UNIVERSITY OF CALIFORNIA
Los Angeles

Deep Learning for Sensing in Complex and Dynamic Environments

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Electrical & Computer Engineering

by

Tianwei Xing

2021

ABSTRACT OF THE DISSERTATION

Deep Learning for Sensing in Complex and Dynamic Environments

by

Tianwei Xing
Doctor of Philosophy in Electrical & Computer Engineering
University of California, Los Angeles, 2021
Professor Mani B. Srivastava, Chair

In recent years there has been an explosion in the availability and use of mobile, wearable, and Internet-of-Things (IoT) devices which generate vast volumes of sensory data. Moreover, deep learning techniques have shown excellent performance on a wide range of tasks such as visual understanding, natural language processing, and speech recognition. Inspired by the success of deep neural networks and the abundance of sensory data, researchers got increasing opportunities to adopt the successful deep learning techniques to various sensing applications. However, real-life sensing tasks usually involve complex spatiotemporal dependencies, which can hardly be captured using only a data-driven approach. The dynamic nature of the sensing environments exacerbates this problem, especially when labeled data are scarce. Unlike visual and natural language data, raw sensor data are opaque to humans, and cannot be labeled retrospectively in a crowdsourced manner. Therefore, training and deploying deep learning models for complex sensing tasks in dynamic scenarios remains a challenge.

In this dissertation, we first focus on the problem of complex event detection over heterogeneous sensory data. To address this problem, the designed system requires not only the perception ability for extracting informative and useful features from raw data, but also the reasoning ability for mining and analyzing the dependencies between the higher-level concepts. We propose DeepCEP, a neural-symbolic framework that combines the power of deep learning models and Complex Event Processing (CEP) engines. DeepCEP encodes

prior knowledge provided by the users to help make effective inferences over the long-term, state-based complex events. To enable the training of this neural-symbolic system, we introduce Neuroplex, which learns from scratch efficiently in complex event settings, under the guidance of high-level prior knowledge. Compared with mainstream deep learning models, Neuroplex reduces the data annotation requirement by 100 times and speeds up the learning process by four times.

Furthermore, we propose the generalized framework DeepSQA for flexible inferencing over heterogeneous sensory data. Given a sensory data context and a task defined on the runtime as natural language questions about the data, DeepSQA can provide an accurate natural language answer. In addition to the DeepSQA, we create SQA-Gen, a software framework for generating SQA datasets using labeled source sensory data, and also generate OppQA with SQA-Gen for benchmarking different SQA models.

Lastly, we investigate the transferability and adaptability of deep learning models under dynamic environments. We propose RecycleML, the first unified framework that transfers knowledge among deep learning models of IoT devices with different input modalities, deployments, or tasks. Using human activity recognition as a case study, over our collected CMActivity dataset, we observe that RecycleML reduces the amount of required labeled data by at least 90% and speeds up the training process by up to 50 times.

The dissertation of Tianwei Xing is approved.

Kai-Wei Chang

Jonathan Kao

Gregory J. Pottie

Mani B. Srivastava, Committee Chair

University of California, Los Angeles

2021

To my parents...

whose countless support and unconditioned love remain with me always

Table of Contents

List of Figures

List of Tables

Acknowledgments

This work described in the dissertation would not have been possible without the encouragement, support, and advice from a number of people at UCLA and otherwise. I want to thank my family, teachers, friends, and all who helped me through my Ph.D. both professionally and personally.

First of all, I would like to thank my advisor Mani Srivastava for all the guidance and support throughout my graduate study. His insightful ideas, a keen understanding of technologies, and great attention to detail are instrumental in shaping my research work and my future career. He provided me the mental support when I needed it the most, and helped me stay the course whenever I was deviating from the road. I greatly appreciate all his contribution of time, energy, and ideas to make my Ph.D. study meaningful, fruitful, and enjoyable. He is undoubtedly a role model to me as a successful scholar and a dedicated professor. I am forever thankful for his unconditional support in both my study and life.

I would like to thank my thesis committee members, Professor Kai-Wei Chang, Professor Jonathon Kao, and Professor Gregory J. Pottie, for the time and effort in helping me improve my research work and this dissertation. Their visionary suggestion and questions on my prospectus, oral examination, and final dissertation helped me gain a better understanding of my research from different perspectives.

My sincere thanks also go to all of my colleagues and collaborators. I am grateful to Dr. Jeng-Hau Lin and Dr. Ritchie Zhao for being my first collaborators and companions. I am also incredibly thankful to my collaborators within DAIS-ITA: Prof. Federico Cerutti, Dr. Lance Kaplan, and Prof.Alun Preece, who have provided insightful suggestions and help on my research of complex event detection and neural-symbolic AI system. A special thanks to my former and current lab mates at NESL, Dr. Luis Garcia, Dr. Supriyo Chakraborty, Dr. Bharathan Balaji, and Sandeep Singh Sandha. I enjoyed the collaboration we had, and I learned a lot from it. Furthermore, I would like to thanks all NESLites who helped and

supported me both in research and life, and who has made our lab a warm and pleasant place for study and research.

Most importantly, I could never finish my Ph.D. study without the support of my family. I want to thank my parents (Yi Song and Cunen Xing) for their unconditional love, unfailing help and guidance. They offered me much advice and encouragement, which was a great source of comfort. I hope I make you proud. To my beloved wife Qi Song: Thank you for your patience, generous support, and encouragement during the most critical stages of my life. I would not have been able to overcome the challenges and stress without your help.

This thesis is based upon work supported in part by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001, by the National Institutes of Health under award #U154EB020404, by the National Science Foundation under award #1636916, and by the CONIX Research Center, one of six centers in JUMP, a Semicon-ductor Research Corporation (SRC) program sponsored by DARPA. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the funding agencies. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

Vita

2010 – 2014	B.S. in Electronics and Information Engineering, Zhejiang University, China.
2014 – 2016	M.S. in Electrical Engineering, University of California, Los Angeles (UCLA).
2017 – 2018	Teaching Assistant, Electrical and Computer Engineering Department, UCLA
2018	Research Intern in Samsung Research America.
2020	Research Intern in IBM Research
2016 – 2021	Graduate Student Researcher, Electrical & Computer Engineering, University of California, Los Angeles (UCLA).

Publications

Tianwei Xing, Luis Garcia, Federico Cerutti, Lance Kaplan, Alun Preece, and Mani Srivastava. "DeepSQA: Understanding Sensory Data via Question Answering." Accepted to The 6th ACM/IEEE Conference on Internet of Things Design and Implementation (IoTDI 2021)

Tianwei Xing, Luis Garcia, Marc Roig Vilamala, Federico Cerutti, Lance Kaplan, Alun Preece, and Mani Srivastava. "Neuroplex: learning to detect complex events in sensor networks through knowledge injection." In Proceedings of the 18th Conference on Embedded Networked Sensor Systems, pp. 489-502. 2020.

Tianwei Xing, Marc Roig Vilamala, Luis Garcia, Federico Cerutti, Lance Kaplan, Alun Preece, and Mani Srivastava. "DeepCEP: Deep complex event processing using distributed

multimodal information." In 2019 IEEE International Conference on Smart Computing (SMARTCOMP), pp. 87-92. IEEE, 2019.

Tianwei Xing, Sandeep Singh Sandha, Bharathan Balaji, Supriyo Chakraborty, and Mani Srivastava. "Enabling edge devices that learn from each other: Cross modal training for activity recognition." In Proceedings of the 1st International Workshop on Edge Systems, Analytics and Networking, pp. 37-42. 2018.

Jeng-Hau Lin, **Tianwei Xing**, Ritchie Zhao, Zhiru Zhang, Mani Srivastava, Zhuowen Tu, and Rajesh K. Gupta. "Binarized convolutional neural networks with separable filters for efficient hardware acceleration." In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, pp. 27-35. 2017.

Vilamala, Marc Roig, Harrison Taylor, **Tianwei Xing**, Luis Garcia, Mani Srivastava, Lance Kaplan, Alun Preece, Angelika Kimmig, and Federico Cerutti. "A Hybrid Neuro-Symbolic Approach for Complex EventProcessing." arXiv preprint arXiv:2009.03420 (2020).

Tomsett, Richard, Amy Widdicombe, **Tianwei Xing**, Supriyo Chakraborty, Simon Julier, Prudhvi Gurram, Raghuveer Rao, and Mani Srivastava. "Why the failure? how adversarial examples can provide insights for interpretable machine learning." In 2018 21st International Conference on Information Fusion (FUSION), pp. 838-845. IEEE, 2018.

Cerutti, Federico, Moustafa Alzantot, **Tianwei Xing**, Daniel Harborne, Jonathan Z. Bakdash, Dave Braines, Supriyo Chakraborty et al. "Learning and reasoning in complex coalition information environments: a critical analysis." In 2018 21st International Conference on Information Fusion (FUSION), pp. 1-8. IEEE, 2018.

CHAPTER 1

Introduction

Smart devices are becoming quite ubiquitous in our lives. Smartphones and tablets are essential devices used for work and pleasure; wearables like smartwatches and bands are broadly used to collect user health data; smart speakers and cameras serve as bridges between users and the ambient environment. Equipped with multimodality sensors and connected with the network, these edge devices are designed to perform intelligent sensing tasks and help make human lives much more convenient.

In recent years, with the explosion in the availability and use of mobile, wearable, and IoT devices, huge volumes of heterogeneous sensory data have been generated. Moreover, advancements in deep learning have led to neural network models that can process raw sensory data with a fantastic performance on various tasks, such as object detection, human activity classification, speech recognition, and audio event detection, beyond the capabilities of even human experts. Most of these tasks aim to seek high-level knowledge about environment status or user behaviors from the multitude of low-level sensor readings. The neural networks are trained explicitly for a task, and integrate both feature extraction and classification processes into a single end-to-end pipeline.

1.1 Challenges and motivations

Albeit showing excellent performance on well-defined and simple tasks, the existing deep learning methods are insufficient to address complex tasks in real scenarios, which usually involve longer temporal and more extensive spatial dependencies. This requires the deep learning models to have both the ability to process unstructured raw data, and the ca-

pability to reason about high-level relationships. For instance, it is crucial to detect the unsanitary operations in the hospitals, where a nurse may forget the disinfection process between processing different patients; and law enforcement needs to detect suspicious activities like someone leaving an unattended bag in sensitive areas, such as the airport. One common feature of these complex tasks is that they are state-based events, which span over large spaces and long periods. The useful information of raw data is usually sparse, which needs to be extracted and then analyzed by complex reasoning logics. Nowadays, a tremendous amount of time and human effort has been spent on doing these complex tasks. The current deep-learning-based systems are having a hard difficulty addressing these problems automatically in an end-to-end manner.

Moreover, training deep learning models for complex event detection is also challenging. Deep neural networks are known for their data-intensive nature. As the problem complexity increases, the size of the required training datasets grows significantly. However, it is impossible to capture and collect real complex event datasets of significant scale as these complex events rarely happen in nature.

Additionally, most of the existing neural networks processing sensory data from IoT devices are specifically trained for a limited number of predefined tasks. Consequently, because of the black-box nature of neural network models, the users cannot obtain any information about the data other than the predefined high-level labels. If new tasks are introduced, users need to collect and annotate the new datasets and repeatedly perform the resource-consuming training process. This problem is exacerbated in complex task scenarios, where the training effort is significantly greater because of the complicated model structure and immense dataset size. Furthermore, except for data such as images and audio, which can be presented and naturally perceived by the highly evolved human brain, IoT devices also collect a broad spectrum of informative sensory data that humans cannot understand easily. For instance, the sensory data from devices like inertial sensors are nothing more than a series of numbers forming waveforms. Due to the lack of appropriate vocabulary, humans usually have difficulty associating these low-level readings to a high-level symbolic

concept. Because of these reasons above, it is necessary to find a way to enable flexible inferencing on the sensory data collected by a network of IoT devices. On the one hand, this would allow the users to define tasks during the runtime, without training new models; on the other hand, users could understand and interpret the opaque sensory data by making various inferences from different dimensions.

On deploying deep learning systems in dynamic and evolving environments, adaptability is of great importance. Recent research has shown that, when the deployment environment's data distribution is different from the training environment, deep learning models suffer colossal performance degradation. For example, change of point of view for cameras and change of body position for wearable devices could introduce the domain shift. This performance drop caused by domain shift needs to be addressed by domain adaptation, where a critical challenge is making the model detect the covariate shift automatically and perform adaptation without human intervention. One extreme case of domain shift is the change of input modalities. For instance, a visual model is first trained on a video dataset for human activity classification, and then deployed in an environment where the audio data is the only input source. This example is a pretty common scenario in our real-life since IoT sensors can get occluded by obstacles or get disabled for privacy concerns.

The capabilities and adaptabilities of deep learning systems in complex and dynamic sensing environments raise a big concern against the continuing adoption of deep learning in the context of ubiquitous IoT devices.

1.2 Contribution and organization

Motivated by the problems above, this dissertation investigates how to efficiently and effectively deploy deep learning systems for IoT devices, especially in complex and dynamic scenarios, while maximizing their performance and cutting down the resource consumption. In this section, we summarize the contributions of the following chapters in this dissertation.

- **Chapter 2:** In this chapter, we focus on building an inference framework for complex

event detection in real-time. With this goal, we propose DEEPCEP, a neural-symbolic framework that integrates the concepts of deep learning models with complex event processing engines to make inferences across distributed, multimodal information streams with complex spatial and temporal dependencies. DEEPCEP utilizes deep learning to detect primitive events. A user can define a complex event to be detected as a particular sequence or pattern of primitive events as well as any other logical predicates that constrain the definition of such an event. The integration of human logic with deep learning models not only increases robustness and interpretability, but also dramatically reduces the amount of training data required. Further, we demonstrate how the uncertainty of a model can be propagated throughout the complex event detection pipeline. Finally, we enumerate the future directions of research enabled by DEEPCEP. We also detail how an end-to-end training model for complex event processing with deep learning may be realized.

- **Chapter 3:** In this chapter, we extend the inference framework DeepCEP to enable training. We present Neuroplex, a neural-symbolic framework that learns to perform complex reasoning on raw sensory data with the help of high-level, injected human knowledge. Neuroplex decomposes the entire complex learning space into explicit perception and reasoning layers, i.e., by maintaining neural networks to perform low-level perception tasks and neurally reconstructed reasoning models to perform high-level, explainable reasoning. After training the neurally reconstructed reasoning model using human knowledge, Neuroplex allows effective end-to-end training of perception models with an additional semantic loss using only sparse, high-level annotations. Our experiments and evaluation show that Neuroplex is capable of learning to efficiently and effectively detect complex events–which cannot be handled by state-of-the-art neural network models. During the training, Neuroplex not only reduces data annotation requirements by 100 times, but also significantly speeds up the learning process for complex event detection by four times.

- **Chapter 4:** To enable flexible inference, we introduce DeepSQA, a generalized Sensory

Question Answering (SQA) framework that aims to enable natural language questions about raw sensory data in distributed and heterogeneous IoT networks. Given a sensory data context and a natural language question about the data, the task is to provide an accurate natural language answer. In addition to the DeepSQA, we create SQA-Gen, a software framework for generating SQA datasets using labeled source sensory data, and also generate OppQA with SQA-Gen for benchmarking different SQA models. We evaluate DeepSQA across several state-of-the-art QA models and lay the foundation and challenges for future SQA research. We further provide open-source implementations of the framework, the dataset generation tool, and access to the generated dataset, to help facilitate research on the SQA problem.

