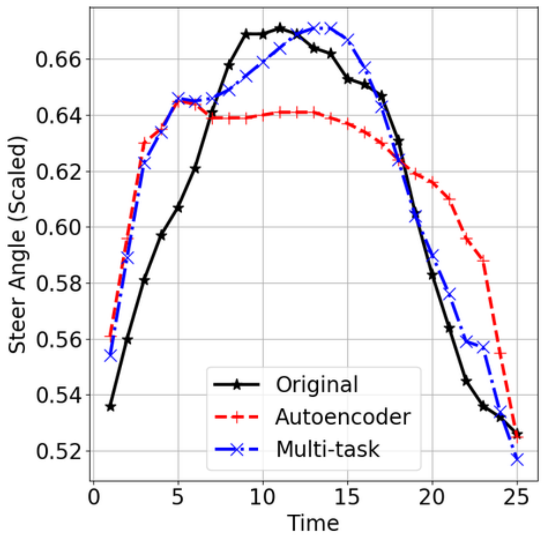


(a) (b)

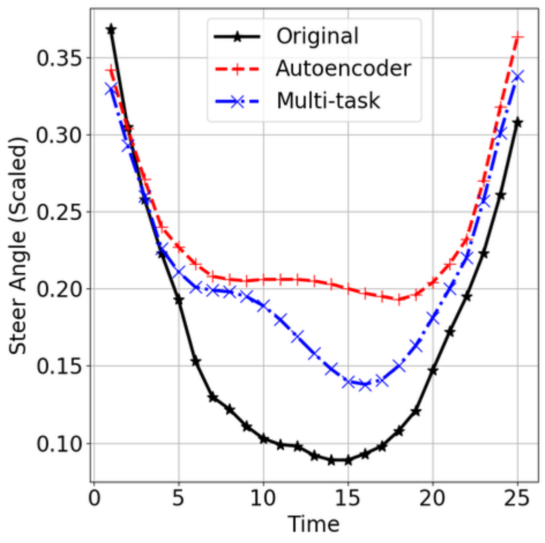
Figure 5.10: Comparison of performance on test data between multi-task learning (our approach) and standalone symbol predictor: (a) cross-entropy loss; (b) symbol prediction accuracy [%].



(a)



(b)



(c)

Figure 5.11: Reconstruction performance of turn data between multi-task learning (our approach) and standalone autoencoder: (a) left turn; (b) right turn; (c) U-turn.

with respect to ground truth). Fig. 5.11 compares the reconstruction performance of three sample turns in test data—Left, Right, U—between our approach and standalone autoencoder. Here, the steer angle data has been scaled to be between 0 and 1. We can notice in all three cases that our approach does a better job of reconstruction when compared to original data

Table 5.2: Comparison of normalized reconstruction MSE losses.

Feature	**LSTM Autoencoder [107]**	**Our Approach**
Steer Angle	0.0005	0.0003
Steer Speed	0.0004	0.0003
Speed	0.0004	0.0003
Yaw	0.0004	0.0003
Pedal Angle	0.0012	0.0012
Pedal Pressure	0.0012	0.0003
Combined	0.3043	0.2082

Table 5.3: Comparison of qualitative results by analyzing top 0.01% scores.

Category	**LSTM Autoencoder [107]**	**Our Approach**	**Our Approach (Scaled Scores)**
Speed	21.7%	21.7%	30.4%
K-turns	13.0%	8.8%	17.4%
U-turns	4.4%	-	-
Lane Change	34.8%	47.8%	39.1%
Normal	26.1%	21.7%	13.1%
Total	100% (23)	100% (23)	100% (23)

Qualitative results: After the network is trained, the reconstruction errors (of dimension 25 due to 5 Hz sampling for 5 secs) for each modality are fit to 25-variable gaussian distribution as explained previously. We also considered another modality which is a combination of all of them. The errors corresponding to this combined modality are fit to a 300-variable (25×6) gaussian distribution. We then passed the test data (in windows) to the network and calculated the Mahalanobis distances (anomaly scores) for each window of data as per Fig. 5.3. We also calculated the scaled anomaly scores using the predicted maneuvers by dividing the anomaly scores with the negative log-likelihood of the predicted maneuvers as per Fig. 5.7. For both cases (scaled and non-scaled), we analyzed the top 0.01% scores and their corresponding windows. For this purpose, we

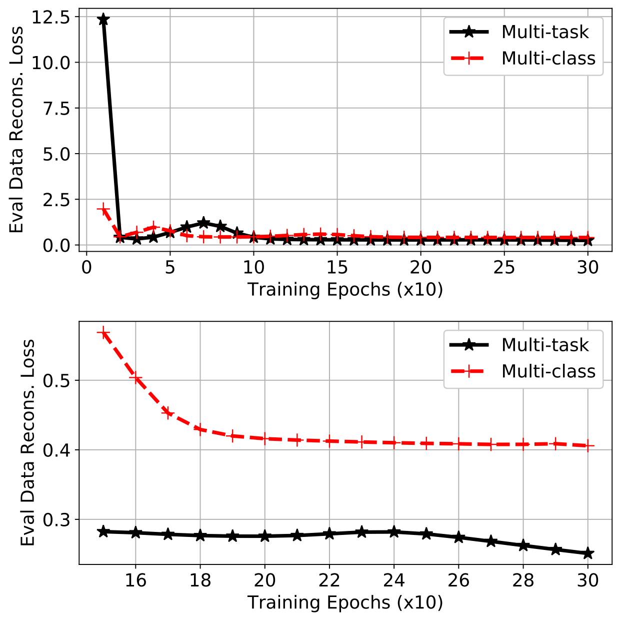


Figure 5.12: (Top) Comparison of eval data MSE reconstruction loss between multi-task learning (our approach) and multi-class LSTM autoencoder. (Bottom) Zoomed version of above showing MSE reconstruction loss for 150 to 300 training epochs, for clear visualization.

extracted the video segment corresponding to each window and manually inspected to check if there is any anomalous behavior. By analyzing the video segments corresponding to top 0.01% anomaly scores, we could classify them into five categories—'Speed' anomalies (e.g., abrupt braking), 'K-turns', 'U-turns', 'Unusual lane change' and finally 'Normal' (no anomaly has been noticed when inspected visually). We have summarized our analysis results in Table 5.3. We can notice that, while the autoencoder classifies U-turns as anomalous, our approach (both scaled and unscaled) does not. We can also notice that our scaled approach classifies lesser 'Normal' and more 'Speed' anomalies. By comparing, the percentage of 'Normal' cases classified as anomalous, we can tell that scaled approach performs better than unscaled, which in turns performs better than standalone autoencoder approach. A video demo showing the different kinds of anomalies (listed in Table 5.3) detected using the above approaches is at [119].

Comparison with multi-class/ensemble LSTM autoencoder: While the above

Table 5.4: Comparison of normalized reconstruction MSE losses (without U-turn data).

Feature	**Multi-class LSTM Autoencoder**	**Our Approach**
Steer Angle	0.0007	0.0004
Steer Speed	0.0005	0.0004
Speed	0.0006	0.0004
Yaw	0.0006	0.0003
Pedal Angle	0.0014	0.0013
Pedal Pressure	0.0012	0.0004
Combined	0.4058	0.2456

fully unsupervised LSTM autoencoder did not make use of the maneuver labels, we compared our approach with multi-class LSTM autoencoder that makes use of the maneuver labels like our approach. For this purpose and in order to test the performance of the algorithms, we considered one of the maneuvers viz., U-turn as an anomaly. That is, after we split the entire data into train and test data windows, we discarded those windows in the train data where the majority maneuver is a U-turn. The remaining train data, which is mainly devoid of any U-turn windows, is fed to our multi-task classifier. For multi-class LSTM autoencoder, we further divided this training data into 10 parts, each part corresponding to one of the 10 maneuvers in Table 5.1 except U-turn. Then we trained 10 LSTM autoencoder classifiers (i.e., an ensemble) corresponding to these 10 maneuvers by providing only the data specific to that maneuver. When given a test data/window, each of the 10 classifiers are used to find 10 reconstruction loss values. Then the lowest of these is considered the reconstruction loss for that test data point.

Quantitative results. Fig. 5.12 shows the quantitative results, which compare the eval data MSE reconstruction loss between our approach (Multi-task) and multi-class LSTM autoencoder approach as the number of training epochs is increased. We recall that the reconstruction loss for multi-class approach is obtained as the lowest reconstruction loss corresponding to 10 different class (maneuver)-specific autoencoder classifiers. We can observe that Multi-task approach finally achieves a lower loss compared to multi-class approach. The final (after 300 epochs of training) reconstruction loss on the eval/test

Table 5.5: Comparison of percentage of U-turns detected (qualitative results) by analyzing top anomaly scores.

Percentile Top Scores	**Multi-class LSTM Autoencoder**	**Our Approach**	**Our Approach (Scaled Scores)**
0.001	0.39% (3/765)	1.70% (13/765)	7.97% (61/765)
0.01	1.96% (15/765)	7.97% (61/765)	29.02% (222/765)
0.1	13.33% (102/765)	17.25% (132/765)	48.63% (372/765)
0.5	73.46% (562/765)	52.68% (403/765)	84.44% (646/765)
1	100.00% (765/765)	99.87% (764/765)	100.00% (765/765)

data for each feature is summarized in Table 5.4. We can notice that our approach achieves lower reconstruction error for all features, compared to multi-class approach.

Qualitative results. In order to evaluate the qualitative performance of the algorithms, we first sorted the reconstruction losses/scores in decreasing order and then found the number of U-turn windows detected in the test data by each approach in the top $0.001, 0.01, 0.1, 0.5, 1$ percentile anomaly scores. The results are shown in Table 5.5. For our multi-task approach, we have two scenarios—actual reconstruction loss and scaled reconstruction loss. Considering especially the top percentiles, we can notice that our approach with scaled scores performs better than our approach with normal scores which in turn performs better than the multi-class approach. For example, considering the top 0.001 percentile anomaly scores for each approach—our approach with scaled scores is able to detect 7.97% i.e., 61 of a total 765 U-turn windows in test data (consisting of 228802 windows), while this number is 1.7% for our approach with actual scores and only 0.39% for multi-class autoencoder approach.

5.5 Summary

In this chapter, we have presented a multi-task learning based anomaly detection framework that performs better than existing LSTM autoencoder based approaches. We leverage domain knowledge to reduce false positives. We have qualitatively and quantitatively showed the benefits of the proposed approach on 150 hours of driving data.

Chapter 6

On-board Deep-learning-based UAV Fault Cause Detection and Identification

We have seen in the previous chapter techniques for anomaly detection from multi-modal scalar sensor data, with specific application to driving data. In this chapter, we extend some of those techniques for anomaly detection in Unmanned Aerial Vehicle (UAV) sensor data. Further, we will also propose techniques for anomaly *identification* from the same sensor data.

The advancement in the technology of UAVs/drones and concern of safety are pushing many government and defense organizations to use UAVs for surveillance. E-shopping companies like Amazon are planning to use UAVs for home delivery of their products. Further, drones are also being planned for use as mobile air-policing vehicles in some countries. The advantage that drones can replace humans in potentially dangerous situations is the main factor behind investing and researching on UAV technology and solutions.

6.1 Introduction

Drones or UAVs are Cyber Physical Systems (CPS) with the increase of which, there are risks of both physical as well as cyber attacks on them [120]. Examples of cyber attacks are GPS spoofing attacks [121], signal jamming, control command attacks, attacks on sensors [122], keylogging virus, etc. Examples of physical attacks (both unintentional and intentional) are bird hits, abrupt wind changes, broken propellers, etc. Large sized drones/UAVs are capable of killing people if they fall from heights due to the massive potential energy possessed by them. The increase in use of drones/UAVs currently and in projected future makes real-time incident analysis for drones or UAVs a priority.

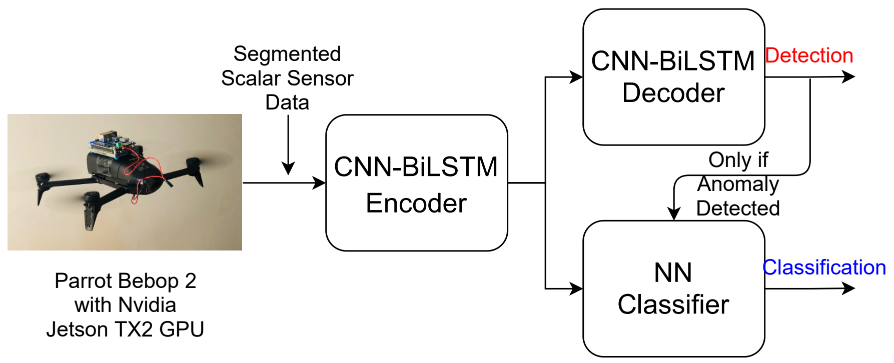


Figure 6.1: An overview of our proposed on-board deep-learning based UAV fault detection and identification/classification framework.