- **Chapter 5:** In this chapter, we focus on the deployment of deep learning models in dynamic environments, where either the input modality or the task is changing. Our approach, called RecycleML, uses cross-modal transfer to accelerate the learning and adaptation of deep learning models across different sensing modalities. Using human activity recognition as a case study, over our collected CMActivity dataset, we observe that RecycleML reduces the amount of required labeled data by at least 90% and speeds up the training process by up to 50 times in comparison to training the edge device from scratch.

CHAPTER 2

DeepCEP: Deep Complex Event Processing Using Distributed Multimodal Information

2.1 Introduction

Big data is useless without a proper framework of analysis [78]. At the sensor level, advancements in deep learning architectures have made incredible progress for inferencing in the context of both unimodal and multimodal data streams [82, 62, 99, 41]. While deep learning models have already proven to perform remarkably in their respective domains, they typically only reason about almost instantaneous temporal features, e.g., the occurrence of a particular object in a given image. The detection of such occurrences–which we refer to as *simple events*–may compose more interesting events that may evolve over long periods of time in different spatial contexts, e.g., a nurse who does not follow a sanitary protocol in one room may lead to endangerment of a patient's health in another room. We refer to such composed events with complex temporal and spatial dependencies as *complex events*.

The definition of a *complex event* is amorphous as complex event processing (CEP) is a mature field of research [33] and has been applied across a variety of domains. Recently, complex *activity* recognition has emerged as its own field of research as well. CEP systems target the problem of processing streams of data to detect complex patterns over long periods of time or across relatively contemporaneous events across multiple streams, while complex activity recognition is generally associated with the classification of events composed of almost instantaneous features, e.g., different forms of human activities, using stateful learning architectures, e.g., recurrent neural networks. And although the current state-of-the-art

CEP systems can detect such composed events over long periods of time, they are typically designed to handle structured data [127]–not the raw data that can be handled by deep learning models.

In this paper, we propose DeepCEP (Sections 3.3 and 2.4), a framework that leverages the power of deep learning models to fuse and process raw data from a distributed set of sensors. The instantaneous inferences made by the deep learning framework are then fed into a state-based *complex event detector*. This event detector allows for the definition of finite state machines that represent a candidate complex event. When a particular pattern of simple events is detected, DeepCEP will then feed this candidate complex event into a final logical layer–which we refer to as a *selector*–to determine if all associated logical predicates for a type of complex event are satisfied. Because of the probabilistic and uncertain nature of deep learning inferencing, the transitions of the finite state machines as well as the associated logical predicates of the complex event's selector engine are specified as ProbLog [17] expressions. This allows the framework to support end-to-end propagation of the uncertainty [95] from the detected simple event to the generated complex event. We evaluate the DeepCEP framework on an exemplary use case study (Section 4.6) to highlight the capabilities of DeepCEP and to establish a precedent for future research directions (Section 4.7).

2.2 Related Work

This section provides the preliminaries necessary to understand the fundamental concepts of DeepCEP. We first provide an overview prior works and concepts related to the detection of complex events, then discuss state-of-the-art approaches to complex event processing (CEP). Finally, we discuss relevant probabilistic logic programming techniques that may facilitate a logical layer for deep learning methods for CEP.

2.2.1 Deep Learning for Complex Events

There has been a body of work on deep learning methods that focus on reasoning about complex events. In particular, several works in video classification have proposed solutions for detecting *complex activities* over short periods of time. Wu et.al [117] fuse spatial information, motion information and temporal clues to classify videos of activities that have a general label. Jiang et.al [54] jointly exploit the feature relationships and class relationships for improving categorization performance. Zhang et.al [129] recognize complex events in video data by fusing multiple semantic cues, including human actions, objects, and scenes. Gan et.al [34] focus on complex event detection in videos while also providing the key evidences of the detection result, which makes the black-box deep learning model intrepretable. They generate a spatial-temporal saliency map and find the key frames in the video which are most indicative to the event. Images are also considered to contain complex events. Xiong et.al [121] developed a method for recognizing events involving interactions among people and objects using static images. Beyond visual classification, some works [28] focus on detecting acoustic events, while several studies [116, 51] use multimodal information to perform classification. Similarly, anomalous event detection [122, 92] utilizes motion features to extract temporal-spatial localization features for complex event detection. Another work [124] has shown how complex events can be detected from unconstrained videos in offline mode.

Although the aforementioned works have shown promising results in their respective domains, they do not have a clear definition for *complex events*. Generally, they use the term to describe events that contains interactions between different elements. Furthermore, these works typically only consider processing information from a single input instead of a distributed set of heterogeneous sensors. Although multimodal data fusion has been explored, they fail to fuse the information at a semantic level so as to provide a clear explanation of the result. Additionally, learning-based models alone cannot learn extremely long temporal dependencies well even with the help of the LSTM [44] structure. They typically reason about events on the order of seconds. Finally, in order to have good performance without

the integration of human logic, learning-based methods necessitate the consumption of large amounts of data with an expensive annotation process. For the purpose of clarity, we provide a formal defintion of *complex events*.

Definition 2.2.1 (Complex event)**.** In this paper, the term *complex event* is strictly defined as a particular pattern or sequence of two or more instances of *simple events* that have spatial and/or temporal dependencies.

2.2.2 Complex Event Processing

Complex Event Processing (CEP) [33, 31, 90] is a mature technique used for processing and analyzing streams of data from multiple sources to detect complex event patterns that suggest complicated situations. CEP systems can be regarded as processes of abstraction: they receive and match low-level events and generate high level complex events. SASE [37, 115] is a CEP system that uses an SQL-like language for filtering, correlation, and transformation of streams of data. Similarly, Cayuga [22] and Siddhi [103] are open-sourced, high performance complex event processing systems with well-defined query languages for event abstractions and pattern detection. Lam et.al [64] extend the CPS modeling language ThingML to support complex event processing. Bizarro et.al [12] propose BiCEP for benchmarking CEP systems.

However, most of the CEP systems are designed for structured data [127], i.e., temperature readings, business transactions, and RFID data. One of the primary reasons is that users cannot define rules or initiate queries on unstructured data which have no particular meaning. Aslam et.al [5] propose a system that handles Internet of Multimedia Things(IoMT) events as native event types in an event processing engine by extending processing languages with the introduction of multimedia analysis operators. They optimized their system to reduce time complexity while maintaining acceptable accuracy. However, this work only focuses on processing simple event streams and queries without considering complex events that involve complex temporal and spatial dependencies. We now provide an overview of our DeepCEP framework that combines deep learning and complex event processing with

end-to-end propagation of uncertainty.

2.2.3 Using Logic for CEP

ProbLog [18, 30][1] belongs to a family of probabilistic logic programming (PLP) languages [16] following Sato's distribution semantics [94]. It extends logic programming by annotating some ground facts with their probability of being true, which generalizes a single program into a distribution over programs that share their rules, but differ in their databases. More specifically, a ProbLog program consists of two parts, a set F of ground probabilistic facts `p::f` where `p` is a probability and `f` a ground atom, and a set R of rules `h :- b1,...,bn` where `h` is a logical atom and the `bi` are literals.[2] While the semantics is defined for countably infinite sets of probabilistic facts, see [94] for details, we restrict the discussion to the finite case in the following. ProbLog considers the ground probabilistic facts as independent random variables, i.e., we obtain the following probability distribution P_F over truth value assignments to sets of ground facts $F' \subseteq F$: $P_F(F') = \prod_{f_i \in F'} p_i \cdot \prod_{f_i \in F \setminus F'} (1 - p_i)$

As each logic program obtained by choosing a truth value for every probabilistic fact has a unique least Herbrand model, P_F can be used to define the *success probability* $P(q)$ of a query q, that is, the probability that q is true in a randomly chosen such program, as the sum over all programs that entail q:

$$P(q) := \sum_{\substack{F' \subseteq F \\ \exists \theta F' \cup R \models q\theta}} P_F(F') = \sum_{\substack{F' \subseteq F \\ \exists \theta F' \cup R \models q\theta}} \prod_{f_i \in F'} p_i \cdot \prod_{f_i \in F \setminus F'} (1 - p_i).$$

Inference in ProbLog is concerned with computing marginal probabilities of queries, i.e., ground atoms, under this distribution, potentially conditioned on a conjunction of evidence atoms. While this is a #P-hard problem in general, ProbLog relies on state-of-the-art knowledge compilation techniques to achieve scalable inference across a wide range of models.

[1] More information on ProbLog, including an open source implementation and an interactive online tutorial, can be found at `https://dtai.cs.kuleuven.be/problog/`.

[2] For the semantics of ProbLog to be well-defined, the set of rules has to have a two-valued well-founded model for each subset of the probabilistic facts: a sufficient condition for this is for programs to be stratified, i.e., have no loops through negation. See [16, 30] for further details.

In [100], Skarlatidis and his team propose using Event Calculus in ProbLog in order to more accurately detect complex events. In order to do so, they assign certain probabilities to the input event. These probabilities can then be used in conjunction with rules in ProbLog that define the start and end of an event. They make use of the fact that, in Event Calculus, multiple initializations and terminations of the same event can happen by increasing or decreasing the probability of the event accordingly to the probabilities of the input events.

2.3 System Overview

The DeepCEP framework is composed of two components: deep event abstractors and an uncertainty-robust CEP engine. Unlike prior CEP engines, we consider the sources that are generating the primitive events as a component of the framework. Figure 2.1 shows the system overview of the DeepCEP framework. The deep data abstractors may consist of one or more edge nodes that are processing raw data from a possibly heterogeneous set of sensors. Each edge node feeds the raw data into an associated deep learning inferencing model that generates a primitive event. The inference results are typically "simple" events, e.g., object categories, human actions, or a person's identity. A *primitive event* is defined as the generated abstraction of a simple event and its attributes. The generated primitive events are then sent to the uncertainty-robust CEP engine.

Prior to runtime, the uncertainty-robust CEP engine processes the complex event (CE) definitions–which are composed of primitive event patterns and logical predicates provided by a user. In this paper, we introduce a CE language that allows for the explicit definitions of both temporal and spatial relationships amongst primitive events. Given these CE definitions, the uncertainty-robust CEP engine creates finite state machine models (FSM) to detect candidate complex event patterns at runtime. These candidate complex event patterns are fed into a selector model that checks the candidate patterns against the provided spatiotemporal logical predicates. If these properties are satisfied, a complex event is generated.

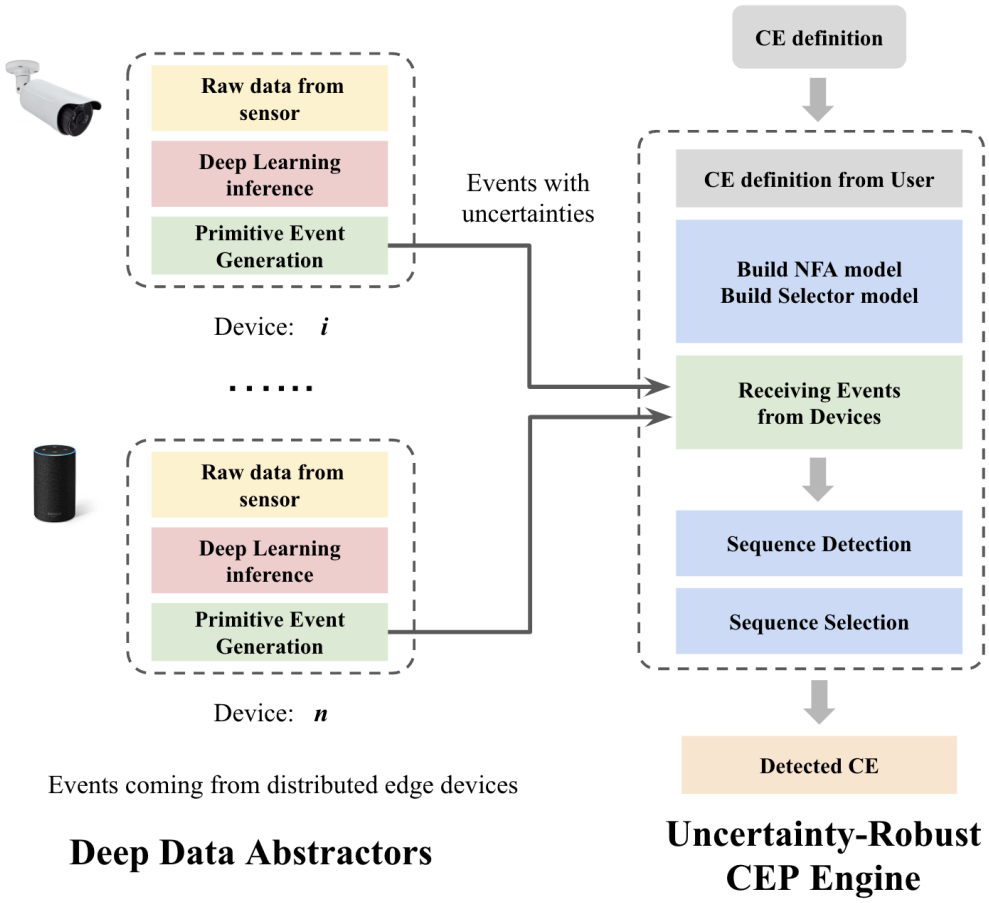

Figure 2.1: System Overview of DeepCEP.

Motivating case study. For the purpose of clarity, we provide a simple yet significant case study that underscores the utility of DeepCEP. The detection of suspicious activities is a complex task of great importance in the physical security domain, e.g., detecting unattended items in a public space, unusual loitering in a sensitive area, house intrusion and theft, or illegal transactions. Such activities are composed of a series of primitive events. For our example, we will focus on the case of detecting an unattended item in a public space. For instance, suppose a subject enters a train station with a luggage bag. After wandering for a period of time, the subject places the luggage inside the passenger lounge. The person then leaves the train station without the luggage bag. The unattended bag is now considered as possibly dangerous and immediate action is required. DeepCEP would need to detect such a pattern of events while minimizing the false positive rate, i.e., DeepCEP should distinguish between suspicious activity and normal subjects who may be momentarily separated from

their bag. The objective of this paper is to design a system that can automatically detect and reason about different complex events, e.g., a person entering a train station, putting down the luggage, and exiting without the luggage. In the next section, we will detail the design of the DeepCEP framework.

2.4 System Design

2.4.1 Deep Data Abstractor

The formulation of the deep data abstractor nodes is application-specific as DeepCEP is intended to work with a heterogeneous set of sensors. However, the primitive events that are sent to the uncertainty-robust CEP should conform to the data structure that defines the event type attributes. The abstract base class for an event type has the following two attributes: `sid` that corresponds to a sensor's identification and `time_stamp` that corresponds to the time stamp of a generated primitive event. The user can provide application-specific implementations of the base class to include other attributes of an event, e.g., a `person_id` for person identification.