The hobbyists owning drones/UAVs, researchers and the government all will be more curious to know the cause that prevented UAVs from not reaching its destination or deviated from its intended path. As such UAV fault/anomaly *detection* as well as cause *identification* are important. Firstly it is important to detect when the UAV's operation deviates from the normal. Once it is determined that something is anomalous, more resources can be utilized to identify the cause. Identifying reasons for failure is important so that appropriate action can be taken to minimize further loss. For example, in the case of a car, knowing that the failure is caused by a flat tire will help to avoid actions such as sudden braking which will further exacerbate the situation (as it results in loss of control). Similarly in case of a flying object, for certain failures, gliding may be the best solution instead of the much obvious landing. On the other hand Artificial Intelligence (AI) based data driven techniques are increasingly being used to solve many complex problems in several domains relating to autonomous vehicles [119, 123, 124], smartphones [125–127], among others.

Our Approach: Direct and continuous analysis of sensor data for real-time identification of faults is not recommended due to two reasons—(i) it is computationally prohibitive especially on resource constrained devices such as UAVs as it requires processing huge set of sensor data; (ii) it requires significant amount of precious on-board memory resources to store the real-time stream of sensor data. Hence, considering the resource constraints [128, 129] of these devices, we adopt a two-step approach (Fig. 6.1) where in the identification/classification step is carried out only if an anomalous behavior is detected in the sensor data. Unlike the previous works which are mostly model-based [130], we follow a completely (sensor) data-driven approach (using UAV Inertial Measurement Unit (IMU) sensor data such as accelerometer, gyroscope, etc.) for both detection and identification steps.

The reason for choosing data-driven approach (such as deep learning techniques) over traditional model-based approaches are as follows. Deep learning techniques—(a) have ability to learn complex patterns especially non-linear functions; the sensor data of a UAV at times of potential crash events (such as broken propeller) is highly non-linear and complex in nature; (b) have no requirement to manually design the features from the data—the layers in a deep network learn meaningful features on their own during the training process. This also translates to another advantage of not needing domain expertise to extract the features; (c) can work with unlabeled data in unsupervised fashion to generate features. This is very much beneficial for crash-like scenarios due to scarce availability of labelled data. To this end, we propose a novel Convolutional Neural Network (CNN) and bidirectional-Long Short Term Memory (bi-LSTM) deep neural network based autoencoder for *detection* of faults/anomalous patterns followed by a CNN-LSTM deep network for their *classification/identification.*

Main Contributions can be summarized as follows.

- We propose a novel Convolutional Neural Network (CNN) and bidirectional Long Short Term Memory (bi-LSTM) based deep autoencoder network architecture for real-time *detection* of anomalous patterns in UAV IMU sensor data.

- We propose a novel CNN and LSTM based deep neural network classifier for real-time *identification* of the (cause of) fault/attack/crash based on the UAV IMU sensor data.

- We induce crash scenarios by modifying the firmware internals of both the AirSim drone simulator as well as a real drone [131].

- We validate the proposed models via both experiments and simulations. According to the results, our solution is able to detect anomalies with over 90% accuracy and can classify drone mis-operations correctly with about 99% (simulation data) and upto 85% accuracy (experimental data).

Chapter Outline: In Sect. 6.2, we review related work. In Sect. 6.3, we describe our UAV fault/crash detection and identification methods. In Sect. 6.4, we present both experimental and simulation results. Finally, in Sect. 6.5, we summarize the chapter.

6.2 Related Work

We position our work with respect to the related work that can be classified into the following categories: (i) Fault Detection and Identification (FDI); (ii) anomaly detection; (iii); deep learning approaches. The related work presented below corresponding to (ii) and (iii) should be read in conjunction with similar content presented in the previous chapter.

Fault Detection and Identification (FDI): Much of the existing work on FDI focuses on faults in sensors/actuators in the UAV. For example, Panitsrisit et al. [132] propose a hardware duplication system consisting of piezoresistive sensor, pressure sensor, and current sensors to detect faults in the elevator of the UAV. Any abnormal outputs from these sensors will be detected as a failure. Rago et al. [133] present a FDI method for the failure of sensors/actuators based on Interacting Multiple-Model (IMM) Kalman Filter approach. Actuator/sensor failures are represented by a change in the model representing the dynamics (measurements) of the system. Drozeski et al. [134] present an FDI method using three-layer feed-forward neural network based on state information. Heredia et al. [135] uses Observer/Kalman Identification (OKID) estimator to estimate the system state from measured input-output data. Detection of faults is done by noting deviation from the expected output beyond an accepted threshold. They use a separate estimator for each output to make the identification problem trivial. Taking a different approach, Suarez et al. [136] use kalman filtering in combination with visual techniques such as 3D projection from two observers to detect faults in the target UAV in a multi-UAV setting.

Anomaly Detection: Anomaly detection is generally an unsupervised machine learning (ML) technique due to lack of sufficient examples for the anomalous class. Within unsupervised learning, it can be broadly classified into the following categories —statistical/regression, dimensionality reduction and distribution-based approaches. In statistical approaches [113], features are generally hand-made from the data such as mean, variance, entropy, etc. Certain statistical tests/formal rule checking actions are performed on these features to determine if the data is anomalous. However these approaches work only when the anomalous patterns are known apriori so they can be monitored on the sensor data. In dimensionality reduction, the data is projected onto a low-dimensional representation (such as principal components in Principal Component Analysis (PCA)). The idea is that, this low-dimensional representation captures the most important features of the input data. Then clustering techniques such as k-means or Gaussian Mixture Models (GMMs) are used to cluster these low-dimensional features to identify anomalies. In distribution-based approaches, the training data is fit to a distribution (such as multi-variate gaussian distribution or a mixture of them). Then given a test point, distance is calculated of this test point from the fitted distribution (e.g., Mahalanobis distance) representing the measure of anomaly.

Deep-learning Approaches: Deep-learning techniques have been widely used to solve many problems in different domains. For example, deep neural network architectures have been used to predict seizures [137], deep CNNs have been used extensively in content recommendation [138], speech recognition [139], computer vision [140], etc. On the other hand, RNNs and LSTMs have been used for Model Predictive Control based robotic manipulation [141], language modeling [142], phoneme recognition [143], etc. To the authors' best knowledge, deep learning techniques have never been used to detect/identify the cause of the UAV crashes based on sensor data. In this chapter, we propose novel deep learning architectures to detect and identify the cause of UAV crashes or crash-like scenarios from drone's IMU data.

6.3 Proposed Solution

In this section, we first explain our CNN bi-LSTM autoencoder network to *detect* anomalies followed by CNN bi-LSTM network classifier to *identify* anomalies.

CNN and bi-LSTM based Detector ('AutoEnc'): Anomaly detection using unsupervised learning consists of two steps. In step 1, the system is trained with several normal examples to learn representations of the input data e.g., GMM clustering. Because we are dealing with temporal data, a sliding window approach needs to be adopted to learn these representations. In step 2, given a test data point, we define an anomaly score based on the learned representations, e.g., distance from the mean of the cluster. In an LSTM autoencoder, input time series data, $\{x_0, x_1, ...x_n\}$, of size $n+1$ (corresponding to one window of data segmented from full data) is fed to the encoders which consist of $n+1$ LSTM cells. The output of the last LSTM cell—called the embedding—is fed as input to a series of $n+1$ LSTM cells to generate an output, $\{a_0, a_1, ...a_n\}$. The autoencoder is trained by minimizing the mean squared reconstruction error between input and output as in $|\mathbf{x} - \mathbf{a}|^2$.

Convolutional bi-LSTM Encoder: The basic encoder in an LSTM autoencoder does not perform sufficiently well as it not does take into account: (i) inter channel/modal correlations (ii) directionality of data. We design an encoder that addresses these issues as shown in Fig. 6.2(top). It consists of a series of 1-dimensional (1D) convolutional layers followed by bi-LSTM layers. The convolutional layers help in capturing inter-channel spatial correlations, while the LSTM layers help in capturing inter- and intra-channel temporal correlations. The number of filters, the size of filter kernel and the type of padding (with a default stride length of 1) is indicated in Fig. 6.2(top). For example, in the first convolution step, 48 filters of size 5×1 are applied to the input data of size, $25 \times 1 \times 6$ (assuming a 6-channel input data), to result in an output of size, $25 \times 1 \times 48$. Unidirectional LSTM layers capture temporal patterns only in one-direction, while the data might exhibit interesting patterns in both directions. Hence to capture these patterns, we have a second set of LSTM cells for which the data is fed in the reverse order. Further, we have multiple layers of these bi-LSTM layers to extract more hierarchical information. All the data that has been processed through multiple convolutional and bi-LSTM layers is available in the cell states of final LSTM cells. This is the output of the encoder which will be fed as input to our decoder.

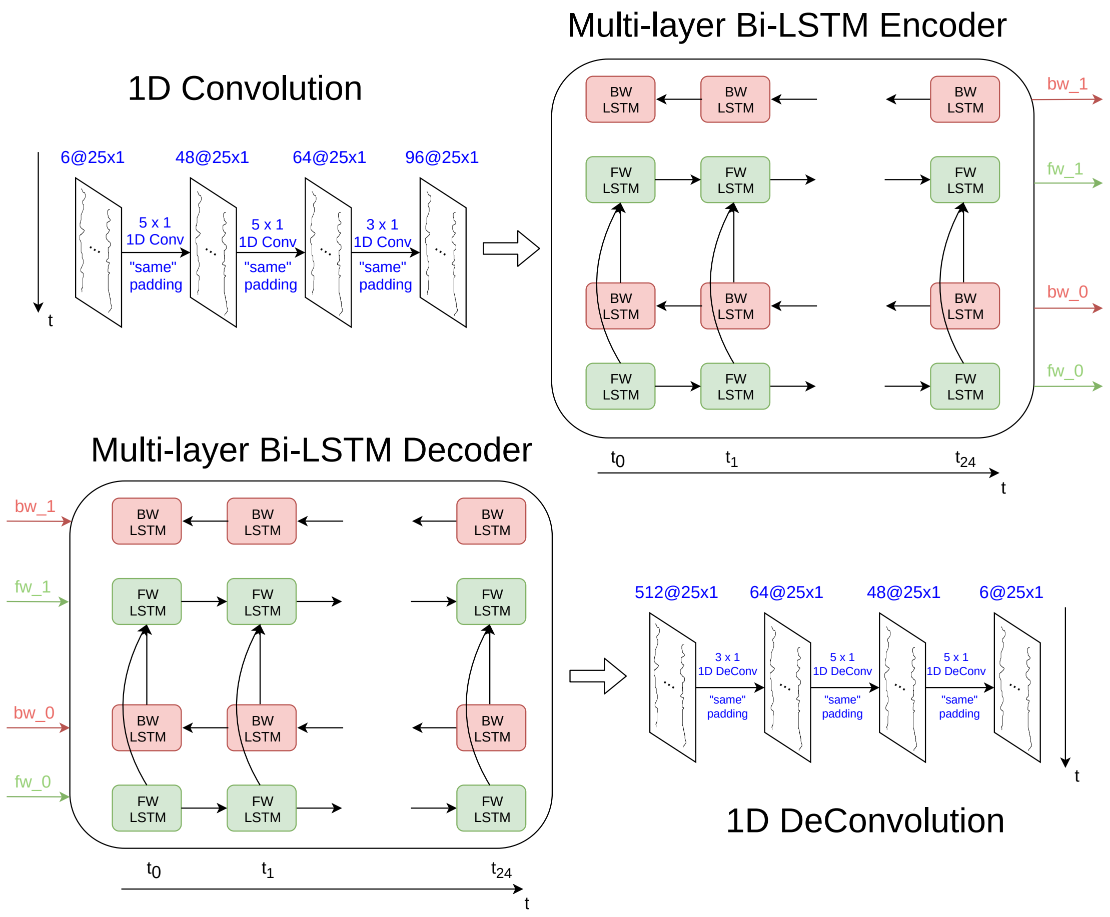

Multi-layer Bi-LSTM Encoder
1D Convolution
6@25x1
48@25x1
64@25x1
96@25x1
5 x 1 1D Conv "same" padding
5 x 1 1D Conv "same" padding
3 x 1 1D Conv "same" padding
t
BW LSTM
FW LSTM
bw_1
fw_1
bw_0
fw_0
t0
t1
t24
Multi-layer Bi-LSTM Decoder
512@25x1
64@25x1
48@25x1
6@25x1
3 x 1 1D DeConv "same" padding
5 x 1 1D DeConv "same" padding
5 x 1 1D DeConv "same" padding
1D DeConvolution

Figure 6.2: Convolutional and bi-LSTM encoder (top) and decoder (bottom) of the proposed autoencoder.

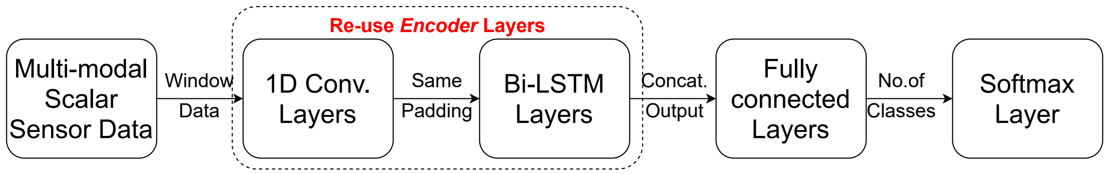


Figure 6.3: Proposed deep CNN and bi-LSTM architecture for fault classification. For 1D Conv. and bi-LSTM layers, please refer the encoder in Fig. 6.2.

Convolutional bi-LSTM Decoder: The decoder performs encoder operations in reverse order so as to reconstruct the input data (Fig. 6.2(bottom)). It first consists of bi-LSTM layers which take the final cell states from encoder as one of the inputs (the other input being zero). Other input (other than the previous cell state) can be either zero or the output of the previous LSTM cell. The outputs of LSTM layers are fed as input to a series of 1D de-convolutional layers which perform reverse of convolution (also called transposed convolution) to generate data with same shape as that of input data to encoder (6-channel 1D data of length 25).

Though reconstruction error can directly be used as a measure of anomaly for step 2, we design an enhanced method with further processing to obtain better results (Fig. 5.3). After the network is trained, the train data is again fed to the trained network to capture the reconstruction errors. These errors are then fit to a multivariate gaussian distribution. Given a test data point, the reconstruction error is first calculated using the trained model. Mahalanobis distance of the error is then calculated with respect to the fitted gaussian model using the formula shown in Fig. 5.3. These distances, which are considered anomaly scores are then sorted in decreasing order and analyzed as per requirements e.g., top 0.01%.

CNN and LSTM based Classifier ('DCLNN'): We consider the drone's sensor signatures in crash or crash-like scenarios to be very valuable. These signatures are mostly unique to the events that caused them. As such, we claim that these signatures can be used to identify those events by building a classifier mapping sensor signatures to events that caused them. For example, the data collected from the 3DR Solo drone [144] after a propeller was broken is shown in Fig. 6.4. The plots show that the drone was at stable state when one of its propellers was broken, this resulted in large variations

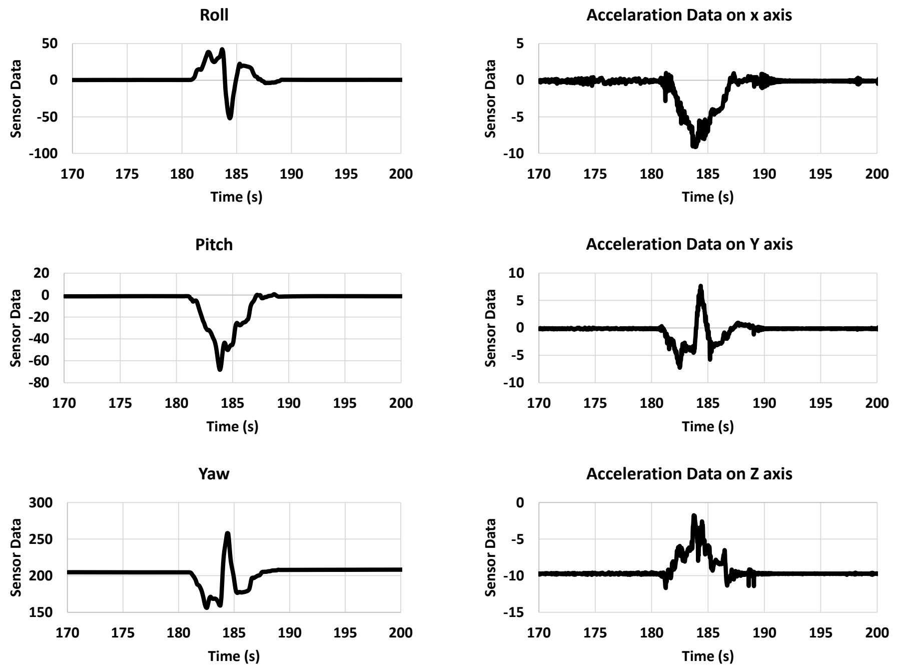
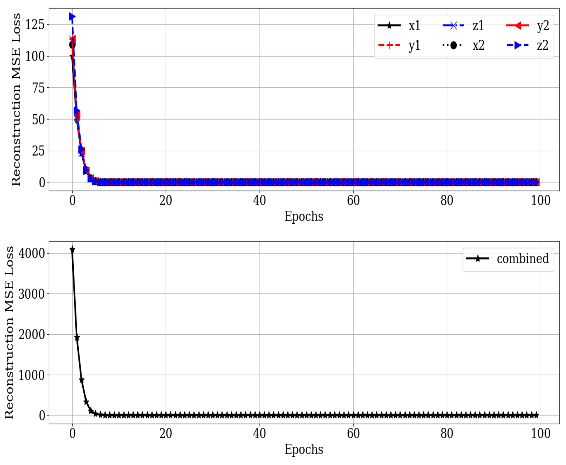


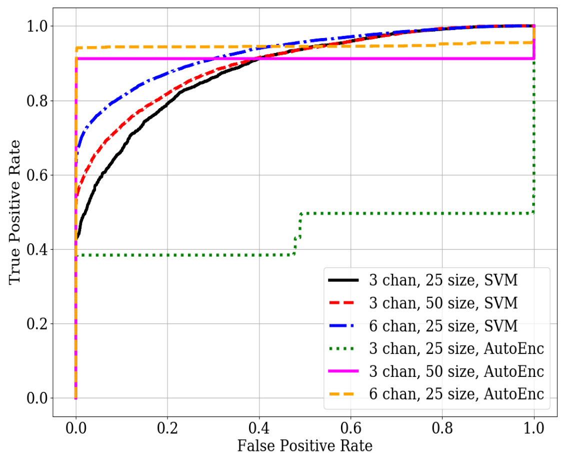
Figure 6.4: Sensor data corresponding to broken propeller scenario.

in the accelerometer and pitch-roll-yaw data (unique signatures) since the drone accelerates in a particular direction after the thrust on the broken propeller is zero. These signatures are zoomed to show the variation. Other sensor data such as gyroscope and magnetometer show similar variations (not shown).

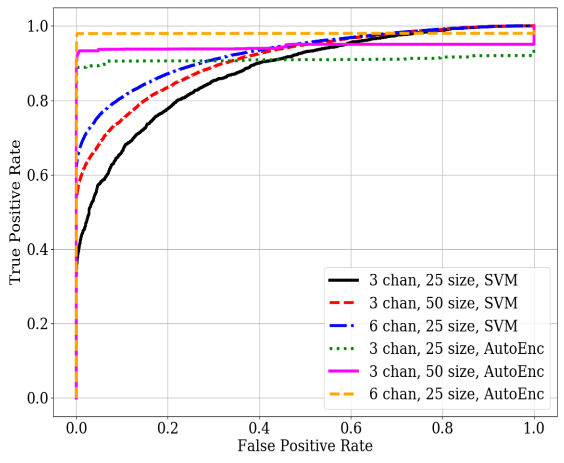
We propose a novel CNN and bi-LSTM based architecture to classify the sensor data in real-time to identify any potential crash scenarios. This is useful in two ways—(a) it can be used for recovery planning to stabilize the drone using either redundant hardware or designing appropriate controller techniques that work only based on less than usual number of actuators; (b) in cases where the crash is unavoidable, the logged sensor data can be used to identify the cause of the crash offline. The proposed deep architecture is shown in Fig. 6.3. It consists of convolutional layers in the beginning as in the case of 'AutoEnc' to extract important static/spatial features followed by LSTM layers to capture the dynamic/temporal variations in the sensor data. As such the encoder layers in Fig. 6.2 can be reused as shown in both Fig. 6.3 and Fig. 6.1. At every convolutional layer, each channel is processed by multiple kernels/filters resulting in those many feature maps in the subsequent layer. The output of the convolutional layers is time-wise unrolled and passed through bi-LSTM layers to capture the temporal



(a)



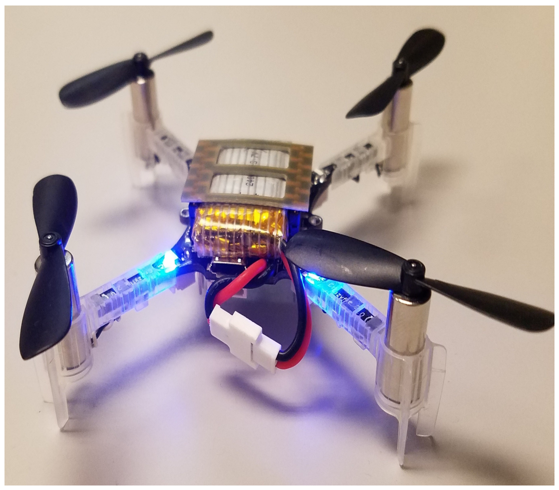
(b)



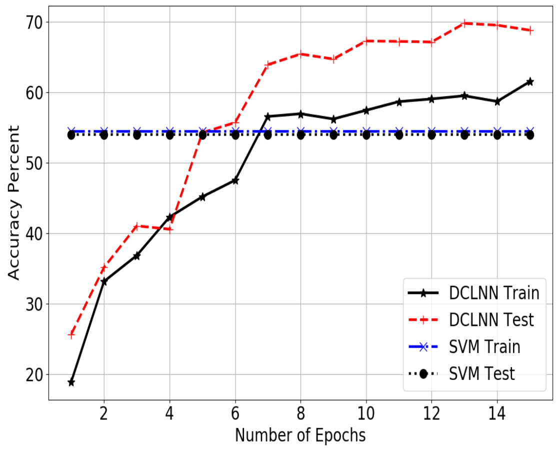
(c)

Figure 6.5: (a) Training reconstruction loss across each channel and combined data (experimental data); Receiver Operating Characteristic (ROC) curve for different number of channels and window sizes—(b) experimental data; (c) simulation data.

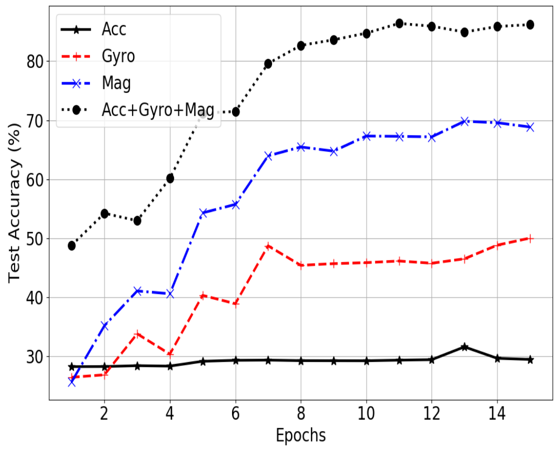
dependencies in both directions. The concatenated outputs (of forward and backward LSTM layers) are then passed through a series of fully connected layers ending in the softmax layer (of length equal to number of classes) that computes the probabilities of different classes. We do not use pooling layers after convolutional layers as our input data has been segmented into windows and the output of the convolutional layers need to be passed through time-series LSTM layers [145]. We also introduce dropout at several layers in our architectures wherein, some of the activations chosen randomly are



(a)



(b)



(c)

Figure 6.6: (a) Crazyflie 2.0 drone used for experiments; (b) Accuracy using magnetometer data; (c) Comparison of test accuracy across different channels.

made zero. This will act as regularization and help prevent the network to depend on some idiosyncrasies and instead learn the general structure.

Real-time operation. Once the models are trained offline, detection and (if necessary) identification can be done in real-time as it comprises only of segmenting the streaming sensor data and applying a few matrix multiplications to arrive at the result. Identification step is carried out only if the anomaly scores from detection step is beyond a threshold. Once the cause is identified/diagnosed in real-time, appropriate actions can be taken to stabilize/safeguard the drone operation.