In order to generate primitive events, the distributed deep data abstractor nodes collect multimodal data at a constant sampling rate. The data are fed into associated deep learning models to provide semantic meaning. The deep learning model will have a probabilistic output that may model the uncertainty of a detected primitive event. Further, the inference results are generated at the same constant rate and are used to describe the observed situation with a finite result set using the associated event type data structures. As such, the system is a synchronous, state-based system.

However, sending detection results directly to the uncertainty-robust CEP engine would incur a large communication overhead and increase the complexity of detecting complex events due to redundant information. Instead, we use a primitive event generator to produce primitive events given a sequence of states. Every time a new state is processed, the event generator compares it with the previous states to see if there is a "Change of State". For

every state change, a primitive event describing the change is generated and sent to the uncertainty robust CEP engine.

2.4.2 Uncertainty-Robust CEP Engine

The definition of a complex event (CE) is the logical conjunction of an instance of a pattern of primitive events as well as a series of logical predicates associated with the spatiotemporal properties of the primitive events. Each primitive event is assigned an *event type* and contains a set of *event attributes*. Primitive events of different event types are composed in various formats to form complex events. The event attributes describe the event metadata such as the event time, the event location, as well as any problem-dependent attributes, e.g., the identity of a detected person. We denote a primitive event as $A_i(t)$, indicating that such an event is true if and only if the type of event A_i happens at time t. Given this notation, we can now formalize our CEP language.

CEP grammar. The BNF grammar for DeepCEP's CEP engine is defined in Figure 2.2. The <input-title> clause determines the source of a primitive event stream, and the <complex-event-title> clause names the new complex event. The core of a complex event definition resides in the <format-pattern> and <constraint> clauses: the former describes what are the types of primitive events and how are they combined, while the latter gives the constraints of the attributes. The <constraint> clause is optional as the user may only care about detecting a particular pattern. We first detail the semantics of each format pattern option.

Semantics of primitive event pattern detection. Our primitive pattern detection for CE generation is a derivative of the CEP language for SASE [115]. The formats supported in our CEP language are:

$$SEQ(A_1, A_2, ..., A_n)(t) \equiv \{A_1 A_2 ... A_n\}$$
$$PATTERN(A_1, A_k, ..., A_n)(t) \equiv \{A_1(.*)A_k(.*)...A_n\}$$

Where the right-hand side of the equations are the associated regular expressions. The

⟨*complex-event*⟩ ::= ⟨*input-title*⟩ ⟨*complex-event-title*⟩ ⟨*format-pattern*⟩ ⟨*constraint*⟩? **EOF**;
⟨*input-title*⟩ ::= **INPUT :** ⟨*event-stream-source-id*⟩;
⟨*complex-event-title*⟩ ::= **CE :** ⟨*complex-event-stream-id*⟩;
⟨*format-pattern*⟩ ::= ⟨*combo-format*⟩ **: {** ⟨*event-list*⟩+ **}**;
⟨*combo-format*⟩ ::= **FORMAT-SEQ** | **FORMAT-PATTERN** | **FORMAT-PATTERN-WITHOUT**;
⟨*event-list*⟩ ::= ⟨*event*⟩ (**,** ⟨*event*⟩)*;
⟨*event*⟩ ::= ⟨*event-id*⟩ **:** ⟨*event-type-id*⟩;
⟨*constraint*⟩ ::= **CONSTRAINT : {** ⟨*logical-predicate-list*⟩ **}**;
⟨*logical-predicate-list*⟩ ::= logic-expression (**,** logic-expression)*;

Figure 2.2: Simplified CEP Grammar.

SEQ format represents a sequence of consecutive primitive events that occur in a strict order, the $PATTERN$ format represents a sequence of nonconsecutive primitive events. We also support a $PATTERN\_WITHOUT$ format–which is just a $PATTERN$ that excludes a particular set of primitive events–as well as an ANY format–which expresses a CE that could be any type of event within a set. The latter is useful for cases where a CE's composition relies solely on the spatiotemporal properties of any detected events in a space.

Because this is a real-time system, DeepCEP utilizes FSMs to represent and maintain each regular expression. In an ideal scenario, all complex events would be associated with deterministic models. However, because the event attributes are checked after the generation of primitive events, the pattern detector may have non-deterministic behavior. This is due to the fact that patterns with repeated event types may stem from different sensors. As such, we must ensure that the associated FSM takes care of all possible transitions for a given state. The non-deterministic finite state machines that define the semantics of SEQ and PATTERN are given in Figures 2.3 and 2.4, respectively. Let us assume $\Sigma = \{A_1, \ldots, A_n\}$ a finite, non-empty, alphabet.

To handle overlapping events and simultaneous states, the CEP engine needs to have memory storing latest events so as not to be myopic. As such, we adopt the method of keeping a runtime stack. This stack records and maintains the active states at each time point for the latest k events, updating the current state when a new event arrives. A running buffer provides details about each relevant event when the complex event is detected.

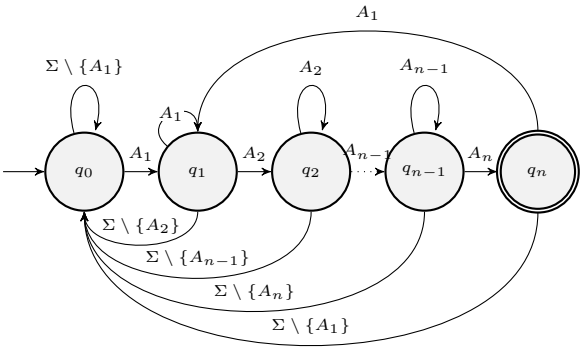

Figure 2.3: Automaton that represents semantics for SEQ detection

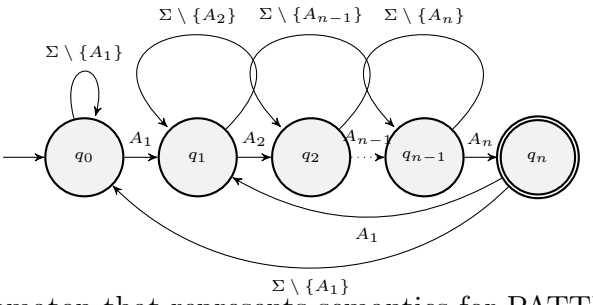

Figure 2.4: Automaton that represents semantics for PATTERN detection

At initialization, the CEP engine first reads a CE definition and creates an FSM model. It then creates and initializes both the event buffer and the event state stack with a size k. Currently, the size number k is a fixed number, where the memory only stores the latest k events with a sliding window for each subsequent event[3]. When a new event arrives, the CEP engine first updates the event buffer by pushing in the new event information and popping out the oldest event. It then updates the event state stack by removing the active states associated with the popped event and updating all the dependencies. If the final state is activated, it means a complex pattern is detected and the CEP engine will output the sequence of events that triggered the final state. This sequence of events will be then fed into the selector model to be checked against the associated logical predicates.

Semantics of logical predicates. As discussed, logical predicates can be defined to filter or *select* candidate complex events. Events attribute constraints are used to describe the temporal and spatial dependencies of complex events, and it is expressed as a set of logical

[3]It is possible to have a memory with a variable size for a fixed time length.

predicates in CEP language. The DeepCEP is intended to support any logic language that can express spatiotemporal properties and has an associated theorem prover. Currently, DeepCEP utilizes ProbLog [17] to express the <logic-expression> clause. This choice is due to the fact that deep learning typically outputs a probabilistic result for a primitive event. The goal of this framework is to propagate the associated uncertainties of a detected primitive event to the uncertainty of a composed complex event. The overall constraint is a Boolean combination (using logical connectives $\vee$ and $\wedge$) of the associated predicates. The complex event is valid only when both the combination format and attribute constraints are satisfied.

At the initialization stage, the Uncertainty-Robust CEP Engine creates the selector model in ProbLog based on the complex event definition created by the user. The associated parser identifies which events it has to take into account as well as the constraints that it has to apply to those events. No code transformation is needed as the user defines the logical constraints in ProbLog. At run time, when a complex pattern is detected, the list of events that can form that event is passed to ProbLog to check if all of the constraints are satisfied and to calculate the probability of the event happening.

Every time a group of events that can potentially create a complex event is detected, the ProbLog Selector first checks if all the constraints are satisfied. If that is not the case, then the probability of the complex event happening is 0. Otherwise, the probability of the complex event happening will be equivalent to the probability of all the events that form it happening at the same time. In order to calculate this, it is necessary to multiply the probabilities of all the individual events happening, using the following formula:

$$P(ComplexEvent) = \prod_{i=1}^{n} P_{Event\_i} \times \prod_{j=1}^{k} \mathbb{1}(Constraint\_j)$$

Therefore, if some of the events are unlikely to have happened, the probability of the complex event happening will be proportionally reduced.

However, ProbLog run comes with a caveat of inefficiency as it needs to consider all the possible ways to calculate the output probability, which requires calculating all possible

combinations. As such, the run time increases exponentially with the number of events that ProbLog has to consider. By providing only the relevant events that are filtered by event generator, computation time is significantly reduced.

2.5 Preliminary Implementation and Evaluation

In this section we present our preliminary implementation and evaluation of DeepCEP[4] based on the motivating case study in Section 3.3. We implemented a compiler in Python for the CEP language using the ANTLR parser generator [1]. Prior to runtime, a complex event definition will be compiled into a pattern detector–which is also implemented and maintained in Python. The associated state machine transitions are implemented in ProbLog as the primitive events are typically associated with a probability. The compiler also generates the associated selector in Python using the ProbLog plugin to enforce the defined ProbLog constraints. The pattern detector is automatically interfaced with the selector while propagating the associated confidence. The implementation for the deep data abstractors as well as the interface between the deep data abstractor and CEP engine is application-specific. We illustrate such an implementation for the unattended bag scenario.

2.5.1 Detecting a Complex Event: *Unattended Bag*

We simulated the aforementioned motivating case study by setting up three cameras (1080p at 240 fps each) in a hallway that are monitoring a "sensitive" area in front of a laboratory door. One camera is monitoring one doorway, another is monitoring the area in front of the laboratory door, and another is monitoring another door. For simplicity of this simulation, one door has been designated as the "Entrance" of the building and the other door has been designated as the "Exit"–with an implication that a person can only enter a hallway through one door and exit through the other.

[4]The preliminary implementation of DeepCEP can be found here https://github.com/nesl/DeepCEP_-DAIS

```
INPUT: camera-feed
CE: unattended-bag
PATTERN: {e1:person_bag, e2:person_bag, e3:bag e4:person}
Constraints:{
 e1.sid = 'Entrance',
 e2.sid = e3.sid = 'Laboratory',
 e4.sid = 'Exit',
 e1.person_id = e2.person_id = e4.person_id}
```

Figure 2.5: Complex Event definition for unattended bag scenario

In this scenario, we define an unattended bag event as: 1) a person enters a building with a bag; 2) the same person is detected in front of the laboratory with a bag; 3) a bag (without a person) is detected in front of the laboratory; and 4) the same person is detected exiting the building. Furthermore, a primitive event for this scenario is the concatenation of all relevant detected objects in a frame. For instance, the detection of a person and a bag would include both detected objects and have an associated label such as `person&bag`.

Given these assumptions, we define the complex event for an unattended bag as shown in Figure 2.5. The `PATTERN` clause defines the order of the four primitive events that compose our complex event, and the `CONSTRAINT` clause specifies the ProbLog expressions that act as logical constraints for the primitive event attributes.

For this simulation, we implemented the uncertainty-robust CEP engine as a server in Python that can receive the primitive events from various sources. Each camera monitors the three different locations and feeds a sampled frame into an inference mechanism to generate primitive events. In this scenario, we sampled frames at a rate of 2 Hz[5]. We used the YOLOv3 [89] object detection model to give us the label of detected objects[6]. It is important to note that the generated primitive events can be structured data that support

[5]We chose a sampling rate of 2 Hz for the frames based on empirical analysis of the execution time of the inference mechanism on various edge devices

[6]This model can also provide the location of an object in a frame, but was not used by our complex event definition.

the specification of arbitrary event attributes. For instance, although it is not necessary for our scenario, the inference model and associated event attributes may support the assignment of unique person identification.

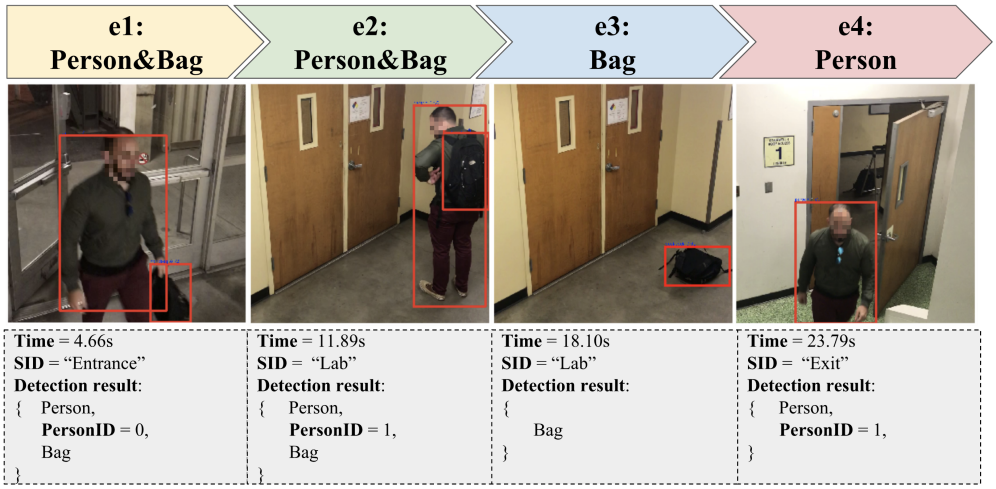

Figure 2.6: DeepCEP evaluated on unattended bag simulation

As discussed, the server maintains an FSM in Python whose transitions are probabilistic based on the detected objects. As such, the states of the FSM are maintained in Python while the probabilistic transitions are maintained in ProbLog. For the pattern detector, the FSM has 5 states, and the event stack stores the latest 10 events [7]. Figure 2.6 summarizes the types of events considered in our scenario as well as sample frames for each case.

The selector model is implemented in ProbLog to filter patterns while propagating the *confidence* of a detected complex event. The ProbLog code for the selector in this example is automatically generated from the CE definition into what we see in Listing 1. The code first defines the events that compose the complex event (lines 2 to 5). It then checks that the order of the events is correct (line 6). Finally, it checks that all the conditions defined by the user are fulfilled (lines 7 to 12).

This implementation allows us to propagate the confidence of a detected event from the object detection all the way to the fully composed complex event. It does so by using the confidence scores of the input events to calculate the confidence of the complex event

[7] We chose an event buffer size of 10 that was sufficient based on empirical analysis.

```
1  ce_event :-
2      event(T1, person_bag, Sid1, Person_id1),
3      event(T2, person_bag, Sid2, Person_id2),
4      event(T3, bag, Sid3, Person_id3),
5      event(T4, person, Sid4, Person_id4),
6      T1 < T2, T2 < T3, T3 < T4,
7      Sid1 == 1, % Where id 1 is "Entrance"
8      Sid2 == Sid3,
9      Sid3 == 2, % Where id 2 is "Laboratory"
10     Sid4 == 3, % Where id 3 is "Exit"
11     Person_id1 == Person_id2,
12     Person_id2 == Person_id4.
13 query(ce_event).
```

Listing 1: Example of ProbLog Selector

happening. *Uncertainty* can also be measured in the same way using uncertainty-aware deep learning models.