6.4 Performance Evaluation

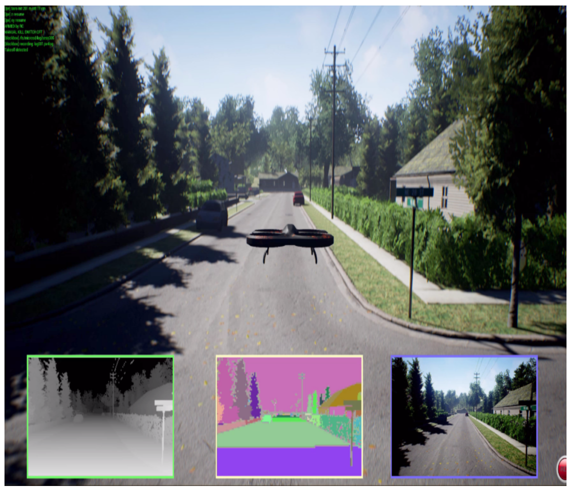
In this section, we first present our experimental and simulation setup followed by detection and identification results. For each case, we compare our approach with traditional machine learning classifiers such as SVM.

Experimental Setup: Crash data in Fig. 6.4 is collected using 3DR Solo drone. However, we could not use the same drone for experiments as it weighs 1500 grams and easily gets damaged when it falls from heights. This is more relevant in the case of deep learning where more amount of data needs to be collected to train the models. For this purpose, we used another small drone, called CrazyFlie 2.0 [131], shown in Fig. 6.6(a) weighing just 37 grams and much more robust to falls. In order to evaluate our approach, we considered a total of 15 crash scenarios (classes)—all combinations of one/two/three/four propeller breakdown cases. We modified the drone firmware by assigning a value of zero to the variables representing the propeller Revolutions Per Minute (RPM) at appropriate levels within the software to induce crash. However, due to firmware limitations, we could not successfully induce the following cases—all four cases of three-propeller breakdown and 2 cases of diagonal two-propeller breakdown, resulting in only 9 classes totally. We collected the data for 30 runs; in each run, the drone would have a 2 second upflight time, an 8 seconds of hovering time followed by crash corresponding to one of the 9 classes. We collected Accelerometer, Gyroscope and Magnetometer data sampled at 100 Hz (maximum supported rate) for each crash event.

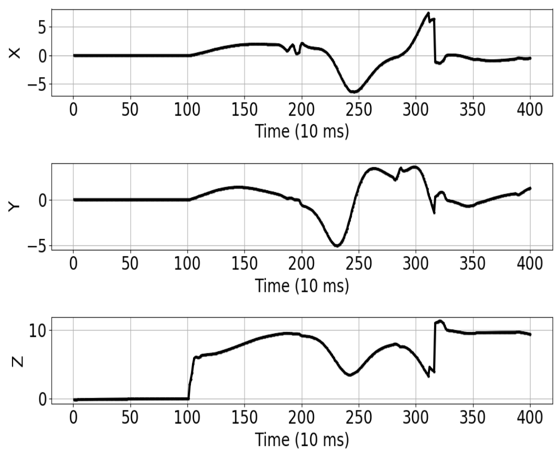
Training Description: We used 70% of data for training and 30% for testing in both experiments and simulations. Since our data is time-series, we processed it into windows of size $100, 50, 25$ timesteps with a stride length of 10 for suitable analysis. All the data till the crash is used for training/testing AutoEnc as it considered normal operation. Transition data at the time of crash is used for training/testing DCLNN along with corresponding class label (i.e., crash scenario mentioned above). We used tensorflow to build, train and test our models with a minibatch size of 512 windows. The number and the size of filters used for CNN layers is as shown in Fig. 6.2. We used two-layers of bi-LSTMs with a hidden size of 256 units for each LSTM cell. We trained the overall network for $100 - 300$ epochs using Adam optimizer [118] with learning rate of 0.01 and epsilon of 0.01.

Detection Results: Fig. 6.5(a) shows the training reconstruction loss of the experimental data for our AutoEnc model over several training epochs for 6-channels (accelerometer and gyroscope) individually and in combined form. We can notice that as the model learns, the reconstruction loss decreases and converges. We considered detection as a binary problem. Fig. 6.5(b) shows the Receiver Operating Characteristic (ROC) curve for the experimental data compared with SVM classifier (with best parameters: Radial Basis Function (RBF) kernel, $C = 10$, $\gamma = 0.1$). We can notice that: (i) accuracy increases as the number of channels or the window size considered is increased; (ii) AutoEnc performs better than SVM except for the 3-channel, 25 window size case. We believe the reason is due to lack of sufficient data for training AutoEnc (as AutoEnc is a deep learning network more data is required to train the model). For the same reason, this behavior is not observed in 6-channel case. Fig. 6.5(c) shows the ROC curve results for our simulation data (we describe the simulator setup and data collection below). We can observe accuracy greater than 97% and also notice that we do not observe the similar problem observed in experimental data.

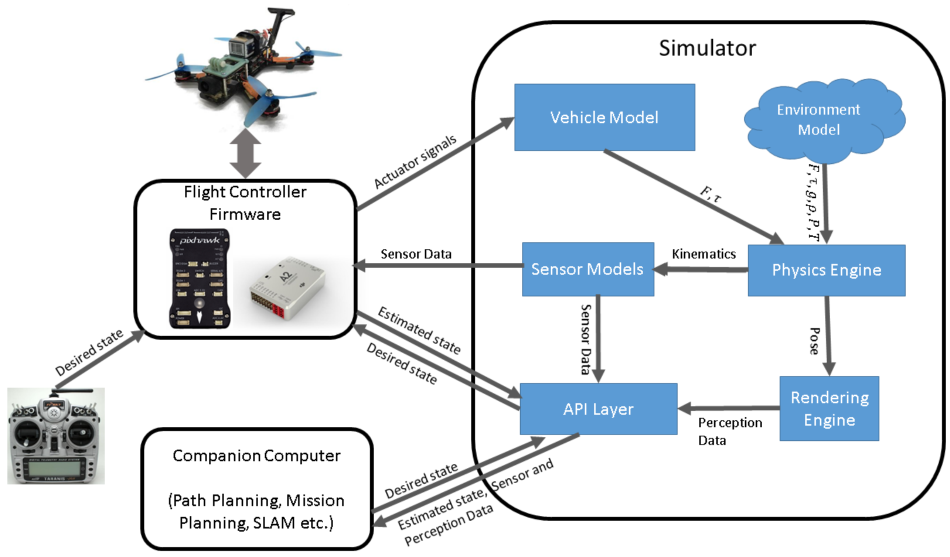
As can be observed from these experimental results, the detection accuracy can still be improved. We believe the reason for this is the unstable nature of our drone (Crazyflie 2.0) used for experiments. As the drone is very light weight (only 37 grams) and hard to control, it exhibits a certain element of randomness. This is also reflected in crash signatures making it hard for the network to learn meaningful representations as we show below for experimental data. On the other hand, we are also restricted in using larger drones due to—(i) state laws prohibiting flying drones outdoors without an expensive license and it is very risky to fly these drones indoors; (ii) these drones get easily damaged when they fall from heights making them unsuitable for crash experiments. *To find a sweet spot between these two extremes, we decided to use a realistic drone simulator to circumvent these problems.* We felt the simulation should not be simplistic, ignoring the physical aspects including environmental objects such as trees and poles, kinematics such as drag, fiction and gravity, etc. For these reasons, we adopted Microsoft's AirSim



(a)



(b)



(c)

Figure 6.7: (a) A snapshot from AirSim simulator [4] shows an aerial vehicle flying in an urban environment. The inset shows depth, object segmentation and front camera streams generated in real time; (b) Three-dimensional accelerometer data from the simulator after one of the propeller's RPM is made zero (simulating broken propeller crash); (c) The architecture of the simulator depicting the core components and their mutual interactions.

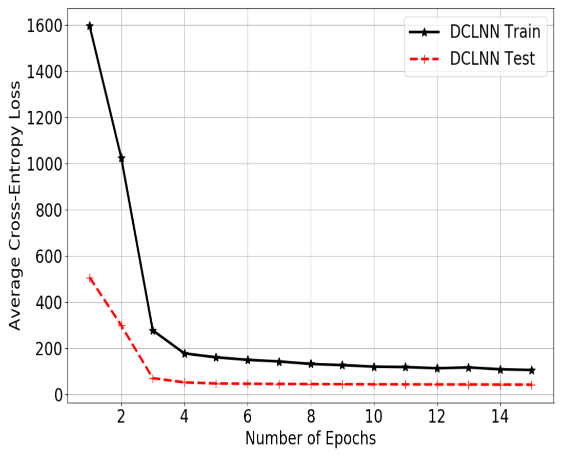
drone simulator [146] which is an open-source simulator written in C++. The simulator tries to simulate the real-world as closely as possible by trying to include all effects involved including collisions (see Fig. 6.7(a) for a screenshot of the simulator).

Simulator Setup: We first describe the simulator architecture followed by data collection method.

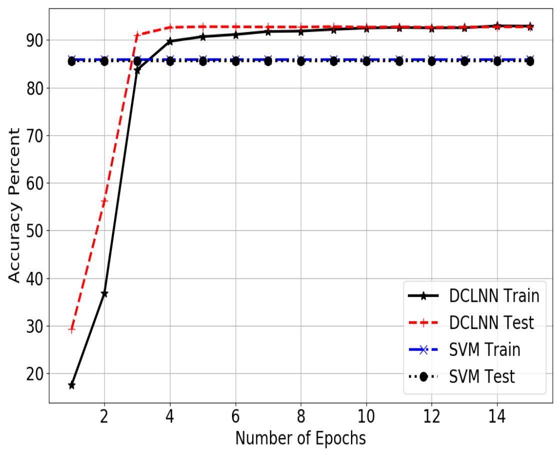
Simulator Architecture: The core components of AirSim include environment model, vehicle model, physics engine, sensor models, rendering interface, public API layer and an interface layer for vehicle firmware as depicted in Fig. 6.7(c). It is necessary to have simulated environments have reasonable details. For this purpose, AirSim leverages rendering technologies implemented by Unreal engine [147]. In addition, AirSim also utilizes the underlying pipeline in the Unreal engine to detect collisions. The AirSim code base is implemented as a plugin for the Unreal engine. For in-depth details on different *realistic* models used in the simulator, please refer [146].

Inducing Crash: In order to simulate a broken propeller, we make its RPM to zero by supplying zero current to its motor. However, for this to work successfully, it is necessary to constantly provide zero current to the effected motor. A one-time operation would not be sufficient as the currents to the motors are generated in a high-frequency update loop using a PID controller and hence correct current values (which do not induce crash) are provided to the motors in subsequent update loops. Hence, we modified the firmware code to make the effected propeller's motor current to zero inside the update loop itself so that its RPM is continuously zero, resulting in a crash. Fig. 6.7(b) shows the accelerometer data (x,y,z axes) after the crash is induced by making the RPM of one of the propellers to zero. By comparing with the actual drone crash data (accelerometer) from 3DR Solo drone in Fig. 6.4, we can see that the simulation data is more complex and hence difficult to learn than the former. We successfully simulated all 15 crash scenarios (classes). In all these scenarios, we collected 18-channel data viz., linear and angular versions of acceleration, velocity, and position along all the three axes from the start of the crash until the end. We repeated this experiment 300 times to account for noises and gather sufficient data. For results below, we used only 3-channel linear acceleration data (instead of all 18 channels), unless otherwise specified, as there is a need to limit the amount of data to process on a real drone.

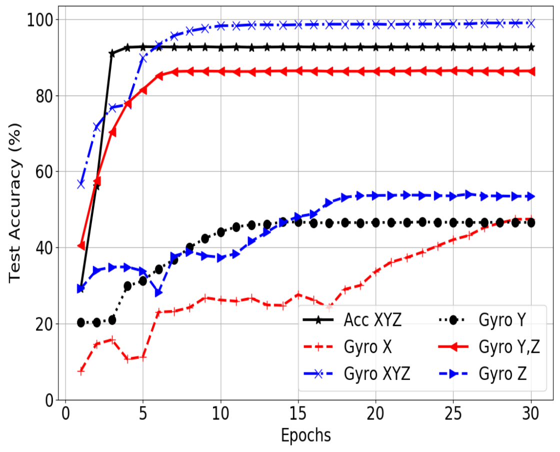
Identification/Classification Results: We used our DCLNN architecture in



(a)



(b)



(c)

Figure 6.8: (a) Cross-entropy loss vs. number of training epochs for both train data and test data; (b) Accuracy vs. number of training epochs compared between DCLNN (ours) and SVM classifier for both train and test data; (c) Accuracy on test data when the channel data fed to the network is varied.