2.5.2 Discussion and Future Work

The DeepCEP framework is a work in progress, and although the example above works well in theory, there are several limitations and future research directions.

Timing window of complex events. For the previous CE definition, if a bag is left at the laboratory, and a person is detected leaving without a bag, it is an "unattended item" CE. However, there are scenarios where innocent people may forget to take their bag with them for a brief period of time and consequently trigger the detection of an unattended bag. One way to avoid such false alarms is to add a clock function in our CE language. In the CE definition, we could add the constraint that a bag is unattended if it is left for longer than 10 minutes after the person leaves.

Uncertainty of a primitive event. In the current system, the confidence scores of primitive events are determined only by the detection result of a single frame, which could be

unreliable at times due to environmental factors such as occlusion. We plan to update the confidence of primitive events using the information from consecutive frames to implement more robust detection. In addition, we would apply uncertainty-aware models[95] to provide a better uncertainty measure.

Uncertainty of time. In real systems, multiple devices are not perfectly time synchronized. A tiny time difference could cause tremendous changes in time-sensitive applications. So the uncertainty of events should not only come from inference results, but also from the timestamps of events. We will combine these uncertainties together and propagate them to the CEP engine in order to calculate the final confidence of complex event.

End-to-end training. Another critical direction is the training of the current DeepCEP pipeline. Right now, the deep learning models used on devices are pre-trained models provided by users. However, these generalized models sometimes do not perform well in personalized environments. As such, it is necessary to update the model parameters during runtime. Further, in order to reduce the tedious overhead and human effort of providing detailed annotations of simple events, we want to train the models with only high-level complex event annotations. This would require the models to learn new labels that have never been seen before. Finally, to maintain the interpretability of our results, the associated logical constraints should also be learned using DeepCEP.

2.6 Conclusion

In this paper, we proposed our DeepCEP framework, which combines the logical reasoning ability of CEP and the inference power of Deep learning models to detect complex events for unstructured, distributed multimodal data. We illustrated how DeepCEP can be instrumented for an application scenario of detecting an unattended bag in a sensitive area. In particular, we highlighted how DeepCEP provides a framework to propagate the uncertainty from a detected simple event to a composed complex event using ProbLog. Finally, we enumerated future research directions for DeepCEP.

CHAPTER 3

Neuroplex: Learning to Detect Complex Events in Sensor Networks through Knowledge Injection

3.1 Introduction

Temporal and spatial relationships are natural elements that occur in human learning tasks such as language, motion, and vision [65]. During the past decade, deep learning researchers have achieved great success simulating human cognitive processes using sensors for associated tasks such as object detection, activity classification, and autonomous driving. While demonstrating excellent performance on different sensing tasks, deep neural networks rely on large volumes of training data. Complex spatial-temporal classification tasks suffer from data scarcity, thus making it challenging to design robust learning frameworks. Further, state-of-the-art frameworks are currently limited to a few thousand-time steps while sacrificing interpretability[29]. For instance, deep learning models can currently be trained to detect whether a nurse is sitting or standing in a video stream. However, it would be intractable to train a model that can detect a nurse who does not follow a sanitary protocol before moving between patients—a task that spans arbitrarily long periods, an arbitrary combination of spaces, and time-dependent constraints, and for which a large dataset would not be available. Intuitively, if a set of deep learning classifiers are utilized to detect the simpler activities that happen on the order of seconds, e.g., the nurse entering a room or washing his hands, a human would be able to identify a logical sequence of events that correspond to a violation.

In this paper, we introduce NEUROPLEX, a neural-symbolic framework that splits the

entire learning space into a *perception* space and a *reasoning* space. For the perception task, it trains deep neural networks to get low-level symbolic concepts; while accepting the injection of symbolic human knowledge for high-level reasoning. The entire model can be trained end-to-end with only high-level annotations, which also alleviates the burden of data annotation. In comparison to prior work that combines symbolic and neural reasoning [73, 109, 112], we focus on injecting symbolic knowledge expressed as finite state machines and logical rules, which capture the complex temporal and spatial dependencies for these complex tasks. Given the hierarchical reasoning approach, we formulate the problem as a complex event processing (CEP) problem as was done in previous works [120]. In this work, we provide end-to-end training as opposed to just the forward path.

To summarize, NEUROPLEX makes these important contributions:

- NEUROPLEX adapts neural-symbolic approaches that combine deep learning perception with semantic logical models to enable end-to-end learning for detecting complex events from raw sensor data streams. To enable learning, NEUROPLEX leverages a differentiable Neurally Reconstructed Logic (NRLogic) model, which is a neural network trained by a logical machine through a knowledge distillation[43].

- NEUROPLEX is able to train itself using only data with sparse, high-level, complex event labels. By propagating gradients through the differentiable NRLogic model, *perception modules* receive feedback from complex events labels and are trained accordingly. The training could happen both at the initialization stage, where a perception module is untrained, as well as during the fine-tuning stage, where a perception module is a pre-trained off-the-shelf model and needs to be fine-tuned to a specific environment. NEUROPLEX also applies a semantic loss on the intermediate symbolic layer, forcing the perception module to generate a reasonable symbolic output to improve the learning performance.

- We evaluate the NEUROPLEX framework on three complex event datasets, and compare its performance with state-of-the-art neural network models and neural-symbolic

methods. Results show that guided by injected human knowledge, NEUROPLEX can effectively and efficiently learn to detect complex events that other approaches cannot handle. It not only improves the training speed by 4X times, but can also achieve superb performance while using only 2 orders of magnitude less training data.

This paper is organized as follow: In section 3.2 we formally define what complex event detection is, and discuss the challenge of learning in complex event detection; Section 3.3 formulates the problem we are solving and gives an overview of NEUROPLEX system; Section 3.4 details the inference and training pipeline of NEUROPLEX, and describes how models are trained in both perception and reasoning module[1]; In section 4.6, we perform a set of experiments on three different complex event dataset, and demonstrate the effectiveness and efficiency of NEUROPLEX, and discuss the current limitations and future directions in section 4.6.6; Section 3.7 lists a set of related works, and section 4.7 concludes the paper.

3.2 Background and Motivation

NEUROPLEX aims to detect complex events with intricate spatial and temporal dependencies across multiple sensors. In this section, we formally define what complex events are and discuss the motivation of NEUROPLEX.

3.2.1 Simple and Complex Events

A *simple event* is an event that happens over a short period of time, which usually can be succinctly described by a single word label or a short phrase [73], and can be captured by a single sensor. Modern deep learning models have shown remarkable performance in detecting simple events. For example, a simple event can be the occurrence of a particular object in a given image (e.g., a truck or a suitcase), an audio segment in a waveform (e.g., a siren or a gunshot), or a specific action/activity performed by the subject in the camera feed or

[1]The data and codes of NEUROPLEX are available at https://github.com/NESL/Neuroplex

IMU sensor trace (e.g., walking or opening a door). Simple events are the atoms composing *complex events*.

Definition 3.2.1 (**Complex event**). In this paper, a *complex event* is strictly defined as a particular pattern or sequence of ≥ 2 instances of *simple events* that have spatial and/or temporal dependencies.

Under this definition, a *complex event* must be composed of multiple *simple events* that may evolve over long periods of time in different spatial contexts with various participants. One important distinction between a *complex event* and a *complex activity* is that, although both of them can be decomposed to a set of atomic events, the composing events of a *complex event* may or may NOT be consecutive in time and space. For example, the "long-jump" sporting event is considered a complex activity that consists of five sub-activities: "standing still", "running", "jumping", "landing", and "standing up". These activities are consecutive and can be captured from a single sensor. An example of *complex event* is a sanitary protocol violation event in a hospital scenario: a nurse could violate the sanitary protocol if she or he processes one patient, and then processes another patient without proper sanitation. In this case, the related events (processing patients and hand-washing) need to be detected and analyzed over a broad temporal and spatial range.

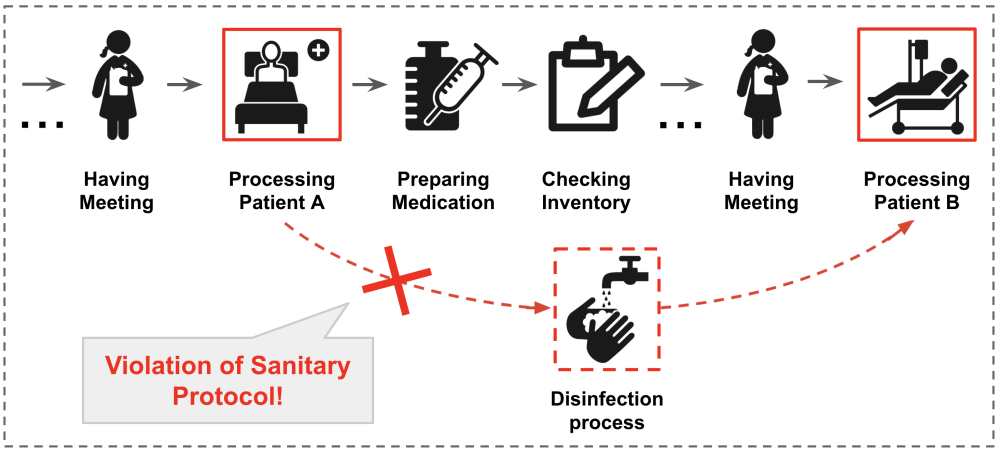

Figure 3.1: Complex Event: Violation of sanitary protocol.

As illustrated in Figure 3.1, these essential events are separated by intermediate, unrelated events such as medication preparation and inventory checking. Although deep learning

models can excel at identifying each composing event, developing a robust inferencing mechanism for the associated complex event is challenging.

Complex Event Detection Systems Although the complex event detection task is challenging, it is of great importance and spans numerous applications in distributed sensor networks, e.g., detecting suspicious activities for security surveillance or the aforementioned protocol violations in hospitals. Recent works have proposed to use neural-symbolic systems [120, 73, 109] to detect complex events. With a hybrid approach, pre-trained neural network models are applied to recognize simple events from multimodal sensory data, and rule-based logic models are used to perform high-level reasoning over a sequence of simple events extracted by deep learning models. The logic models are defined via a set of logic rules, specifying the temporal and spatial relationship between simple events. The logic rules are given by users based on human knowledge.

3.2.2 Learning for Complex Event Detection

Although hybrid neural-symbolic systems have shown to work well on complex event detection tasks, the existing works only focus on the inference stage, where it is assumed that pre-trained deep learning models for detecting simple events are available.

However, this assumption is not practical in real scenarios. First, off-the-shelf deep learning models are trained on standard population-scale datasets and may not perform well when deployed without fine-tuning in a personalized environment. Second, the definition of complex events involves a set of simple events, which may not be included in the output directory of pre-trained deep learning models. For example, when detecting an unsanitary nursing event, if the pre-trained activity classification model can only have a label set of ["washing hands", "walking", "running"] without a "processing patient" label, then the complex event detection system would not work unless a new activity classification model supporting "processing patient" is trained. Third, the performance of neural-symbolic systems relies primarily on the performance of deep learning models for simple events. Since the complex event definition fixes the logic, the final detection result can be totally different

if the deep learning model outputs change. Also, the system's performance degrades when the accuracy of the detection models decreases[73].

Thus, enabling training in complex event detection systems is of great importance. A training pipeline allows the system to fine-tune itself during runtime when deployed in a new environment. Further, the system can train newly added, untrained deep learning models from scratch using only high-level annotations, i.e., it is possible to train an activity classification model supporting a "processing patient" activity only with the high-level "unsanitary nursing event" annotations. High-level annotation significantly reduces the simple event labeling effort for users.

3.3 Problem Formulation and Overview

In this section, we first formalize the complex event detection problem, and then describe NEUROPLEX's design overview.

3.3.1 Complex Event Problem Formulation

Without loss of generality, we consider a sensor streaming data continuously.[2] At every time step i, the sensor generates a triple of raw data information, $d_i = \{x_i, t_i, c_i\}$, where: x_i is raw data of a certain modality captured by the sensor; t_i is the timestamp of the raw data sample; and c_i corresponds to any domain-specific metadata or attributes of the data sample. The attributes can represent any physical features or context related to the piece of data, e.g. the location of the sensor. Depending on the usage scenario, a machine learning model f_θ maps raw information x_i to a set of symbolic classes or values $f_\theta(x_i)$, which we refer to as a simple event, w_i, and where θ is the parameter of model f. A primitive event denoted as $e_i = \{w_i, t_i, c_i\}$, is the abstraction of the raw data sample d_i.

The problem that NEUROPLEX addresses is learning and reasoning about a data stream

[2]Although we are discussing learning of NEUROPLEX in a single sensor scenario, it can be generalized to cases where there exist multiple sensors with different modalities.

of length K. We assume that the length is either user-defined or determined by the limitations of the implemented system. At any given sample point $i \geq$ K, NEUROPLEX should be able to determine if a complex event exists for the previous K samples, i.e., $\{e_{i-K+1},...,e_{i-1}, e_i\}$. Further, we assume that a set of logical rules, ϕ, can be utilized from prior knowledge, e.g., human knowledge, to describe the logical dependencies between primitive events to compose complex events. The logical rules, ϕ, are comprised of both a pattern definition, ω, as well as any additional logical constraints, ψ, for the primitive events, i.e., $\phi = \{\omega,\psi\}$. Formally, we define a complex event as a sequence or pattern of primitive events $\omega = e_1^{CE}, e_2^{CE}, ...e_k^{CE}$ that may have an additional set of logical constraints, ψ. The primitive events $e_i^{CE}, (i = 1, 2, ..., k)$ correspond to the composite primitive events for a complex event, and k is the number of primitive events in this complex event. Each rule in ψ specifies a logical relationship between primitive events composing the complex event. For a complex event to be detected, all of the logical constraints in ψ must be satisfied. Based on this definition, it is possible to have different complex events happening at the same time when they have the same terminal primitive event, i.e., $e_{k_1}^{CE_1} = e_{k_2}^{CE_2}$. It is also possible to have multiple complex events of the same type happening at the same time if more than one set of satisfying primitive events are detected within the time window K.

At training time, the goal of NEUROPLEX is to train a perception model f for each raw sensor data stream, i.e., to learn the optimal set of parameters θ using only streams of raw sensory data $d_i = \{x_i, t_i, c_i\}$ and a complex event label.

3.3.2 Neuroplex Overview

NEUROPLEX uses a hybrid neural-symbolic framework to detect complex events. The deep learning models for detecting simple events compose the *perception module*, and a high-level, human-input logical reasoning machine is referred to as the *reasoning module*, as illustrated in figure 3.2. Given human knowledge injected into the system in the form of logic rules defining complex events, NEUROPLEX enables both a forward inference path and a backward training path. In order to support training, NEUROPLEX tackles the challenge of how to make the

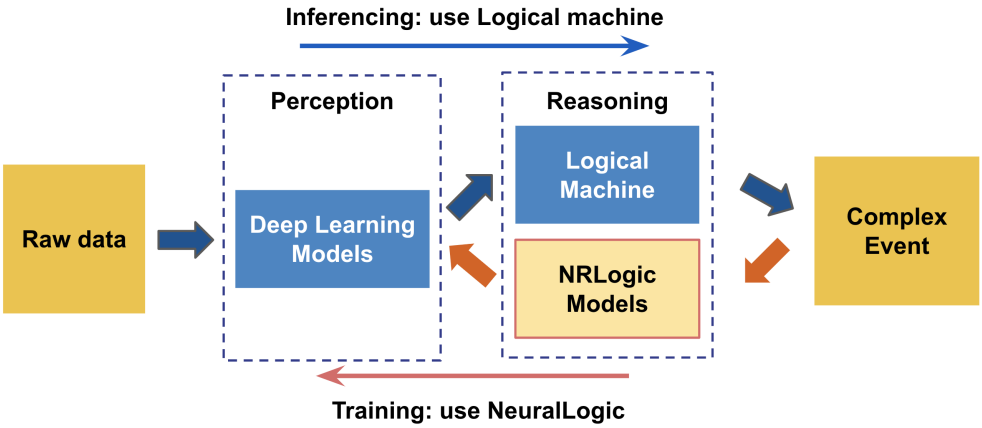

Figure 3.2: NEUROPLEX system with Perception module and dual form Reasoning module.

reasoning module differentiable while maintaining its function, and how to perform effective training over perception module with only high-level complex event annotations. The details of our methodology are described in the next section.

3.4 Neuroplex Design

In this section we detail the design of the NEUROPLEX framework. We first discuss how NEUROPLEX is initialized before describing how the framework is trained.

3.4.1 Neural-Symbolic Initialization of Neuroplex

Before NEUROPLEX can be trained, we first need to provide a mechanism to initialize the learning framework. The initialized structure *without training* would be semantically equivalent to an inference-only, hybrid neural-symbolic framework such as DeepCEP [120]–depicted in Figure 3.3.

3.4.1.1 Reasoning Module Initialization

The reasoning module is initialized by a logical machine in the form of a complex event processing (CEP) engine, which takes user-defined complex event patterns and generates the corresponding machinery for detection. We previously formulated a complex event as a

pattern or sequence, ω, of primitive events, together with some logical constraints, ψ, that impose fine-grained limits on temporal and event attributes. Thus, we decompose the set of logical rules $\phi = \{\omega, \psi\}$ to a finite-state-machine (FSM) model ω and additional logical constraints ψ. The FSM model ω describes the temporal ordering of primitive events of a complex event, and the logical constraints ψ describe the arithmetic logical relations that are not covered by ω. For example, for the complex event of a nurse violating a sanitary protocol (Figure 3.1), the FSM pattern detector, ω, would model the pattern of the nurse's activities to detect two instances of a nurse processing a patient that was not separated by a disinfection process activity. The logical constraints, ψ, would correspond to checking whether the pattern happened within a particular time frame as well as to ensure the primitive events correspond to the appropriate sensors (e.g., the two instances of a patient being processed happened at different locations).

To formalize these definitions, we present a complex event grammar in Backus-Naur Form (BNF) that enables the injection of human knowledge into NEUROPLEX.

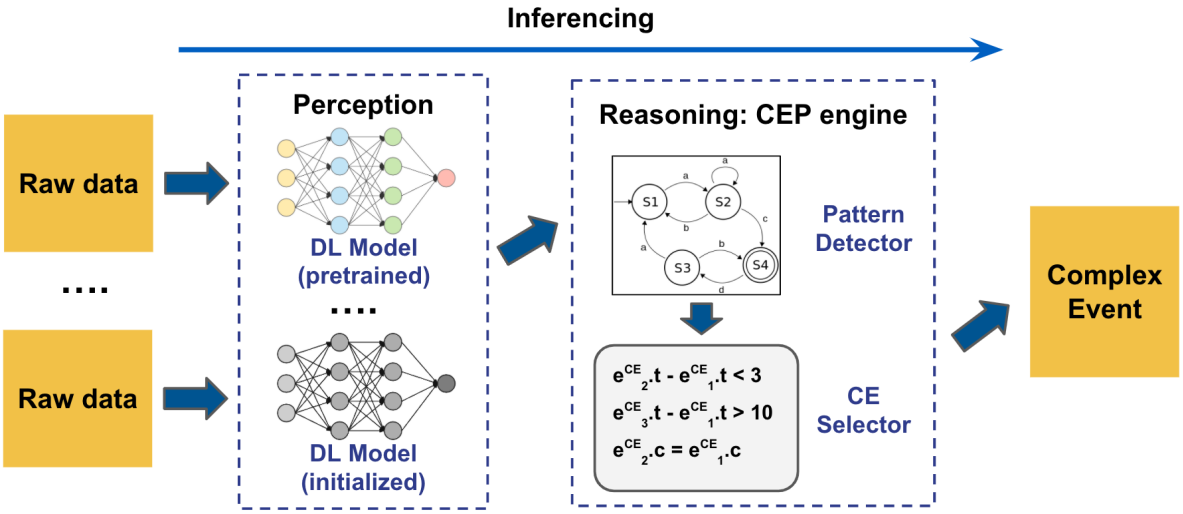

Figure 3.3: NEUROPLEX neural-symbolic initialization. The reasoning module is initialized with human-defined pattern detectors and logical constraints, while the perception module is initialized with a set of deep learning models that may or may not be pre-trained for each raw data source.

CEP reasoning module grammar. The Backus-Naur Form grammar for NEUROPLEX's

⟨*complex-event*⟩ ::= ⟨*input-title*⟩ ⟨*complex-event-title*⟩ ⟨*format-pattern*⟩ ⟨*constraint*⟩? **EOF**;
⟨*input-title*⟩ ::= **INPUT :** ⟨*event-stream-source-id*⟩;
⟨*complex-event-title*⟩ ::= **CE :** ⟨*complex-event-stream-id*⟩;
⟨*format-pattern*⟩ ::= ⟨*combo-format*⟩ **: {** ⟨*event-list*⟩+ **}**;
⟨*combo-format*⟩ ::= **FORMAT-SEQ** | **FORMAT-PATTERN** | **FORMAT-PATTERN-WITHOUT**;
⟨*event-list*⟩ ::= ⟨*event*⟩ (**,** ⟨*event*⟩)*;
⟨*event*⟩ ::= ⟨*event-id*⟩ **:** ⟨*event-type-id*⟩;
⟨*constraint*⟩ ::= **CONSTRAINT : {** ⟨*logical-predicate-list*⟩ **}**;
⟨*logical-predicate-list*⟩ ::= logic-expression (**,** logic-expression)*;

Figure 3.4: Simplified CEP Grammar, whose syntax semantics are significantly adapted from [120]. In this context, the grammar is used to initialize the training framework as opposed to define an explicit CEP engine.

CEP engine is defined in Figure 3.4 and builds upon the grammar of a previous work [120]. We utilize the <input-title> and <complex-event-title> clauses to label the source of a primitive event stream and the associated complex event, respectively. The <format-pattern> clause defines the aforementioned FSM by describing what the candidate primitive event patterns are. The <constraint> clause defines the logical constraints ψ.

Unlike the prior work [120], our grammar's semantics carry a significantly different meaning at training time. The goal of the NEUROPLEX design is to utilize these definitions to bootstrap the training for the reasoning layer of a CEP engine. Without training enabled, the framework would be semantically equivalent to the previous hybrid neural-symbolic inferencing frameworks [120, 77]–as shown in Figure 3.3. In this context, the human-defined logical machine will be utilized as a ground truth at training time. Thus, we define the semantics for how a defined pattern detector processes primitive events as well as how logical constraints are enforced.

Semantics of primitive event pattern detection. The primitive event pattern specified by the <format-pattern> clause is a derivative of the CEP language for SASE [115]. The

formats supported by our CEP grammar are:

$$SEQ(A_1, A_2, ..., A_n)(t) \equiv \{A_1 A_2 ... A_n\}$$
$$PATTERN(A_1, A_k, ..., A_n)(t) \equiv \{A_1(.*)A_k(.*)...A_n\}$$

Where the right-hand side of the equations are the associated regular expressions. The SEQ format represents a sequence of consecutive primitive events that occur in a strict order, while the $PATTERN$ format represents a sequence of nonconsecutive primitive events. At runtime, the defined logical machine will utilize finite state machines (FSMs) to represent each regular expression and maintain the current state of the pattern detector.

When NEUROPLEX initializes the logical machine, the CEP engine first reads a CE definition and creates an FSM model. It then creates and initializes both the event buffer and the event state stack with a size K. Currently, the size number K is a fixed number, where the memory only stores the latest K events with a sliding window for each subsequent event.[3] When a new event arrives, the CEP engine first updates the event buffer by pushing in the new event information and popping out the oldest event. It then updates the event state stack by removing the active states associated with the popped event and updating all the dependencies. If the final state is activated, it means a complex pattern is detected and the CEP engine will output the sequence of events that triggered the final state. This sequence of events will then be fed into the *selector model* to be checked against the associated logical constraints.

Semantics of logical constraints. As discussed, logical predicates can be defined to *filter* or *select* candidate complex events as shown in Figure 3.3. Event attribute constraints are used to describe the spatial dependencies as well as additional temporal constraints of complex events. They are expressed as a set of logical predicates in the CEP language. NEUROPLEX is intended to support any logic language that can express spatial-temporal properties and has an associated theorem prover. Currently, NEUROPLEX utilizes ProbLog [17] to express the <logic-expression> clause. This choice is due to the fact that deep learning

[3]It is possible to have a memory with a variable size for a fixed time length.

typically outputs a probabilistic result for a primitive event. The reasoning framework aims to propagate the associated probability of a detected primitive event to the probability of a composed complex event. The overall constraint is a Boolean combination (using logical connectives $\vee$ and $\wedge$) of the associated predicates. The complex event is valid only when both the *combination format* and *attribute constraints* are satisfied.

When the logical machine is initialized, the CEP Engine creates the selector model in ProbLog based on the user's complex event definition. The associated parser identifies which events to be taken into account as well as the constraints to be applied to those events. No code transformation is needed as the user defines the logical constraints in ProbLog. At run time, when a complex pattern is detected, the list of events that can form the complex event is passed to ProbLog to check if all of the constraints are satisfied and to calculate the probability of the event happening.

3.4.1.2 Perception Module Initialization

As previously discussed, the perception module should have a set of deep learning models that can abstract the raw data for each sensor in the network. The event symbols that are used by the human to define complex event patterns and constraints should be a subset of the label set for the associated perception module, i.e., users can only define reasoning rules based on event labels generated by the deep learning models. We assume that each provided model is initialized either with random weights or with pre-trained weights, i.e., NEUROPLEX can be initialized with off-the-shelf models trained on population-scale data–as depicted in Figure 3.3. However, the pre-trained weights are not necessary for NEUROPLEX's training process.

3.4.2 Neuroplex Training Framework Design

The training pipeline for NEUROPLEX decomposes the original end-to-end learning problem into two sub-problems of perception and reasoning. NEUROPLEX can learn each part separately with the aid of injected semantic knowledge to significantly reduce the learning

space.

When the logic rules of complex events are known, we need to train the perception module to connect the path between raw data and complex events. However, because logic is not differentiable, we first propose a deep learning model called NRLogic, which approximates the function of the original logical machine in the reasoning module. Depending on the application, this NRLogic model can be trained using knowledge distillation [43] with only synthetic data labeled by the logical machine. During this process, only the reasoning model is updated. Once trained, the reasoning module has a dual form of representation: a logical representation and a neural representation. This enables the training of the perception module to be performed in a supervised manner. With the annotation of complex events, we freeze the parameters of the NRLogic model and calculate the gradient of the loss with respect to the deep learning models in the perception module. Additionally, logical constraints can be added to the intermediate symbolic layer between perception and reasoning module, imposing another regularization term for training. In this phase, the reasoning module is only used to propagate the gradients, and its parameters are kept intact to preserve its functionality. Figure 3.5 shows the training pipeline of NEUROPLEX's perception module.

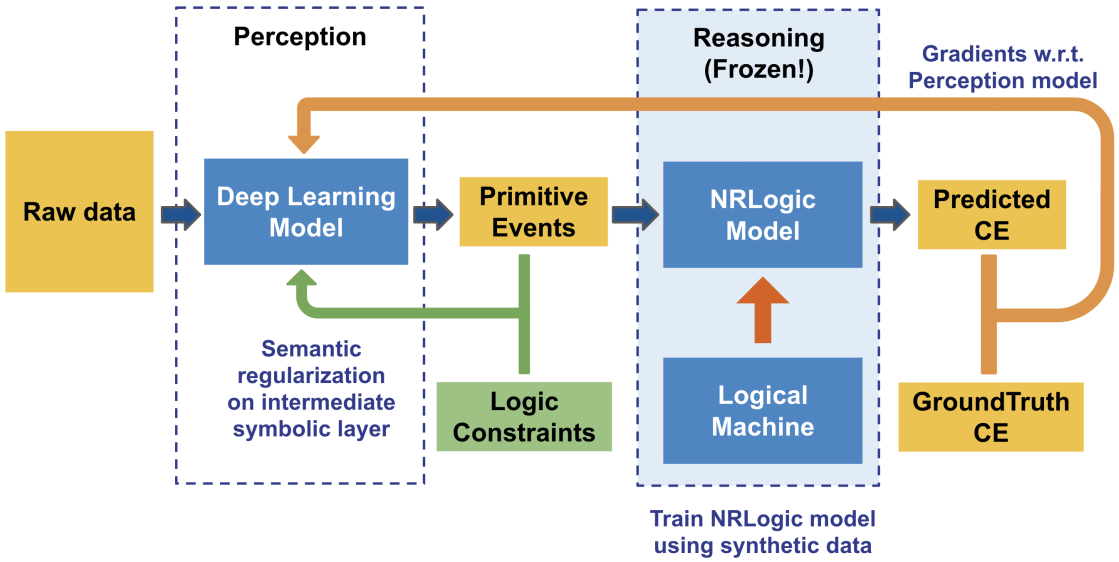

Figure 3.5: Training on NEUROPLEX's perception module.

Training of the NRLogic network. In NEUROPLEX, the intermediate layer between the

perception module and the reasoning module is symbolic. For example, if we want to detect a set of composed complex activities, then the perception module could be an activity classifier, and the reasoning module is the logical machine expressing the pattern and constraints between atomic activities. The corresponding primitive event in the intermediate layer is a classified activity result with the associated timestamp and attributes, $e_t = \{f_\theta(x_i), t_i, c_i\}$. The reasoning module here has both structured input space and structured output space, so we can easily generate data sequence samples, and use logical machine ϕ to get the ground truth annotations y.