Fig. 6.3 with three convolutional layers ($[48, 64, 96]$ filters with kernel sizes $[5, 5, 3]$) followed by two bi-directional LSTM layers (128 units) and one dense/softmax layer. The training is performed for 15 epochs in mini-batches of size 64 using Adam optimizer and categorical cross entropy as loss function. We now present identification results using experimental data.

Comparison with SVM: Fig. 6.6(b) shows the accuracy of our model (DCLNN) compared against classical machine learning classifier such as Support Vector Machine (SVM). Accuracy is defined as the percentage of windows classified correctly. We can notice that DCLNN's accuracy increases as the training is carried out over several epochs (cross-entropy loss also reduces accordingly but is not plotted due to space limitations), reaching to 70% finally. We can also notice that it performs better than SVM classifier (RBF kernel, $C = 10$, $\gamma = 0.01$), which could only achieve 54% accuracy. Due to space limitations, we have shown comparison only with Magnetometer data; in other channels too, DCLNN performs better than SVM in a similar manner.

Variation with Channels: There is a need to limit the amount of data processed during inference considering the resource scarcity of UAVs. We were curious if all the channels contributed equally to the learning of the network. For this purpose, we trained our network by giving different channel data each time and plotting accuracy on test data as shown in Fig. 6.6(c). We can notice that in 3-channel scenario, Magnetometer performs best with 70% accuracy, while combining all 9-channel data yields an accuracy of about 85%. We now present results corresponding to simulation data.

Comparison with SVM: Fig. 6.8(a) shows the training and testing loss as our network is trained over several epochs. We notice that the loss reduces and converges to a value. This indicates that the network is learning over time and fits the data reasonably. Fig. 6.8(b) shows the accuracy of our model (DCLNN) compared against SVM. Interestingly, after only 3 epochs, DCLNN almost converged with about 90% accuracy (with final value around 93%). The test accuracy also finally converges to 93%, whereas SVM classifier (RBF kernel, $C = 100$, $\gamma = 0.01$) achieves only 85% accuracy. We did not plot F1-score as our class distribution is equal.

Variation with Channels: Fig. 6.8(c) shows the variation with channel data. We can see that 3-channel gyroscope data works better (with about 98% accuracy) than 3-channel accelerometer (93% accuracy) data. Within 3-channel gyroscope, we can notice that axes Y and Z give better performance than axis X. By knowing this, we can discard axis X data and only process axes Y and Z to limit the amount of computation. We can see that axes Y and Z combined can give an accuracy of about 87% accuracy, which

Table 6.1: Inference times for autoencoder (AutoEnc) on different hardware platforms.

No. Chans.	**Desktop (Training)**	**Desktop (Inference)**	**Raspberry Pi (Inference)**	**Jetson TX2 (Inference)**
1 (x/y/z)	202 ms	82 ms	312 ms	83 ms
2 (xy/yz/xz)	386 ms	87 ms	372 ms	92 ms
3 (xyz)	561 ms	95 ms	484 ms	101 ms

could be sufficient in some cases.

Raspberry-Pi and Nvidia Jetson Profiling: The above results are obtained on a desktop computer with Intel QuadCore i7-2600 3.4 GHz processor with 8GB RAM. However, we wanted to know how long it takes to do inference on hardware that can be mounted on a drone and powered using a drone battery. For this purpose, we considered two embedded computing devices—Raspberry Pi 3 Model B and Nvidia Jetson TX2 module, both of which can be mounted on a drone to augment its computing capabilities. The former has a QuadCore 1.2 GHz Broadcom BCM2837 processor with 1GB RAM, the latter has Quadcore 2 GHz ARM processor with 8GB of CUDA-compatible graphic memory. The results are shown in Table 6.1. The numbers indicate the time taken to do inference on a single window of data (100 samples) with the specified number of channels and the hardware. These numbers show that AutoEnc is amenable for real-time inference on a drone aiding in detection of potentially danger modes. Specifically, we can observe that in the case of Nvidia Jetson TX2, the results are impressive around 100 ms or less, which means the drone can do inference on the last one second of sensor data every 100 ms. It is to be noted that, while new sensor data is sampled every 10 ms, it may not be necessary to do the inference at the same speed.

Video Demo: A video demo of the crash experiments performed on Crazyflie 2.0 drone and using AirSim simulator can be found at [148].

6.5 Summary

We proposed novel deep architectures to detect and identify the cause of the malfunction. We have shown that the proposed architecture is able to achieve over 90% accuracy for detection and upto 85% accuracy for identification over experimental data (all-channels combined) and 99% over simulation data (just 3-channels).

Chapter 7

Conclusion and Future Directions

This chapter summarizes the main contributions of this dissertation and discusses future research directions that are worth investigating leveraging the frameworks proposed in this dissertation.

7.1 Summary of Dissertation Contributions

This dissertation provides techniques for real-time autonomic decision making using a group of Intelligent Physical Systems (IPSs) acting in a dynamic and uncertain environment. We specifically considered the application of target/object search using a team of UAVs in a post-disaster scenario. This dissertation discusses the techniques necessary to achieve its objective viz., self-coordination and self-optimization via multi-UAV coordination, and self-healing via proactive monitoring and lastly, the real-time aspects.

Towards this, firstly, we presented a simple approach based on Multi-Agent Reinforcement Learning (MARL) for coordinated data collection in a disaster scenario using human bystanders and UAVs.

Secondly, we extended this simplistic multi-UAV coordination framework to distributed actor-critic based Multi-Agent Deep Reinforcement Learning (MADRL) framework which allows high-dimensional state and action spaces, distributed coordination, unlike the MARL framework. Also this framework addresses the non-stationarity problem inherent to multi-agent settings, which cannot be handled by the MARL framework.

Thirdly, as context constitutes an important ingredient for decision making, it is vital to validate the sensor context of the UAV in certain secure missions. Towards this, we proposed techniques for context modeling and prediction leveraging the theory of Hidden Markov Models (HMMs). The proposed models capture both the personal and group behavior aspects of an UAV from its history of contextual observations.

Fourthly, as the UAVs are operating in a dynamic and uncertain environment, we proposed techniques for real-time monitoring of the operation of the UAV. This constitutes techniques for real-time anomaly detection and classification, that can later enable the correct sequence of remedial actions.

Generalizability: The proposed techniques can be generalized to any group of UAVs operating in a dynamic/uncertain environment.

7.2 Future Directions

In order to further expand the state-of-the-art, we present possible future research directions for each chapter below.

Chapter 2: Firstly, two different kinds of agents can be introduced—human bystanders and rescue personnel each with a different set of actions. Specifically bystanders are assumed to be static with four actions of panning/tilting the phone, while rescue personnel have additional actions of translation (East, West, North, South). Secondly, as privacy is an important aspect, Privacy-Preserving Multi-Agent Deep Reinforcement Learning (PPMADRL) framework can be studied that accounts for user privacy while the media data is being captured by the agents (human bystanders/rescue personnel). Lastly, distributed and local processing using mobile computing paradigm can be investigated for 3D reconstruction as the internet connection may not be reliable in disaster scenarios.

Chapter 3: Firstly, the proposed framework can be extended to non-grid/image based environments. In non-grid/image environments, the observation is the current image seen by the UAV. The state consists of the concatenation of UAV's own observation and the similarity with respect to the other agents' observations calculated using image similarity metrics. The actor and critic architectures can be augmented with Convolutional Neural Networks (CNNs) to process image state spaces. For exploration, each UAV maintains a buffer of previously visited images and compares the current image with the images in its buffer to determine novelty/exploration reward. Collision reward can be formulated by comparing the similarity of the UAV's observations with other UAVs' observations between two consecutive time instants.

Secondly, in this chapter, we have assumed that the communication among the UAVs is synchronous i.e., all UAVs sending their states and actions to other UAVs every time instant, although occasional communication failures are handled by the proposed framework. In some scenarios, it is possible that the team of UAVs working on a mission are split into several subteams either voluntarily due to privacy/security/geographical constraints or involuntarily due to prolonged communication failures. In such scenarios, asynchronous coordination of the UAVs is necessary, for which Multi-Agent Hierarchical Deep Reinforcement Learning (MAHDRL) framework can be explored. In this framework, an additional RL process is run on a server that coordinates the actions of the subteams. HRL is a powerful framework as there are two RL processes running on two hierarchical levels (server and UAVs). For example, such a framework can be used to coordinate among subteams of UAVs operating on a joint national security mission, where the subteams belong to different agencies e.g., Federal Emergency Management Agency (FEMA) and Department of Homeland Security (DHS). Another application can be using Autonomous Underwater Vehicles (AUVs) for underwater adaptive sampling of interesting phenomenon such as water pH, dissolved oxygen levels. In this scenario, communication among the AUV underwater is difficult due to the water medium and hence, they need to surface and coordinate via an Autonomous Surface Vehicle (ASV). The overall goal of adaptive sampling is realized via MAHDRL framework where (i) an RL process running on the ASV is used to optimize the resurfacing time for each AUV (i.e., determine how long each AUV should stay underwater and perform adaptive sampling before surfacing and coordinating with the ASV); (ii) an RL process running on each AUV is optimized to perform the function of adaptive sampling.

Chapter 4: Firstly, we have observed that using homomorphic encryption has significantly increased the training (parameter estimation) time of the HMM algorithms. Hence, reducing these training times leveraging suboptimal encryption algorithms can be investigated; Secondly, it is known that LSTMs are able to capture both long term and short term temporal dependencies in time-series data. Hence, deep learning based

LSTMs can be investigated to model the personal and group behaviours instead of the traditional ML based HMMs. Also, HMMs follow Markov property which means that they are not able to capture dependencies more than one time step into the past, unlike LSTMs. Performance of HMM and LSTM based approaches can be compared and contrasted. Thirdly, it will be interesting to study how the training of the LSTM network can be carried out preserving the privacy of the users' data. For example, combining LSTMs and encryption can be investigated. Lastly, while the proposed context modeling and prediction techniques have been validated for the case of mobile phones, the techniques are equally applicable and extensible to UAVs. It will be interesting to investigate how the remaining operational time, and other indoor locations can be predicted ahead of time to the benefit of the UAV. In particular, the application of multi-UAV package delivery consisting of several hops passing through both indoor and outdoor environments can be considered. In such a scenario, each UAV will establish some sort of personal and group activities (e.g., follow a particular order of hops very frequently) over time, which can enhance the overall productivity of the application via collaborative filtering based knowledge sharing.

Chapter 5: Firstly, more investigation of False Positives (FP) and False Negatives (FNs) can be carried out. For FNs, an existing heuristic based video anomaly detection technique can be used to identify anomalies which can then be filtered by humans. Given this ground truth information, it can be studied if the proposed models are able to capture these anomalies. For FPs, we have seen some artifacts in the data corresponding to 'Normal' (FP) cases (leading them to be classified as anomalous). Nevertheless, it is important to investigate why some other 'Normal' (FP) cases are still classified as anomalous. Secondly, the proposed anomaly detection technique can be enhanced by leveraging the video data in two ways: (i) an iterative approach can be adopted where the anomalies detected using video data can be used to improve (e.g., via cross-comparison) the anomalies detected using the CANbus scalar data and vice-versa. At the end of this iterative approach, an improved anomaly detection framework using just the scalar data can be obtained; (ii) a multi-modal anomaly detection framework consisting of both scalar and video data can be investigated to enhance the overall accuracy. Thirdly, Generative Adversarial Networks (GANs) have shown some promising results recently in the domain of Computer Vision (CV) for tasks such as image denoising, image-to-image and text-to-image translation among others. Considering this, novel variants of GANs, in addition to Variational AutoEncoders (VAEs) (both of which are generative models like the autoencoders presented in this chapter) can be investigated in further improving the anomaly detection performance. Lastly, it is worth investigating on how to extend the symbol predictor (of the multi-task learning framework presented in this chapter) to UAVs. One way can be to encode the maneuvers of the UAV similarly to that of the car such as ‘take off’, ‘veer left’, ‘hover’, ‘land’, etc. This will help in being able to extend the multi-task learning based anomaly detection to UAV data.

Chapter 6: The profiling of these algorithms on NVIDIA Jetson TX2 GPU shows that it takes about 100 ms to run the detection algorithm for each window of data. While this may be necessary for real-time operation in some applications, it is still high for some other time-critical applications/missions. There are several opportunities for extending the work presented in this chapter. Firstly, software optimizations can be investigated such as model pruning, distillation, teacher-student approach among others to further reduce the inference times albeit with certain performance degradation. Secondly, hardware optimizations can be investigated such as realizing the algorithms on an FPGA device instead of a GPU device, as the former provides faster inference time than the latter for Machine Learning (ML) algorithms. Other optimizations include FPGA optimization, implementations on analog hardware.

Thirdly, a hardware-software approach such as mixed/hybrid analog-digital ML system can be adopted. The reasons to consider a mixed analog-digital system are as follows: (a) when executing the same image-classification workload, the FPGA consumes about a couple of Watts, while the GPU consumes about twice as much in both training and inference phases. This means that these devices cannot be ON continuously (e.g., for continuous anomaly detection) as the remaining operational time is a valuable resource in case of UAV. (b) analog hardware is known to be fast and energy efficient compared to digital systems due to (i) avoiding the use of power-hungry

Analog-to-Digital Converters (ADCs) and (ii) the processing time of an analog circuit is faster than a digital circuit if the operations are done by the laws of circuits and the signal-processing chain is short (since ADCs are not part of the signal-processing chain in the analog domain) [149]. *The basic multiplication and summation operations can be realized very efficiently in the analog domain using memristor crossbar arrays [150].* The best of both analog and digital circuits can be taken to investigate a mixed analog-digital hardware/software co-designed system, where the analog part is used as a pre-stage to identify interesting cases that will be worked upon in detail by the complex cascaded digital system. Specifically, such a system consists of processing the data in two domains, first energy-efficient but less accurate 'Analog' stage, followed by the less energy-efficient albeit more accurate 'Digital' stage. Digital system consisting of FPGA is triggered only when the analog system's output goes high (i.e., positive indication of anomaly). Furthermore, in the analog domain, all the available sensor data is first grouped into M groups. Data/signals in each group can be first compressed using Analog Joint Source Channel Coding [151] by dedicated analog circuits to obtain the low-dimension compressed sensing data. These M compressed signals are then fed to *analog memristor crossbar array* [150], which implements a neural network in the analog domain using memristor devices. Analog memristor crossbar array hardware realizes efficiently multiplication and addition operations, leveraging the Ohm's law and the Kirchhoff's Current Law (KCL) of circuits to provide ultra-low-power and high-speed realizations. This analog sensing system has very low false negatives (i.e., it does not miss any positive detection), but might not be good with respect to false positives. This is when the digital system is activated. Such digital system, in fact, does not run continuously but only when a detection is triggered by the analog system. Hence, the overall system conserves power via: (i) not running the FPGA inference continuously, which is of the order of few mW; (ii) compressing the signals via AJSCC so as to work in the compressed domain; (iii) running inference in the analog domain as there is no need to use power-hungry ADCs. Hence, any false positives detected by the analog system are reprocessed in the digital system for accurate detection. The digital system is also designed to perform advanced operations such as: (i) quantification/classification of anomalies; (ii) retraining the neural networks so as to personalize the anomaly detection specific to the UAV. *Note that the digital realization of ML does not have computation constraints, therefore is suited for algorithms that cannot be efficiently realized by analog circuits.*

Fourthly, the proposed models can be tested on (i) more crash/attack scenarios e.g., partially broken but functional propellers, cyber attacks, bird hits (simulated via abruptly pulling the UAV using a rope), sudden wind gusts (simulated via subjecting the flying UAV to air from a strong fan), etc.; and (ii) on heterogeneous UAV platforms to assess their overall generalizability potential. Lastly, we only discussed fault detection and diagnosis so far. Real decision-making/intervention methodologies can be investigated to prevent crash/further malfunctioning from the identified misoperation. For example, recovery of the UAV in AirSim simulator when one propeller is broken can be demonstrated such as: immediately after the algorithm detects which propeller is broken, necessary control has to be applied to the corresponding motors, so that the UAV stabilizes on only two or three working propellers.

In summary, we can notice that several new and exciting research directions can be pursued starting from the research techniques presented in this dissertation. The proposed new directions cut across several domains including Machine Learning/Artificial Intelligence, biosensing, smartphone sensing, human-computer interaction, multi-UAV coordination among other domains.

References

[1] UPI.com, "St. Louis firefighters work to contain a fire on the roof of an apartment building during a five alarm fire in St. Louis on July 17, 2012." https://www.upi.com/News_Photos/Features/Firefighters-in-St-Louis/6861/ph4/, 2012.

[2] "Video Demo: Real-time 3D Reconstruction Using Single Drone." https://www.youtube.com/watch?v=NFlVvXDjh9g.

[3] "Video Demo: Multi-Agent Reinforcement Learning based 3D Reconstruction Using Drones." https://www.dropbox.com/s/qv499apo74ulcx2/capstone.mp4.

[4] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles," in *Field and Service Robotics*, 2017.

[5] R. S. Sutton and A. G. Barto, *Reinforcement learning : an introduction.* MIT Press, 1998.

[6] "Video Demo of Agents Trajectories with Same and Different Positions of Targets During Training and Testing." https://www.dropbox.com/sh/vzjs044nn7ek0m0/AADHfab3xX0YUoXmRMSXMM3oa?dl=0, 2020.

[7] V. Ramanishka, Y.-T. Chen, T. Misu, and K. Saenko, "Toward Driving Scene Understanding: A Dataset for Learning Driver Behavior and Causal Reasoning," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[8] M. J. Olsen, K. F. Cheung, Y. YamazakI, S. Butcher, M. Garlock, S. Yim, S. McGarity, I. Robertson, L. Burgos, and Y. L. Young, "Damage assessment of the 2010 chile earthquake and tsunami using terrestrial laser scanning," *Earthquake Spectra*, vol. 28, no. S1, pp. S179–S197, 2010.

[9] G. Chock, L. Carden, I. Robertson, M. Olsen, and G. Yu, "Tohoku tsunami-induced building failure analysis with implications for us tsunami and seismic design codes," *Earthquake Spectra*, vol. 29, no. s1, pp. S99–S126, 2013.

[10] J. Gong and A. Maher, "Use of mobile lidar data to assess hurricane damage and visualize community vulnerability," *Transportation Research Record: Journal of the Transportation Research Board*, no. 2459, pp. 119–126, 2014.

[11] A. G. Kashani, P. S. Crawford, S. K. Biswas, A. J. Graettinger, and D. Grau, "Automated tornado damage assessment and wind speed estimation based on terrestrial laser scanning," *Journal of Computing in Civil Engineering*, vol. 29, no. 3, p. 04014051, 2014.

[12] A. Hatzikyriakou, N. Lin, J. Gong, S. Xian, X. Hu, and A. Kennedy, "Component-based vulnerability analysis for residential structures subjected to storm surge impact from hurricane sandy," *Natural Hazards Review*, p. 05015005, 2015.

[13] M. J. Olsen, F. Kuester, B. J. Chang, and T. C. Hutchinson, "Terrestrial laser scanning-based structural damage assessment," *Journal of Computing in Civil Engineering*, vol. 24, no. 3, pp. 264–272, 2009.

[14] S. C. Yim, K. F. Cheung, M. J. Olsen, and Y. Yamazaki, "Tohoku tsunami survey, modeling and probabilistic load estimation applications," in *Proc. of the International Symposium on Engineering Lessons Learned from the 2011 Great East Japan Earthquake*, pp. 430–443, 2012.

[15] A. G. Kashani and A. J. Graettinger, "Cluster-based roof covering damage detection in ground-based lidar data," *Automation in Construction*, vol. 58, pp. 19–27, 2015.

[16] A. Wharton, *Simulation and Investigation of Multi-Agent Reinforcement Learning for Building Evacuation Scenarios (Master Thesis)*. PhD thesis, St Catherine's College, 2009.

[17] C. Schönauer, E. Vonach, G. Gerstweiler, and H. Kaufmann, "3D Building Reconstruction and Thermal Mapping in Fire Brigade Operations," in *Proc. of the Augmented Human International Conference*, AH '13, (New York, NY, USA), pp. 202–205, ACM, 2013.

[18] E. K. Lee, H. Viswanathan, and D. Pompili, "RescueNet: Reinforcement-Learning-based Communication Framework for Emergency Networking," *Computer Networks (Elsevier)*, vol. 98, pp. 14–28, apr 2016.

[19] RoboCup Federation, "RoboCup Rescue Project." `http://www.robocuprescue.org/wiki/index.php?title=Main_Page`.

[20] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones," *ACM Transactions on Computer Systems (TOCS)*, vol. 32, no. 2, p. 5, 2014.

[21] R. Stevens, C. Gibler, J. Crussell, J. Erickson, and H. Chen, "Investigating user privacy in android ad libraries," in *Workshop on Mobile Security Technologies (MoST)*, Citeseer, 2012.

[22] D. Damopoulos, G. Kambourakis, M. Anagnostopoulos, S. Gritzalis, and J. H. Park, "User privacy and modern mobile services: are they on the same path?," *Personal and ubiquitous computing*, vol. 17, no. 7, pp. 1437–1448, 2013.

[23] E. Toch, "Crowdsourcing privacy preferences in context-aware applications," *Personal and ubiquitous computing*, vol. 18, no. 1, pp. 129–141, 2014.

[24] S. Trepte and L. Reinecke, *Privacy online: Perspectives on privacy and self-disclosure in the social web.* Springer Science & Business Media, 2011.

[25] H. To, G. Ghinita, and C. Shahabi, "A framework for protecting worker location privacy in spatial crowdsourcing," *Proceedings of the VLDB Endowment*, vol. 7, no. 10, pp. 919–930, 2014.

[26] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, "Android permissions: User attention, comprehension, and behavior," in *Proceedings of the Eighth Symposium on Usable Privacy and Security*, p. 3, ACM, 2012.

[27] D. Barrera, H. G. Kayacik, P. C. van Oorschot, and A. Somayaji, "A methodology for empirical analysis of permission-based security models and its application to android," in *Proceedings of the 17th ACM conference on Computer and communications security*, pp. 73–84, ACM, 2010.

[28] W. Enck, "Defending users against smartphone apps: Techniques and future directions," in *Information Systems Security*, pp. 49–70, Springer, 2011.

[29] 2013. German state fines Google for Street View data breach; available at `http://www.reuters.com/article/us-google-privacy-idUSBRE93L0VU20130422`.

[30] S. Mehrotra, C. T. Butts, D. Kalashnikov, N. Venkatasubramanian, R. R. Rao, G. Chockalingam, R. Eguchi, B. J. Adams, and C. Huyck, "Project RESCUE: challenges in responding to the unexpected," in *Electronic Imaging 2004*, pp. 179–192, 2003.

[31] M. Kyng, E. T. Nielsen, and M. Kristensen, "Challenges in Designing Interactive Systems for Emergency Response," in *Proc. of the Conference on Designing Interactive Systems*, DIS '06, (New York, NY, USA), pp. 301–310, ACM, 2006.

[32] V. Balasubramanian, D. Massaguer, S. Mehrotra, and N. Venkatasubramanian, "DrillSim: A Simulation Framework for Emergency Response Drills," in *Intelligence and Security Informatics*, vol. 3975 of *Lecture Notes in Computer Science*, pp. 237–248, Springer Berlin Heidelberg, 2006.

[33] S. D. Ramchurn, F. Wu, W. Jiang, J. E. Fischer, S. Reece, S. Roberts, T. Rodden, C. Greenhalgh, and N. R. Jennings, "Human–agent collaboration for disaster response," *Autonomous Agents and Multi-Agent Systems*, pp. 1–30, 2015.

[34] L. Busoniu, R. Babuska, and B. De Schutter, "A Comprehensive Survey of Multiagent Reinforcement Learning," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 38, pp. 156–172, mar 2008.

[35] M. Lauer and M. A. Riedmiller, "An Algorithm for Distributed Reinforcement Learning in Cooperative Multi-Agent Systems," in *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, (San Francisco, CA, USA), pp. 535–542, Morgan Kaufmann Publishers Inc., 2000.

[36] C. Mariano and E. Morales, "A New Distributed Reinforcement Learning Algorithm for Multiple Objective Optimization Problems," in *Advances in Artificial Intelligence* (M. Monard and J. Sichman, eds.), vol. 1952 of *Lecture Notes in Computer Science*, pp. 290–299, Springer Berlin Heidelberg, 2000.