Because the logical machine ϕ used in complex event detection contains arithmetic logical constraints ψ and finite state machine ω, we use a recurrent neural network structure $p(E)$ to mimic the reasoning module. The input $E_i = \{e_{i-K+1}, .., e_{i-1}, e_i\}$ denotes all the primitive events happening in the past time window K. The training procedure of the NRLogic model can be regarded as a knowledge distillation process [43]. The logical machine is the teacher—which provides ground truth values to the generated primitive events, and the primitive event sequence and label pairs $\{E_i, y_i\}$ are then used to train the student NRLogic network.

Since the ground truth annotation y_i describes instances of multiple complex events, it is possible to have different complex events occurring multiple times in a given time window. Therefore, we formulate this problem as a multi-label regression problem. The annotation y_i and prediction $p(E_i)$ would be an m-dimensional vector, where m is the number of complex events we are detecting. Each entry y_i^j represents the number of complex events j happening at the current time sample i.

The NRLogic model is trained using a number of synthetic data as a preparation step for the perception module training. Since the dimension of the input primitive event data is not large, the RNN model employed here can be relatively simple, and thus, the training process would not introduce significant overhead. We keep training the NRLogic until it converges.

Training of the perception module. Once the NRLogic model p is well-trained, we can integrate it as a layer in the training pipeline,i.e., we concatenate it with the perception module we need to train. The new integrated model still complies with the structure of

NEUROPLEX, where the reasoning module follows the perception module, but the reasoning module is expressed in its neural network form. In this way, we successfully build a differentiable path between raw data and complex event labels.

Since the NRLogic network is pre-trained to mimic the logical machine, we freeze its weights to preserve its functionality. We then calculate the gradient of the complex event prediction loss with respect to the perception model only, $\partial L(y_t, p(E_t))/\partial\theta$, and use this gradient to update the model. The loss L for the regression task is the mean squared error:

$$L_{MSE} = \frac{1}{N}\sum_{t=1}^{N}(p(E_t) - y_t)^2 \tag{3.1}$$

Because of the frozen NRLogic, the perception model is forced to learn to predict the corresponding event types. After the training finishes, the perception model ideally performs as if trained with fully-annotated event-level data.

Semantic Loss on an Intermediate Layer. One of the most significant features of NEUROPLEX is that it has a symbolic intermediate layer. Since the primitive event e_i in the intermediate layer contains the event type information w_i, which is usually expressed in the form of a softmax score, we can impose an additional semantic loss on it to further facilitate training.

We use the idea from [123]: in a multi-class classification task, a well-trained model should give output with exactly one of the classes being true, and the others being false. With this idea, a semantic loss function is introduced to force the mass of the softmax vector to accumulate for a single class, i.e.,

$$L_{Semantic} = -\log\sum_{i=1}^{m} p_i \prod_{j\neq i}(1 - p_j) \tag{3.2}$$

where m refers to the number of classes in a softmax vector. The intuition here is to minimize the negative log probability of generating a state that satisfies the logical constraints with sampling probabilities equal to the softmax values. When the probabilities of classes are evenly distributed, the loss value would be a large value close to 1, and if only one of the classes has a probability of 1 and the rest being 0, the loss value is equal to 0.

Therefore, training the perception network entails optimizing a combined loss:

$$L = \lambda \times L_{Semantic} + (1 - \lambda) \times L_{MSE} \tag{3.3}$$

and the gradient used to update the network becomes:

$$\frac{(1 - \lambda) \cdot \partial L_{MSE}(y_t, p(E_t)) + \lambda \cdot \partial L_{semantic}(f_\theta(X_t))}{\partial \theta}$$

The λ is the hyper-parameter that controls the strength of semantic regularization.

3.5 Evaluation

In this section, we empirically evaluate the proposed training method for NEUROPLEX on three complex event datasets. The goal is to evaluate whether NEUROPLEX is capable of learning and detecting complex events, and how it performs compared with state-of-the-art deep learning methods and neuro-symbolic methods. We further perform an ablation study to evaluate how it improves training by adding a semantic loss on the intermediate symbolic layer.

NEUROPLEX is implemented using Tensorflow and Keras frameworks, and evaluated on a desktop machine with an Nvidia RTX Titan GPU. We first conduct an evaluation on a synthesized MNIST sequence data to thoroughly analyze the performance of NEUROPLEX's method, while comparing it with a set of strong baselines. Then we test NEUROPLEX on a complex audio event dataset and a complex nursing event dataset, both of which are constructed using real sensory data.

3.5.1 MNIST Sequence Complex Events

To explicitly control the complexity of the logic in complex events, we synthesize our dataset using MNIST [68] digit images which we refer to as the MNIST Sequence Complex Event dataset. This dataset is generated by randomly creating sequences of MNIST images. Each image in an MNIST sequence is assigned with an increasing timestamp t and an attribute

ID c. To get the ground-truth label of every MNIST sequence, we randomly generate a set of logical rules $\phi = \{\omega, \psi\}$, and then apply these rules to the sequence to see if the complex events exist. For instance, a complex event in this context, CE_1, can be a pattern of $\{e_1^{CE_1}$: "1" $\Rightarrow e_2^{CE_1}$: "3" $\Rightarrow e_3^{CE_1}$: "9"$\}$. The primitive events $e_1^{CE_1}$, $e_2^{CE_1}$, and $e_3^{CE_1}$, must happen in a sequential order with the corresponding digits 1, 3, and 9 as the event types. Additionally, temporal and spatial constraints can be added to the complex event definition, e.g., the set of temporal constraints $\psi \equiv e_2^{CE_1}.time() - e_1^{CE_1}.time() < 10s$ and $e_3^{CE_1}.time() - e_1^{CE_1}.time() > 3s$ and spatial constraints $dist(e_1^{CE_1}.location(), e_2^{CE_1}.location(), e_3^{CE_1}.location()) < 100$ limit the complex events to those that satisfy the generated rules.

We say that a complex event CE_1 is happening if a sequence contains the pattern of primitive events $[e_1^{CE_1}, e_2^{CE_1}, e_3^{CE_1}]$—with primitive event $e_3^{CE_1}$ being the last event in the sequence—and if all the logical constraints ψ are satisfied. Each sequence of primitive events is one sample fed to the complex event detection system. The goal of this dataset is to detect the occurrence of different complex events for each raw image sequence.

The complex event detection system has a maximum time window of length K, which means that at most K recent primitive events are considered when making a detection. Thus, we generate the MNIST sequence data with length equals to K. Apparently, as window length K increases, the difficulty of complex event detection increases as well. It is because that not only the system requires a larger memory to store and process the latest events, but also the input space and complex events arrangement increase exponentially. In our experiment, we change the size of window length K in different simulations, to test the robustness of NEUROPLEX as the complex event detection task becomes more difficult.

Experimental setup. In our experiment, we used a LeNet convolution network [68] as the perception model for digit classification, and an LSTM network as the NRLogic model. Specifically, the LeNet is a CNN architecture with two convolution modules (convolution layer + Relu activation + maxpooling) with a 5-by-5 kernel, followed by two fully-connected layers. It achieves about 99.2% testing accuracy when trained on MNIST training data directly and tested on MNIST testing set.

In the NRLogic model, we use a simple network with one LSTM layer with 64 hidden units plus one fully-connected layer to capture the logic of complex events. Since it is a regression model, the output layer has m nodes and linear activations, where m is the number of complex events types. The optimal λ value in function (3) is set to $1e-4$, and we use a grid search to find reasonable parameters. The experiments here use an Adam regularizer with a 0.001 learning rate for training both the NRLogic model and the perception model, and the batch size is set to 256. The NRLogic model is pre-trained on randomly generated data until convergence within 200 epochs.

Performance measure. We evaluated the learning performance by looking at the Mean Absolute Error (MAE) of the complex event prediction. However, this term sometimes cannot reflect the model performance directly. We therefore compare the predicted scores with the ground truth values and calculate the prediction accuracy (Acc) by rounding the prediction numbers:

$$Acc = \frac{\sum_{i=1}^{N}\sum_{j=1}^{m}\mathbb{1}(round(p(E_i)^j) == y_i^j)}{m \times N}$$

The superscript j refers to the j-th entry of the prediction, and m represents the types of complex events. The accuracy is calculated as the average correct prediction rate for all types of complex events on a testing dataset with N samples.

Additionally, for models with a similar structure as NEUROPLEX, we also evaluate the performance of the LeNet on the MNIST testing dataset after training to measure the actual learning performance on the perception model.

Learning comparison with strong baseline methods. We first test the learning performance of NEUROPLEX on an MNIST sequence dataset, as shown in Table 3.2, simulation 1. Here we are considering 4 different complex events happening in a window of 10 primitive events. The logic rules of complex events are randomly generated, while each complex event is composed of two to three primitive events, and the average length of the complex events is 2.8. The number of unique primitive event types is 10, which means that these four complex events cover all the 10 digits in MNIST.

We compared NEUROPLEX with mainstream deep learning baselines:

- **(1) CRNN network:** It has exactly the same structure as the model we are using in NEUROPLEX: CNN+LSTM structure, inspired by [97]. The only difference is that in the CRNN model, human knowledge is not injected, so the LSTM layer is not pre-trained.
- **(2) C3D model:** The C3D network has a similar 3D convolutional structure as [72] to capture temporal dependencies between image frames. The total number of parameters is around 4.1 million, which is much more complex compared with our model with 1.3 million parameters.
- **(3) Neuroplex w/o semantic loss:** We also perform an ablation study by removing the semantic loss imposed on the intermediate layer of NEUROPLEX to see the extent to which it helps training.
- **(4) Oracle:** The oracle method uses a neural-symbolic approach with perfectly pre-trained perception models and 100% correct human-defined logical rules. This method only represents the theoretically best performance we can get, since pre-trained networks tuned to specific environments are usually not available because of the heterogeneity of different domains and different task requirements.

Although NEUROPLEX has human knowledge injected and distilled to NRLogic model, the pre-training overhead is pretty small. This is because the NRLogic model is not complex, and it has symbolic input and output space. The model converges to optimum within a few minutes during the pre-training process.

As shown in Figure 3.6, we train different models on a training dataset of 10K sequences for 200 epochs, and plot the learning curves on 2K validation dataset in terms of MAE. Clearly, NEUROPLEX learns faster than other baselines, and it trains the best final model with a performance close to the oracle approach. The C3D model learns slowly, and it fails to converge to the optimum in 200 epochs. Though having the same structure as NEUROPLEX,

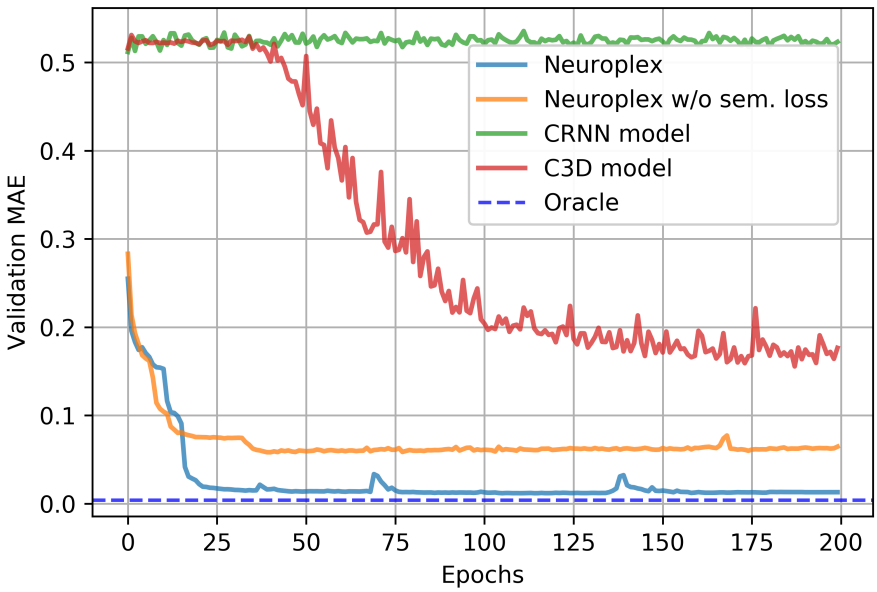

Figure 3.6: Prediction MAE (on the validation set) changes as training progressed in simulation-1

	Oracle	NEUROPLEX	NEUROPLEX (w/o)	CRNN	C3D
Perception Acc	99.19%	**98.87%**	70.55%	10.09%	NA
Validation MAE	0.002	**0.013**	0.065	0.523	0.176
Converted Acc	99.85	**99.39%**	96.02%	69.98%	88.47%

Table 3.1: Performance comparison on MNIST Sequence

the CRNN model struggles to capture the dependencies between raw data and complex events due to the high task complexity. In the ablation study, the model without semantic loss has an inferior performance compared with the original NEUROPLEX, proving that the semantic loss indeed helps training and regularizing the model.

We also measure the performance of different methods in Table 3.1. Clearly, NEUROPLEX could achieve the lowest MAE on the complex event prediction, and the converted accuracy is higher than 99%. The LeNet in the perception module is also well-trained to get an accuracy of 98.87%, which is pretty close to the model trained on the original MNIST training data (99.2%). The CRNN model which has the same network architecture but without human knowledge injected does not capture the complex events well, and the LeNet on the CRNN

could not perform image recognition effectively.

Figure 3.7 shows that, as training progresses, the performance of LeNet is improving with the entire model. Also, since the NRLogic model is frozen and not trainable, it perfectly maintains the functionality of the original logic models. As we expected, the overall complex detection performance is totally dependent on the ability of the perception module.

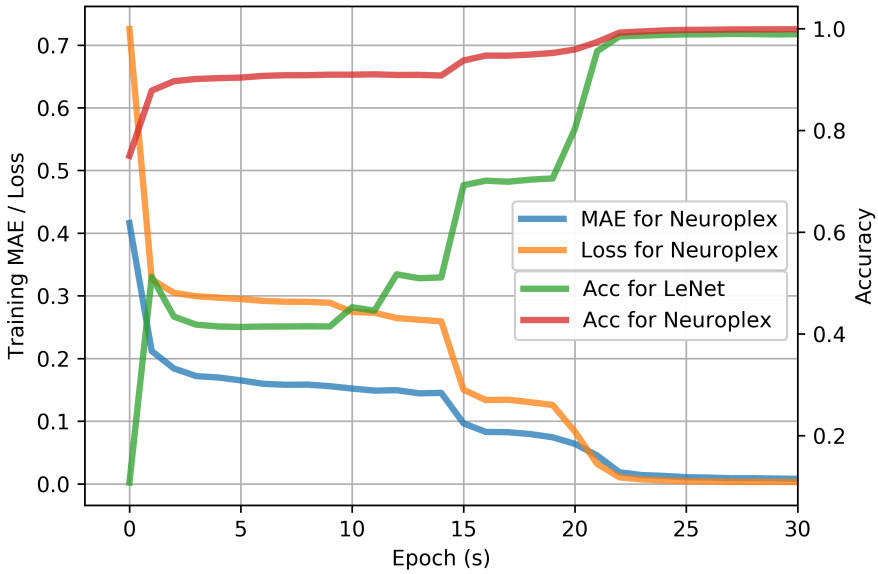

Figure 3.7: The performance of perception model increases as training processed in NEUROPLEX

Comparison with neural-symbolic methods. We performed a preliminary investigation using the state-of-the-art complex event detection method described in [110], which is a neuro-symbolic architecture that combines a neural network—which processes raw data—and logic programming—to express the patterns that define a complex event. The system allows for end-to-end learning using DeepProbLog[77]. However, the probabilistic logic programming aspect of this system makes it quite inefficient in terms of training time. In a preliminary investigation in this direction, while NEUROPLEX takes 5.4 milliseconds in training over a CPU—and even less on GPUs—a DeepProbLog instance is around four orders of magnitude slower. We believe that the flexibility of having a human-understandable and easily manipulable logical regularisation will be valuable for articulated complex event detection rules, but that first requires a coordinated effort of the neuro-symbolic community

to improve the engineering of DeepProbLog.