[37] J. Huang, B. Yang, and D.-y. Liu, "A Distributed Q-Learning Algorithm for Multi-Agent Team Coordination," in *Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference on*, vol. 1, pp. 108–113, aug 2005.

[38] C. Wu, "VisualSFM: A Visual Structure from Motion System." `http://ccwu.me/vsfm/`, 2011.

[39] S. McCann, "3D Reconstruction from Multiple Images," tech. rep., Stanford University, 2015.

[40] "Video Demo: Smartphone-enabled Human-robot Cooperation for Disaster Situational Awareness." `https://www.dropbox.com/s/1fx8odlyirudxuc/Argus_demo_v2.mp4?dl=0`, 2015.

[41] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy Gradient Methods for Reinforcement Learning with Function Approximation," in *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, (Cambridge, MA, USA), pp. 1057–1063, MIT Press, 1999.

[42] F. Niroui, B. Sprenger, and G. Nejat, "Robot exploration in unknown cluttered environments when dealing with uncertainty," in *Proceedings - 2017 IEEE 5th International Symposium on Robotics and Intelligent Sensors, IRIS 2017*, vol. 2018-Janua, pp. 224–229, Institute of Electrical and Electronics Engineers Inc., jan 2018.

[43] N. Basilico and F. Amigoni, "Exploration strategies based on multi-criteria decision making for searching environments in rescue operations," *Autonomous Robots*, vol. 31, pp. 401–417, nov 2011.

[44] Y. Mei, Y. H. Lu, C. S. Lee, and Y. C. Hu, "Energy-efficient mobile robot exploration," in *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2006, pp. 505–511, 2006.

[45] T. Lei and L. Ming, "A robot exploration strategy based on Q-learning network," in *2016 IEEE International Conference on Real-Time Computing and Robotics, RCAR 2016*, pp. 57–62, Institute of Electrical and Electronics Engineers Inc., dec 2016.

[46] J. Zhang, L. Tai, J. Boedecker, W. Burgard, and M. Liu, "Neural SLAM: Learning to Explore with External Memory," *DeepAI*, jun 2017.

[47] F. Niroui, K. Zhang, Z. Kashino, and G. Nejat, "Deep Reinforcement Learning Robot for Search and Rescue Applications: Exploration in Unknown Cluttered Environments," *IEEE Robotics and Automation Letters*, vol. 4, pp. 610–617, apr 2019.

[48] J. Euler, A. Horn, D. Haumann, J. Adamy, and O. V. Stryk, "Cooperative n-boundary tracking in large scale environments," in *2012 IEEE 9th International Conference on Mobile Ad-Hoc and Sensor Systems (MASS 2012)*, vol. Supplement, pp. 1–6, 2012.

[49] J. Delmerico, E. Mueggler, J. Nitsch, and D. Scaramuzza, "Active Autonomous Aerial Exploration for Ground Robot Path Planning," *IEEE Robotics and Automation Letters*, vol. 2, pp. 664–671, apr 2017.

[50] C. Sampedro, A. Rodriguez-Ramos, H. Bavle, A. Carrio, P. de la Puente, and P. Campoy, "A Fully-Autonomous Aerial Robot for Search and Rescue Applications in Indoor Environments using Learning-Based Techniques," *Journal of Intelligent and Robotic Systems: Theory and Applications*, vol. 95, pp. 601–627, aug 2019.

[51] E. Yanmaz, S. Yahyanejad, B. Rinner, H. Hellwagner, and C. Bettstetter, "Drone networks: Communications, coordination, and sensing," *Ad Hoc Networks*, vol. 68, pp. 1–15, jan 2018.

[52] A. Khan, E. Yanmaz, and B. Rinner, "Information Exchange and Decision Making in Micro Aerial Vehicle Networks for Cooperative Search," *IEEE Transactions on Control of Network Systems*, vol. 2, no. 4, pp. 335–347, 2015.

[53] J. Hu, L. Xie, and J. Xu, "Vision-based multi-agent cooperative target search," in *2012 12th International Conference on Control, Automation, Robotics and Vision, ICARCV 2012*, pp. 895–900, 2012.

[54] J. G. C. Zuluaga, J. P. Leidig, C. Trefftz, and G. Wolffe, "Deep Reinforcement Learning for Autonomous Search and Rescue," in *Proceedings of the IEEE National Aerospace Electronics Conference, NAECON*, vol. 2018-July, pp. 521–524, Institute of Electrical and Electronics Engineers Inc., dec 2018.

[55] V. Sadhu, G. Salles-Loustau, D. Pompili, S. Zonouz, and V. Sritapan, "Argus: Smartphone-enabled human cooperation via multi-agent reinforcement learning for disaster situational awareness," in *IEEE International Conference on Autonomic Computing (ICAC)*, 2016.

[56] M. L. Littman, "Markov Games as a Framework for Multi-Agent Reinforcement Learning," in *Proceedings of the Eleventh International Conference on International Conference on Machine Learning*, ICML'94, (San Francisco, CA, USA), pp. 157–163, Morgan Kaufmann Publishers Inc., 1994.

[57] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, (Red Hook, NY, USA), pp. 6382–6393, Curran Associates Inc., 2017.

[58] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[59] J. N. Foerster, Y. M. Assael, N. de Freitas, and S. Whiteson, "Learning to Communicate with Deep Multi-Agent Reinforcement Learning," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, (Red Hook, NY, USA), pp. 2145–2153, Curran Associates Inc., 2016.

[60] M. Tan, "Machine learning : proceedings of the tenth international conference, University of Massachusetts, Amherst, June 27-29, 1993," in *Proceedings of the Tenth International Conference on Machine Learning*, (Amherst, MA), p. 348, Morgan Kaufmann Pub, 1993.

[61] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *International Conference on Learning Representations (ICLR)*, 2015.

[62] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver, and K. Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pp. 1928–1937, JMLR.org, 2016.

[63] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proc. of International Conference on Machine Learning (ICML)*, 2014.

[64] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Commun. ACM*, vol. 60, pp. 84–90, may 2017.

[65] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," 2016.

[66] TensorFlow Core v2.3.0, "Dynamic RNN." `https://www.tensorflow.org/api_docs/python/tf/compat/v1/nn/dynamic_rnn`, 2018.

[67] Niantic, "Pokemon Go." `http://www.pokemongo.com`, 2016.

[68] V. Sadhu, D. Pompili, S. Zonouz, and V. Sritapan, "CollabLoc: Privacy-preserving multi-modal localization via collaborative information fusion," in *IEEE International Conference on Computer Communications and Networks (ICCCN)*, (Vancouver, BC), 2017.

[69] V. Pejovic and M. Musolesi, "Anticipatory Mobile Computing: A Survey of the State of the Art and Research Challenges," *ACM Comput. Surv.*, vol. 47, apr 2015.

[70] G. Salles-Loustau, L. Garcia, K. Joshi, and S. Zonouz, "Don't just BYOD, Bring-Your-Own-App Too! Protection via Virtual Micro Security Perimeters," in *IEEE/IFIP International Conference on Dependable Systems Networks*, jun 2016.

[71] N. O. Tippenhauer, C. Pöpper, K. B. Rasmussen, and S. Capkun, "On the requirements for successful GPS spoofing attacks," in *Proceedings of the 18th ACM conference on Computer and communications security - CCS '11*, (New York, NY, USA), p. 75, ACM Press, 2011.

[72] T. Bao, H. Cao, E. Chen, J. Tian, and H. Xiong, "An Unsupervised Approach to Modeling Personalized Contexts of Mobile Users," in *2010 IEEE International Conference on Data Mining*, pp. 38–47, IEEE, dec 2010.

[73] V. Srinivasan, S. Moghaddam, A. Mukherji, K. K. Rachuri, C. Xu, and E. M. Tapia, "MobileMiner: mining your frequent patterns on your phone," in *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing - UbiComp '14 Adjunct*, (New York, NY, USA), pp. 389–400, ACM Press, 2014.

[74] A. Mukherji, V. Srinivasan, and E. Welbourne, "Adding intelligence to your mobile device via on-device sequential pattern mining," in *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing Adjunct Publication - UbiComp '14 Adjunct*, (New York, NY, USA), pp. 1005–1014, ACM Press, 2014.

[75] K. Farrahi and D. Gatica-Perez, "A probabilistic approach to mining mobile phone data sequences," *Personal and Ubiquitous Computing*, vol. 18, pp. 223–238, jan 2014.

[76] A. Mannini and A. M. Sabatini, "Machine Learning Methods for Classifying Human Physical Activity from On-Body Accelerometers," *Sensors*, vol. 10, pp. 1154–1175, feb 2010.

[77] D. K. Jonathan Feng-shun Lin, "Automatic Human Motion Segmentation and Identification using Feature Guided HMM for Physical Rehabilitation Exercises," in *IEEE/RSJ Int. Workshop Conf. Intelligent Robots and Systems (IROS), Robot. Neurology Rehab.*, 2011.

[78] M. Brand and V. Kettnaker, "Discovery and segmentation of activities in video," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 844–851, 2000.

[79] N. Oliver, B. Rosario, and A. Pentland, "A Bayesian computer vision system for modeling human interactions," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 831–843, 2000.

[80] D. Trabelsi, S. Mohammed, F. Chamroukhi, L. Oukhellou, and Y. Amirat, "An Unsupervised Approach for Automatic Activity Recognition based on Hidden Markov Model Regression," *IEEE Transactions on Automation Science and Engineering*, vol. 10, pp. 829–835, jul 2013.

[81] R. Cilla, M. A. Patricio, J. García, A. Berlanga, and J. M. Molina, "Recognizing Human Activities from Sensors Using Hidden Markov Models Constructed by Feature Selection Techniques," *Algorithms*, vol. 2, pp. 282–300, feb 2009.

[82] L. A. Castro, J. Beltrán, M. Perez, E. Quintana, J. Favela, E. Chávez, M. Rodriguez, and R. Navarro, "Collaborative Opportunistic Sensing with Mobile Phones," in *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication*, UbiComp '14 Adjunct, (New York, NY, USA), pp. 1265–1272, ACM, 2014.

[83] R. Honicky, E. A. Brewer, E. Paulos, and R. White, "N-smarts: Networked Suite of Mobile Atmospheric Real-time Sensors," in *Proceedings of the Second ACM SIGCOMM Workshop on Networked Systems for Developing Regions*, NSDR '08, (New York, NY, USA), pp. 25–30, ACM, 2008.

[84] J. Mantyjarvi, J. Himberg, and P. Huuskonen, "Collaborative context recognition for handheld devices," in *Proc. of the International Conference on Pervasive Computing and Communications (PerCom)*, IEEE, mar 2003.

[85] E. Miluzzo, C. T. Cornelius, A. Ramaswamy, T. Choudhury, Z. Liu, and A. T. Campbell, "Darwin Phones: The Evolution of Sensing and Inference on Mobile Phones," in *Proc. of the International Conference on Mobile Systems, Applications, and Services (MobiSys)*, (New York, NY, USA), ACM, 2010.

[86] H. Polat and W. Du, "Privacy-preserving collaborative filtering using randomized perturbation techniques," in *Proceedings of the Third IEEE International Conference on Data Mining*, (Melbourne, Florida), p. 756, IEEE Computer Society, 2003.

[87] R. Parameswaran and D. M. Blough, "Privacy Preserving Collaborative Filtering Using Data Obfuscation," in *2007 IEEE International Conference on Granular Computing (GRC 2007)*, pp. 380–380, IEEE, nov 2007.

[88] S. Guo, S. Zhong, and A. Zhang, "A Privacy Preserving Markov Model for Sequence Classification," in *Proceedings of the International Conference on Bioinformatics, Computational Biology and Biomedical Informatics - BCB'13*, (New York, NY, USA), pp. 561–568, ACM Press, 2007.

[89] H. X. Nguyen and M. Roughan, "Multi-Observer Privacy-Preserving Hidden Markov Models," *IEEE Transactions on Signal Processing*, vol. 61, pp. 6010–6019, dec 2013.

[90] S. Renckes, H. Polat, and Y. Oysal, "Providing predictions on distributed HMMs with privacy," *Artificial Intelligence Review*, vol. 28, pp. 343–362, dec 2007.

[91] H. Kikuchi, H. Kizawa, and M. Tada, "Privacy-Preserving Collaborative Filtering Schemes," in *2009 International Conference on Availability, Reliability and Security*, pp. 911–916, IEEE, 2009.

[92] W. Ahmad and A. Khokhar, "An Architecture for Privacy Preserving Collaborative Filtering on Web Portals," in *Third International Symposium on Information Assurance and Security*, pp. 273–278, IEEE, aug 2007.

[93] S. Katzenbeisser and M. Petkovic, "Privacy-Preserving Recommendation Systems for Consumer Healthcare Services," in *2008 Third International Conference on Availability, Reliability and Security*, pp. 889–895, IEEE, mar 2008.

[94] J. Canny, "Collaborative Filtering with Privacy," in *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, p. 308, IEEE Computer Society Press, 2002.

[95] Wikipedia, "Plate Notation." `https://en.wikipedia.org/wiki/Plate_notation`.

[96] C. M. Bishop, *Pattern recognition and machine learning.* Springer, 2006.

[97] A. Gruber, W. Yair, and Michal Rosen-Zvi, "Hidden Topic Markov Models.," *AISTATS*, vol. 7, pp. 163–170, 2007.

[98] A. Chopra, "Comparative Analysis of Key Exchange Algorithms in Cryptography and its Implementation," *IMS Manthan (The Journal of Innovations)*, vol. 8, no. 2, 2015.

[99] R. L. Sahita, U. S. Warrier, P. Dewan, and R. S. Narjala, "Executing trusted applications with reduced trusted computing base," 2014.

[100] P. Paillier, "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes," in *Advances in Cryptology — EUROCRYPT '99*, pp. 223–238, Berlin, Heidelberg: Springer Berlin Heidelberg, 1999.

[101] M. Pathak, S. Rane, W. Sun, and B. Raj, "Privacy preserving probabilistic inference with Hidden Markov Models," in *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5868–5871, IEEE, may 2011.

[102] Y. Chon, E. Talipov, H. Shin, and H. Cha, "Mobility prediction-based smartphone energy optimization for everyday location monitoring," in *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems - SenSys '11*, (New York, NY, USA), p. 82, ACM Press, 2011.

[103] G. Heinrich, "Parameter estimation for text analysis," tech. rep., University of Leipzig, 2008.

[104] P. Blunsom, "Hidden Markov Models," tech. rep., Dept. of Computer Science, Utah State University, 2004.

[105] R. Bitar, P. Parag, and S. El Rouayheb, "Minimizing Latency for Secure Distributed Computing," in *IEEE International Symposium on Information Theory (ISIT)*, (Aachen), IEEE, 2017.

[106] Digital Trends, "Volvo to Release Level 4 Autonomous XC90 in 2021." `https://www.digitaltrends.com/cars/volvo-xc-90-level-4-autonomy/`.

[107] P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, and G. Shroff, "LSTM-based Encoder-Decoder for Multi-sensor Anomaly Detection," in *Anomaly Detection Workshop, International Conference on Machine Learning (ICML)*, (New York, NY, USA), 2016.

[108] M. A. Hayes and M. A. Capretz, "Contextual anomaly detection framework for big sensor data," *Journal of Big Data*, vol. 2, p. 2, dec 2015.

[109] A. Capozzoli, F. Lauro, and I. Khan, "Fault detection analysis using data mining techniques for a cluster of smart office buildings," *Expert Systems with Applications*, vol. 42, pp. 4324–4338, jun 2015.

[110] Twitter, "Introducing practical and robust anomaly detection in a time series." `https://blog.twitter.com/engineering/en_us/a/2015/introducing-practical-and-robust-anomaly-detection-in-a-time-series.html`, 2015.

[111] Netflix, "RAD—Outlier Detection on Big Data." `http://techblog.netflix.com/2015/02/rad-outlier-detection-on-big-data.html`, 2015.

[112] M. Toledano, I. Cohen, Y. Ben-Simhon, and I. Tadeski, "Real-time anomaly detection system for time series at scale," in *Proceedings of the KDD: Workshop on Anomaly Detection in Finance*, vol. 71 of *Proceedings of Machine Learning Research*, pp. 56–65, 2018.

[113] D. B. Araya, K. Grolinger, H. F. ElYamany, M. A. Capretz, and G. Bitsuamlak, "An ensemble learning framework for anomaly detection in building energy consumption," *Energy and Buildings*, vol. 144, pp. 191–206, jun 2017.

[114] M. Hasan, J. Choi, J. Neumann, A. K. Roy-Chowdhury, and L. S. Davis, "Learning Temporal Regularity in Video Sequences," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, jun 2016.

[115] A. Taylor, S. Leblanc, and N. Japkowicz, "Anomaly Detection in Automobile Control Network Data with Long Short-Term Memory Networks," in *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pp. 130–139, IEEE, oct 2016.

[116] D. Hallac, S. Bhooshan, M. Chen, K. Abida, R. Sosic, and J. Leskovec, "Drive2Vec: Multiscale State-Space Embedding of Vehicular Sensor Data," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pp. 3233–3238, IEEE, nov 2018.

[117] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal, "Long Short Term Memory Networks for Anomaly Detection in Time Series," in *European Symposium on Artificial Neural Networks*, (Bruges Belgium), 2015.

[118] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *International Conference on Learning Representations*, (San Diego, CA, USA), may 2015.

[119] V. Sadhu, T. Misu, and D. Pompili, "Deep Multi-Task Learning for Anomalous Driving Detection Using CAN Bus Scalar Sensor Data," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1–6, 2019.

[120] K. Hartmann and C. Steup, "The vulnerability of uavs to cyber attacks-an approach to the risk assessment," in *Cyber Conflict (CyCon), 2013 5th International Conference on*, pp. 1–23, IEEE, 2013.

[121] A. J. Kerns, D. P. Shepard, J. A. Bhatti, and T. E. Humphreys, "Unmanned aircraft capture and control via gps spoofing," *Journal of Field Robotics*, vol. 31, no. 4, pp. 617–636, 2014.

[122] Y. Son, H. Shin, D. Kim, Y. Park, J. Noh, K. Choi, J. Choi, Y. Kim, *et al.*, "Rocking drones with intentional sound noise on gyroscopic sensors," in *Proceedings of the 24th USENIX Conference on Security Symposium*, pp. 881–896, USENIX Association, 2015.

[123] M. Rahmati, M. Nadeem, V. Sadhu, and D. Pompili, "UW-MARL: Multi-Agent Reinforcement Learning for Underwater Adaptive Sampling using Autonomous Vehicles," in *ACM International Conference on Underwater Networks and Systems (WUWNet)*, (Atlanta, GA, USA), pp. 1–6, oct 2019.

[124] W. Chen, M. Rahmati, V. Sadhu, and D. Pompili, “Real-time Image Enhancement for Vision-based Autonomous Underwater Vehicle Navigation in Murky Waters,” in *ACM International Conference on Underwater Networks and Systems (WUWNet)*, (Atlanta, GA, USA), pp. 1–8, oct 2019.

[125] V. Sadhu, S. Zonouz, V. Sritapan, and D. Pompili, “HCFContext: Smartphone Context Inference via Sequential History-based Collaborative Filtering,” in *IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pp. 1–10, mar 2019.

[126] V. Sadhu, G. Salles-Loustau, D. Pompili, S. Zonouz, and V. Sritapan, “Argus: Smartphone-enabled human cooperation for disaster situational awareness via MARL,” in *IEEE International Conference on Pervasive Computing and Communications Workshops, PerCom Workshops*, 2017.

[127] V. Sadhu, S. Zonouz, V. Sritapan, and D. Pompili, “CollabLoc: Privacy-preserving Multi-modal Collaborative Mobile Phone Localization,” *IEEE Transactions on Mobile Computing*, pp. 1–13, 2019.

[128] X. Zhao, V. Sadhu, and D. Pompili, “Analog Signal Compression and Multiplexing Techniques for Healthcare Internet of Things,” in *IEEE International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*, 2017.

[129] M. Rahmati, V. Sadhu, and D. Pompili, “ECO-UW IoT: Eco-friendly Reliable and Persistent Data Transmission in Underwater Internet of Things,” in *Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, (Boston, MA, USA), pp. 1–9, jun 2019.

[130] A. Hasan, V. Tofterup, and K. Jensen, “Model-based fail-safe module for autonomous multirotor UAVs with parachute systems,” in *2019 International Conference on Unmanned Aircraft Systems, ICUAS 2019*, pp. 406–412, Institute of Electrical and Electronics Engineers Inc., jun 2019.

[131] Bitcraze, “Crazyflie 2.0.” `https://www.bitcraze.io/crazyflie-2/`.

[132] P. Panitsrisit and A. Ruangwiset, “Sensor system for fault detection identification and accommodation of elevator of uav,” in *SICE Annual Conference 2011*, pp. 1035–1040, Sept 2011.

[133] C. Rago, R. Prasanth, R. K. Mehra, and R. Fortenbaugh, “Failure detection and identification and fault tolerant control using the imm-kf with applications to the eagle-eye uav,” in *Proceedings of the 37th IEEE Conference on Decision and Control (Cat. No.98CH36171)*, vol. 4, pp. 4208–4213 vol.4, Dec 1998.

[134] G. R. Drozeski, B. Saha, and G. J. Vachtsevanos, “A fault detection and reconfigurable control architecture for unmanned aerial vehicles,” in *2005 IEEE Aerospace Conference*, pp. 1–9, IEEE, 2005.

[135] G. Heredia and A. Ollero, “Sensor fault detection in small autonomous helicopters using observer/kalman filter identification,” in *2009 IEEE International Conference on Mechatronics*, pp. 1–6, April 2009.

[136] A. Suarez, G. Heredia, and A. Ollero, "Cooperative sensor fault recovery in multi-uav systems," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1188–1193, IEEE, 2016.

[137] M.-P. Hosseini, H. Soltanian-Zadeh, K. Elisevich, and D. Pompili, "Cloud-based deep learning of big eeg data for epileptic seizure prediction," in *IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, IEEE, 2016.

[138] A. van den Oord, S. Dieleman, and B. Schrauwen, "Deep content-based music recommendation," in *Proceedings of the 26th International Conference on Neural Information Processing Systems*, pp. 2643–2651, Curran Associates Inc., 2013.

[139] T. N. Sainath, B. Kingsbury, G. Saon, H. Soltau, A.-r. Mohamed, G. Dahl, and B. Ramabhadran, "Deep Convolutional Neural Networks for Large-scale Speech Tasks," *Neural Networks*, vol. 64, pp. 39–48, 2015.

[140] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, ACM, 2012.

[141] I. Lenz, R. Knepper, and A. Saxena, "Deepmpc: Learning deep latent features for model predictive control," in *RSS*, 2015.

[142] T. Mikolov, S. Kombrink, L. Burget, J. Cernocky, and S. Khudanpur, "Extensions of recurrent neural network language model," in *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5528–5531, IEEE, may 2011.

[143] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 6645–6649, IEEE, may 2013.

[144] 3DR, "Solo Drone." `https://3dr.com/solo-drone/`.

[145] F. Ordóñez and D. Roggen, "Deep Convolutional and LSTM Recurrent Neural Networks for Multimodal Wearable Activity Recognition," *Sensors*, vol. 16, p. 115, jan 2016.

[146] Microsoft Research, "AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles." `https://github.com/Microsoft/AirSim`.

[147] B. Karis and E. Games, "Real Shading in Unreal Engine 4," in *Physically Based Shading Theory Practice*, 2013.

[148] V. Sadhu, S. Zonouz, and D. Pompili, "On-board Deep-learning-based Unmanned Aerial Vehicle Fault Cause Detection and Identification," in *International Conference on Robotics and Automation (ICRA)*, pp. 1–7, 2020.

[149] S. Ambrogio, P. Narayanan, H. Tsai, R. M. Shelby, I. Boybat, C. di Nolfo, S. Sidler, M. Giordano, M. Bodini, N. C. P. Farinha, B. Killeen, C. Cheng, Y. Jaoudi, and G. W. Burr, "Equivalent-accuracy accelerated neural-network training using analogue memory," *Nature*, vol. 558, no. 7708, pp. 60–67, 2018.

[150] C. Li, Z. Wang, M. Rao, D. Belkin, W. Song, H. Jiang, P. Yan, Y. Li, P. Lin, M. Hu, N. Ge, J. P. Strachan, M. Barnell, Q. Wu, R. S. Williams, J. J. Yang, and Q. Xia, "Long short-term memory networks in memristor crossbar arrays," *Nature Machine Intelligence*, vol. 1, no. 1, pp. 49–57, 2019.

[151] V. Sadhu, S. Devaraj, and D. Pompili, "Towards Ultra-Low-Power Realization of Analog Joint Source-Channel Coding using MOSFETs," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, (Sapporo, Japan), pp. 1–5, may 2019.