Learning with a limited amount of training data. To evaluate how NEUROPLEX performs in the scarce data scenario, we synthesize a new complex event setting, as per Table 3.2, simulation 4. In this dataset, the complexity of a complex event is greatly reduced to ensure that all the baseline models can learn. The length of the event window is 3, and three unique events are considered in 5 different complex events which have an average length of 2. We adjust the number of available training data, from only 10 samples to 21K samples, and train all models for 200 epochs.

As shown in Figure 3.8, regardless of the training dataset size, the proposed NEUROPLEX method steadily shows the best performance, especially in the case when training data is very limited. Removing the semantic loss on NEUROPLEX would incur a small performance drop and it again proves the benefits of a symbolic layer and the corresponding logical constraints for training. The CRNN trained from scratch cannot learn and reason about complex events effectively even with 21K data, and the complex C3D model begins to show acceptable performance (greater than 85%) only when the amount of training data is greater than 10K. NEUROPLEX is shown to be robust to the data scarcity problem as it can achieve over 90% accuracy with only 20 data samples.

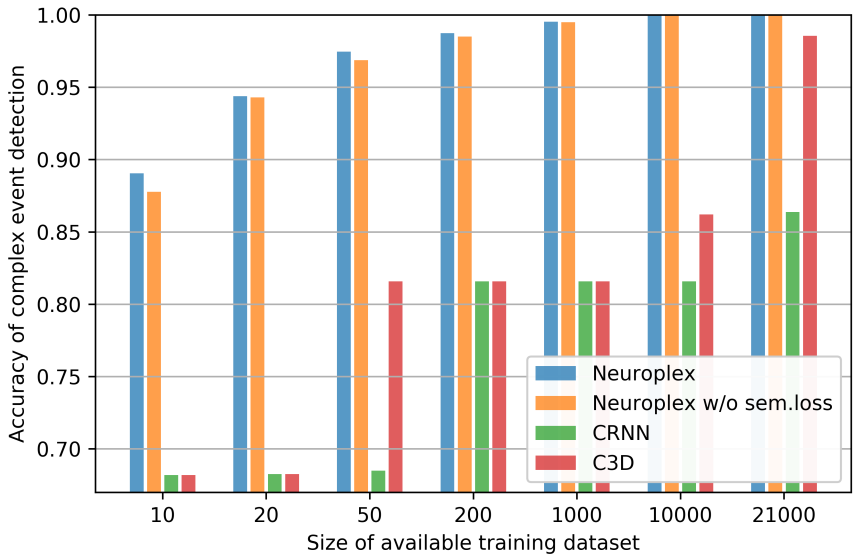

Figure 3.8: The performance changes with size of training data

Performance on different tasks. To further test whether NEUROPLEX scales well for complex event detection, we conduct a set of different experiments with datasets of various complexity. As shown in Table 3.2, simulations 1, 2, and 3 are tasks with increasing complexity, and simulations 4 and 5 are two simple tasks that all the testing models are able to learn.

From our formulation, the complexity of complex event detection would increase when the window length increases. A longer time window with more primitive events implies that the input space increases exponentially. The number of unique events and the length of a complex event, on the other hand, control the complexity of the complex event itself.

In simulation 3, the task is detecting 7 complex events composed of 3 or 4 primitive events in a window of 30 primitive events. Even though the CRNN shows poor performance in this setting, NEUROPLEX could still train the perception model well and achieve high accuracy for the complex event detection task. The fifth row of learning performance in Table 3.2 shows both the converted validation accuracy of NEUROPLEX on the complex detection task as well as the testing accuracy of the perception module. The performance of the C3D model in simulation 3 is better than simulation 2, even though the complexity of the task increases. One possible reason is that in simulation 3, the number of complex events we detect grows to 7, meaning that more feedback information is provided for a single event sequence sample since the annotation increases from 4 to 7.

In the simple task simulation 5, we notice that the CRNN model's performance is unnoticeably better than the proposed NEUROPLEX. This is because both the CRNN baseline and NEUROPLEX use the same network structure, and NEUROPLEX keeps part of its parameters frozen so that NEUROPLEX could be less flexible when finding the optimal solution. Besides, in NEUROPLEX, we decompose the entire learning space into the perception space and the reasoning space and learn them separately. This leads to a simplified problem and a improved learning speed, but the summation of optimalities in two sub-spaces is not necessarily the optimality of the entire space. However, this does not affect the efficacy of the proposed method since NEUROPLEX can get near-optimal performance with great compute,

	Simulation 1	Simulation 2	Simulation 3	Simulation 4	Simulation 5
Window Length	10	20	30	3	2
Number of Unique Events	10	10	10	3	3
Number of CE	4	4	7	5	4
Avg. CE Length	2.8	2.8	3.43	2	2
NEUROPLEX/ Perception	**99.39%** / 98.87%	**99.56%** / 99.17%	**98.65%** / 98.91%	**100.00%** / 99.84%	99.98% / 99.78%
CRNN model	69.98%	7.79%	1.83%	86.37%	**99.99%**
C3D model	88.47%	83.73%	86.91%	98.56%	99.72%

Table 3.2: Summary of CE datasets and training performance of different methods. Simulation 1, 2 and 3 are complex event tasks with normal complexity. Simulation 4 and 5 are the simple complex event scenarios that all the systems can train on.

data efficiency, and speed.

3.5.2 Synthetic Complex Audio Events

In this experiment, we show how NEUROPLEX can be applied to real audio event data. We construct a complex audio event dataset by sampling a subset from the DCASE 2018 challenge task 5 [19] and synthesizing complex events. The audio event data has 9 different classes: absence, cooking, dish-washing, eating, other, social activity, vacuum cleaning, watching TV and working. Each data sample is an audio recording of 10 seconds.

When creating the complex audio event data, we define the rules of complex audio events and build our logical machine based on it to provide annotations. The pattern is defined arbitrarily, as specified in Table 3.3. Some of them are defined based on regular human activities. For example, the complex event No.1 (CE 1) defines a regular dinner activity with a pattern of ["cooking" $\Rightarrow$ "eating" $\Rightarrow$ "dish-washing"].

For generating raw complex audio event waveforms, we first create an empty long audio data sample with a length of 100 seconds and then overlay random audio samples selected from the audio dataset. Since the labels of the audio samples are known, we use the logical machine to get the ground-truth complex event annotation for each generated long audio waveform. Table 3.3 gives a summary of the complex audio event dataset we generate.

With a given large audio file, the system is expected to extract the audio features from raw waveforms and make predictions about the occurrence of complex audio events inside this long period of time. In this experiment, we used the CNN model from DCASE 2018 [20] as our perception module. This CNN model contains two convolution blocks followed by two fully-connected layers with Relu activation. It has 17.8K parameters in total. Batch normalization and dropout are used to add robustness to the model.

The input to the model is the log mel-band energies for audio in a 10-second window. Therefore, to analyze the 100-second-long audio waveform, our system first used a sliding window to extract the mel-band features for every 10 seconds in a non-overlapping manner.

	Event types	Length	Num
CE 1	cooking ⇒ eating ⇒ dishwashing	3	1213
CE 2	social_activity ⇒ cooking ⇒ eating	3	1198
CE 3	working ⇒ other	2	2898
CE 4	watching_tv ⇒ vacuum_cleaner	2	2904
CE 5	absence ⇒ eating	2	2844
CE 6	dishwashing ⇒ cooking	2	2888
CE 7	absence ⇒ social_activity	2	2919
Event types: 9 . Avg length: 2.29. Dataset size: 16162			

Table 3.3: Summary of Complex Audio Event Dataset

The features are extracted from the raw data and then processed by the perception module to generate primitive events. The NRLogic model is also an LSTM network with one hidden layer analyzing the logic between audio events.

Similar to the MNIST sequence simulation, our first baseline model is also a CRNN network using the exact same structure as NEUROPLEX but without semantic knowledge injected. The second baseline model has a similar structure as the baseline CNN, but with twice the dimensions and a total of 55.8K parameters. We use this model to test if the deep learning model that works well on short time-series data would scale well on a much longer time series. These strong baseline models represent the performance of modern deep learning approaches well.

We train all the models for 200 epochs using the Adam optimizer with a learning rate equal to 0.001 and a batch size of 256. In Figure 3.9 we can see the learning curves of different methods in terms of validation MAE. Both the CRNN and NEUROPLEX models can capture the complex audio event, but the baseline CNN model does not converge to optimal. The CNN baseline model fails to learn in this task with a longer time series, which shows the limitation of mainstream deep learning models on complex event tasks.

In Table 3.4, we can see that the NEUROPLEX method could learn complex audio events

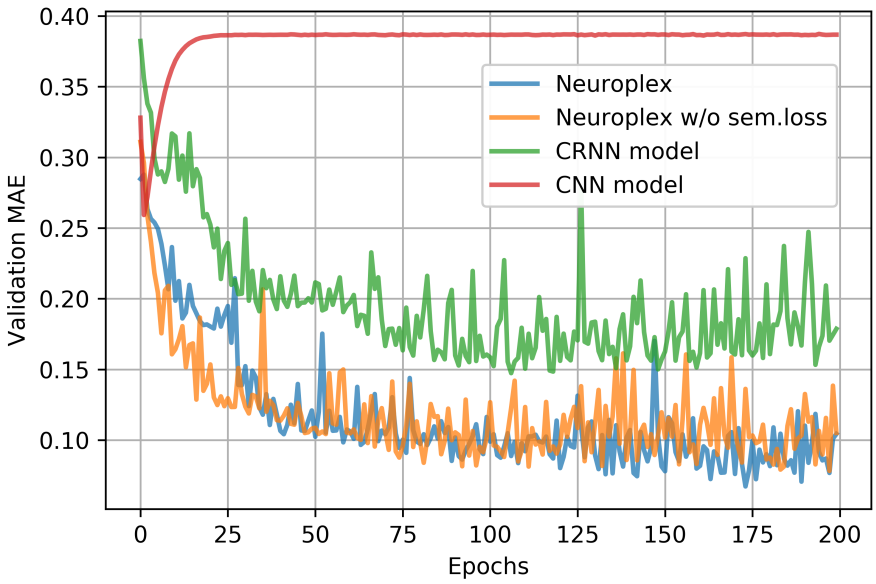

Figure 3.9: Learning Curves on Complex Audio Events

	NEUROPLEX	NEUROPLEX (w/o)	CRNN	ConvNet
Perception Acc	**89.44%**	84.93%	0.22%	NA
Validation MAE	**0.1015**	0.1032	0.1671	0.3884
Converted Acc	**92.53%**	92.39%	89.50%	85.09%

Table 3.4: Model performance on complex audio event data

with the lowest validation MAE and the highest accuracy of 92.53%. Additionally, the perception module performs well (with an accuracy of 89.44%) on the testing audio dataset—given that the same network trained using fully-annotated audio event data can only get an accuracy of 91.14%. The ablation study shows that NEUROPLEX without semantic loss also shows good performance but a little bit inferior to NEUROPLEX, which further proves that the semantic loss helps training.

3.5.3 Complex Nursing Events Detection

The third experiment is conducted on a complex nursing event dataset based on a public dataset from Nursing Activity Recognition Challenge[63]. The dataset contains nurse activity data collected from three sources: Motion Capture, Meditag, and Accelerometer sensors, and

it includes six different activities performed by eight subjects (nurses). These are C1: Vital signs measurements, C2: Blood collection, C3: Blood glucose measurement, C4: Indwelling drip retention and connection, C5: Oral care, and C6: Diaper exchange and cleaning of area. Each of these activities is performed 5 times by each nurse. The data is divided into 1-minute segments.

Because of the noisy and missing data problem in Motion capture and Meditag data, we only use the accelerometer data in our experiment. The data segments with less than 50 seconds are removed from the dataset. To extract features from the variable-length, un-uniformly sampled accelerometer data, we first divide the segment into 30 non-overlapping windows where the duration of each window is 2 seconds. For the data segments with a length smaller than 60 seconds, we perform imputation by filling the missing features with feature values of the last time window. We split the original dataset into a training set and a validation set. The data in the training set is used to generate complex nursing events data, and the validation set is used for evaluating the performance of the perception module.

To construct complex nursing events, we follow a similar approach as previous experiments. In the first simulation(Sim 1 in table 3.7), we randomly sample 10 data segments from the Nursing Activity dataset and concatenate them together representing a nursing activity sequence that takes over 10 minutes. The logic of our complex nursing events can be categorized into two groups as shown in table 3.5: complex nursing events, and violations of sanitary protocol. The constructed complex nursing events dataset has 2319 samples in total, each of which is an accelerometer sequence of ten minutes.

In this experiment, we use a convolutional LSTM structure [119] that is usually used to analyze IMU data. The model contains two 2-D convolution blocks with one LSTM layer and one fully-connected layer. A 0.5 dropout is applied to every layer. As the high-level reasoning logic have the same level of complexity, we keep using the one-layer LSTM network as the NRLogic model. We compare NEUROPLEX with three different baseline models:

- **ConvLSTM:** Like the previous experiments, this baseline has exactly the same network architecture as NEUROPLEX, but without human knowledge injected.

	Complex Nursing Event Name	**Complex Nursing Event logic**
Complex Event	Physiological Measurement	Vital sign ⇒ blood glucose measure ⇒ blood collection
	Indwelling Drip	Vital sign ⇒ Indwelling drip
	Patient Cleaning	Oral care ⇒ Diaper exchange
Protocol Violation	Unsanitary Operation No.1	Diaper exchange ⇒ blood collection
	Unsanitary Operation No.2	Area cleaning ⇒ blood glucose measure
	Unsanitary Operation No.3	Diaper exchange ⇒ indwelling drip

Table 3.5: Logic of Complex Nursing Events

	NEUROPLEX	ConvLSTM	ConvLSTM-2	LSTM-Attention
Perception Acc	**77.59%**	1.72%	NA	NA
Validation MAE	**0.0027**	0.1430	0.1860	0.6245
R-Square	**1.000**	0.882	0.807	0.002
Converted Acc	**100%**	93.67%	89.28%	78.81%

Table 3.6: Model performance on complex nursing event data

- **ConvLSTM-2:** This model has a similar structure to ConvLSTM. Instead of adding another LSTM network after the Conv-LSTM network, it learns to output the label of the complex event directly.

- **LSTM-Attention:** This model is inspired by [39], which demonstrates good performance in the Nursing Activity Challenge. Instead of using the GRU layer, this model uses two layers of LSTM, and the sequence of hidden states are aggregated using the attention mechanism to get a score vector. Two score vectors from two LSTM layers are concatenated, and a fully-connected layer is added to get the softmax result.

All the models above are trained for 400 epochs with Adam optimizer and 0.001 learning rate. The batch size is set to 256. In addition to the converted accuracy, we use the metric R-square to evaluate the performance of the regression task. R-Square measures how well the model fits the dependent variables. The value is usually between 0 to 1, and a bigger value indicates a better fit between the predicted and actual value. R-Square is calculated as follows:

$$R^2 = 1 - (\sum_i (y_i - f_i)^2)/(\sum_i (y_i - \overline{y})^2)$$

Table 3.6 shows that the proposed NEUROPLEX performs well on complex nursing event detection with raw accelerometer data, and the perception model also gets near-native accuracy. (80% accuracy when perception network trained directly on Nursing activity dataset). The other deep learning baselines fail to show comparable performance, and the NEUROPLEX shows the best performance in all different metrics.

Detection over long period of time. To further test whether NEUROPLEX scales well on complex nursing event detection, we conduct a set of experiments with increasing length of the time window, and keep the definition of the complex nursing event the same as that in the simulation 1. We test the NEUROPLEX with the other three baseline models, and measure their performance based on R-square and converted accuracy. As shown in table3.7, as the length of time window increases, the complex event detection becomes harder, and the model performance degrades. We notice that when the time window length is less than 30 minutes, NEUROPLEX can successfully learn to detect complex event at high accuracy. As the time window length grows, it takes more time for the model to converge, and 400 epochs are not sufficient to get the optimum, although it already performs pretty well and beats other baselines by a large margin. In simulation 6, we can see that the NEUROPLEX model can still get about 80% detection accuracy in a one-hour-long time window, proving that NEUROPLEX is robust to long-term reasoning.

Methods	Sim 1	Sim 2	Sim 3	Sim 4	Sim 5	Sim 6
Time window (minutes)	10	20	30	40	50	60
R-square						
Neuroplex	**1.00**	**0.99**	**1.00**	**0.90**	**0.88**	**0.85**
ConvLSTM	0.88	0.90	0.66	0.32	0.33	0.35
ConvLSTM-2	0.81	0.76	0.80	0.76	0.75	0.70
AttentionNet	0.02	0	0	0	-0.01	-0.02
Converted Accuracy						
Neuroplex	**100%**	**98.90%**	**100%**	**83.59%**	**79.00%**	**79.63%**
ConvLSTM	93.67%	83.29%	67.75%	40.79%	39.03%	37.47%
ConvLSTM-2	89.28%	80.08%	75.70%	60.30%	45.83%	39.48%
AttentionNet	78.81%	2.60%	0.62%	0.50%	0.11%	0.02%

Table 3.7: Experiment result of complex nursing activity detection as the length of time window increases.

The other neural-network-based models all suffer from the long temporal range. As we can see, in simulation 1, the ConvLSTM models is able to detect complex events with acceptable performance. However, as the length of time window increases, their performance drops quickly. The ConvLSTM-2 model performs better than ConvLSTM, suggesting that the added LSTM network in ConvLSTM does not help capture temporal dependencies without human knowledge injected. Although it demonstrates good performance on other tasks, the AttentionNet can not learn complex event on accelerometer data. It basically gets an approximate zero R-square value in all the simulations (the R-square is negative in some cases because the model performs worse than the null hypothesis). The converted accuracy in sim 1 is 78.81% because it outputs small fraction numbers which are rounded to zeros.

3.6 Discussion

The design of NEUROPLEX framework is inspired by the human learning process: the perception ability is trained in a data-driven approach, and the mid- to high-level reasoning ability can be taught in an efficient manner—the knowledge is passed in a condensed form of logic rules. Human can know what complex event is even without seeing any examples before. On the other hand, human store knowledge in neurons inside the brain, so high-level logic should be expressed as a neural network as well.

In NEUROPLEX, we create a dual form of the reasoning module so that the system learns by back-propagation in a standard supervised manner. During the inference stage, even though the NRLogic is also available for the forward propagation, we use the logical machine instead. This is because the logical machine is more reliable and explainable than the NRLogic model. Additionally, the logical rules are often more compact than deep neural networks, so they can usually be executed quickly over sensor networks with minimal computation overhead. We enumerate the limitations of the current approach and future research directions as follows:

Distribution of complex sensor reasoning. The hybrid architecture of NEUROPLEX

is designed to thrive in emerging distributed computation architectures that push sensor inferencing towards edge devices. However, primitive events that stem from edge device inferences are currently fused at a single CEP node. Future work can focus on distributing the reasoning module across heterogeneous sensor networks with dynamic computation placement.

Prior knowledge of complex event reasoning logic. We have assumed that the higher-level logic of a complex event is provided by the user. This is a reasonable assumption, and it is broadly used not only in neural-symbolic systems [73, 120, 109], but also in earlier works on rule-based activity recognition [85, 105], activity decomposition using sensory grammar[75], and complex event processing [33, 31, 90]. During training, the use of prior knowledge reduces the burden of data annotation and accelerates the learning process significantly. Additionally, we use the CEP language with a BNF grammar like DeepCEP[120] to formally define the logic rules, which standardizes and simplifies the coding of human knowledge.

However, the system may be deployed in a new or evolving sensor network environment– where the definition of complex events provided by the user may not present robust detection. Future efforts can focus on learning or fine-tuning the reasoning module by freezing the trained perception module and updating the NRLogic module. After learning, the FSM can be extracted from the trained RNN network to provide human-understandable logic.

Complexity of logical reasoning. In the context of complex event detection, the NRLogic model is trained to capture the logic of a complex event, which could be arbitrarily complex. The NEUROPLEX framework can be generalized to a wide range of scenarios, and NRLogic is able to capture not only temporal logic, but also spatial logic over sensory networks. For all experiments in this paper, the complex event logic can be captured effectively by an LSTM network with one hidden layer. However, as the complexity of logic increases, deeper and more sophisticated networks need to be used to approximate the logical function. Future research will investigate what the most efficient structure is to capture the reasoning logic of different complexity in different settings.

Annotation of complex event sensor data. In the problem of complex event detection,

we only care about the occurrence of complex events at the current time. Thus, the user would only make annotations when complex events are actually happening and do not need to care about simple events. This greatly reduces the labeling burden for numerous applications. In this work, we formulate the complex event detection as a regression problem, which requires annotations of complex events happening times. This is sometimes not a trivial task because sensors with events happening over long periods could generate multiple primitive events of the same type, leading to an increase in complex event instances. Also, in real-world scenarios, the complex event labels may not have accurate spatial-temporal metadata, e.g., inaccurate timestamps. Future research will investigate how robust the proposed NEUROPLEX is when the annotation is noisy, and how to aggregate and learn from events of the same type.

Generalization to real multimodal scenarios. Although NEUROPLEX is evaluated on complex events from a single modality in this paper, we believe that it can be extended to multimodal complex event learning tasks as well. We envision a framework similar to [87, 82], with multimodal data streams input as different channels. While the performance of NEUROPLEX in multimodal scenarios is not presented in this paper as the on-going pandemic conditions prevent us from conducting the requisite experimental study, we plan to address it in future work.

3.7 Related Work

A body of earlier work studied complex events in sensor networks [75, 133, 132]; however, we focus on detecting complex events over unstructured data using deep learning models. We first provide an overview of the state-of-the-art deep learning methods related to complex activity detection. After highlighting the scalability and data efficiency issues of these approaches, we describe how prior works have attempted to integrate human logic directly into the deep learning models. Although these approaches help to regularize and bootstrap the associated learning processes, they fail to address the notions of scalability in the spatial-temporal domains. We then highlight a recent class of neuro-symbolic approaches

that combine deep learning with explicit symbolic reasoning, and discuss modern approaches to complex event detection using hybrid systems.

Deep learning for complex activities.

Prior work on deep learning methods has explored learning and analyzing time-series data such as human activities. In particular, several works in video classification have proposed solutions for detecting *complex activities* over short periods [117, 54]. Images [121] and audio [28] are also considered to contain complex events, and several studies [116, 51] use multimodal information to perform classification. Similarly, anomalous event detection [122] utilizes motion features to extract temporal-spatial localization features for complex event detection.

Although the aforementioned works have shown promising results in their respective domains, they do not have a clear definition of *complex events*. Generally, they use the term to describe events that contain interactions between different elements. Furthermore, these works typically only consider processing information from a single input instead of a distributed set of heterogeneous sensors. Although multimodal data fusion has been explored, they fail to fuse the information at a semantic level so as to provide a clear explanation of the result. Additionally, these learning-based models alone cannot learn extremely long temporal dependencies well, even with the help of the LSTM [44] structure and the Attention mechanism[9]. They typically reason about events on the order of seconds. Finally, in order to have effective models that generalize well, learning-based methods necessitate the consumption of large amounts of data with an expensive annotation process [29].

Intuitively, integration of human logic would address these issues in an interpretable way. We next review how prior works have integrated logic reasoning with machine learning (ML).

Combining logic reasoning with ML. Combining reasoning with learning is a popular topic in the AI field, and one interesting direction is integrating symbolic human knowledge with ML models. One approach to integration is to instrument logical formulae into an embedding space while preserving the logical meaning and the relationship between formulae. ConvNet Encoder [60] and TreeLSTM [104, 67, 131] embed formulae using different network

structures. LENSR [118] first converts logical formulae into d-DNNF DAGs and use a Graph Convolutional Network to perform an embedding. Another approach uses knowledge to impose additional logic loss to help augment the original training objective [123, 102, 21, 91]. In [46], a distillation method is used to transfer knowledge from a rule-regularized teacher network to a standard student network. [123] imposes a semantic loss on predicted probability by quantifying the probability of generating a satisfying assignment by randomly sampling from the predictive distribution. Other methods include Logistic circuits [25] and Logic Tensor Network [70] design specific structural to incorporate logic rules. [76] attempts to enforce machine learning models to follow STL logic rules, by training the networks with both ground truth labels and logically-corrected labels.

However, the preceding methods are not designed for reasoning about spatial and temporal events at scale. Therefore, we next discuss neural-symbolic frameworks that allow a hierarchy of reasoning between deep learning and human logic.

Hybrid Neural-symbolic frameworks for complex tasks. Building hybrid systems that utilize the power of both human logic and deep learning is becoming a hot trend. Caesar [73] proposes a system that uses both deep learning models and rule-defined complex activity graphs to recognize complex activities in a multi-camera video surveillance setting. [109] shows that it is possible to fuse proxy deep learning models and use ProbLog [17] defined rules to perform crime detection. In [120], simple events are first captured by deep data abstractors and then reasoned about by a complex event processing (CEP) engine that takes human definitions as logic rules. All of these works use either existing or newly defined languages to inject human knowledge into the system to perform reasoning to detect complex events. They utilize pre-trained neural network models and only focus on the inference path of the problem as learning is beyond their scope.

The ability to *learn* is of great importance, especially when the system is deployed in a new environment. SATNet [112] sits at the boundary between end-to-end deep learning models and neural-symbolic hybrid approaches. It introduced a differentiable SAT solver that can be integrated into deep learning models as a MAXSAT layer. This logical structure

can be learned using a supervised end-to-end approach. However, the SATNet does not have symbolic representation, nor is the MAXSAT layer explainable.

DeepProbLog [77] provides a generalized probabilistic logic programming language that incorporates deep learning into ProbLog. The parameters of both neural networks and logical rules are learned in an end-to-end manner using αProbLog while supporting symbolic representation. An important feature for DeepProbLog is that it supports symbolic representation inside the system. Therefore, unlike black-box deep learning models, the results of DeepProbLog are explainable. While DeepProbLog is designed to handle complex problems where the logic can be expressed as combinational logic, it struggles to represent the sequential logic in ProbLog as the number of nodes grows exponentially.

Unlike prior works, in our approach, logical knowledge is not used to augment training objectives, but rather perform individual reasoning tasks. Trained to mimic the logic rules, the NRLogic is plugged into the system as a layer to perform learning tasks.

3.8 Conclusion

In this work, we presented NEUROPLEX, a neural-symbolic framework for detecting complex events. Using semantic knowledge to guide the learning, NEUROPLEX can learn to detect complex events with much fewer and sparse annotations. Results on different datasets proved the effectiveness, reliability, and robustness of NEUROPLEX. Future work will focus on the deployment of NEUROPLEX in real-world scenarios with multimodal sensory data.

CHAPTER 4

DeepSQA: Understanding Sensor Data via Question Answering

4.1 Introduction

Sensors in various embedded, wearable, and mobile IoT devices produce enormous amounts of data, which algorithms help transform into actionable insights and predictions that guide decisions and interventions at various scales. While recent years have seen the emergence of powerful deep-learning-based neural network models, capable of making rich and complex inferences from large amounts of sensory data, current data-to-decision pipelines are highly constrained as they employ specialized models for specific tasks such as detecting a set of events and activities.

Imagine instead a future where a human decision-maker is not limited to a fixed set of inferences computed from sensory data, and could instead ask flexible natural language questions about events and activities present in the data and get answers. For example, instead of a processing pipeline extracting fixed information about various activities of daily living from a user's wearable and ambient sensors, imagine the user being able to ask "How long did I exercise between lunch and dinner yesterday?" and "How many times did I drink water yesterday?" Or, imagine that instead of being presented with a fixed set of traffic events and statistics derived from time-series data from traffic sensors, a city manager could ask questions such as "How long did the congestion on Highway 99 last?" and "Was there an accident during the hour preceding the congestion?"

Our research is inspired by a vision of providing users with the ability to extract a

variety of inferences from sensory data, by asking flexible natural language questions instead of being limited to the rigid outputs of a fixed set of classification and regression models. To achieve such a capability, one needs a framework that can formulate answers to novel and arbitrary questions about an underlying sensor dataset without requiring a new model to be trained for each question, and also be able to incorporate new knowledge efficiently. Recent advances in deep-learning-based natural language processing and its use for tasks such as asking questions about images[4], texts[88], and databases[42] suggest that restricted forms of such a capability are certainly possible for spatiotemporal sensor data as well. This paper presents the results of our exploration of this problem.

In this work, we propose the DeepSQA, a generalized framework to address the Sensor Question Answering (SQA) problem of answering natural language questions about raw sensory data in distributed and heterogeneous IoT networks. Figure 4.1 illustrates some exemplar questions that DeepSQA can solve. Suppose a user interfaces with a set of IoT devices, e.g., a smartphone and a smart band to keep track of their daily activities. With multimodal data collected by wearable devices, they may want to query their behaviors from different perspectives. For example, the user could ask "How many times did I send an MSM while driving?" or "How long did I exercise between lunch and dinner?" Based on the collected sensory data, the SQA system should answer each of these questions accordingly with the correct natural language answer, e.g., "three times" or "1.5 hours".

In addition to DeepSQA, we also introduce SQA-Gen, a software framework for generating SQA datasets from underlying labeled sensory data. To evaluate DeepSQA, OppQA dataset is created with SQA-Gen for complex human activity question answering. This dataset focuses on spatial-temporal relationships across different sensors, and it contains over 1K sensory contexts and 91K questions. A detailed analysis is performed on this dataset with various modern baseline models, providing insights into the SQA task. We provide open-source implementations of the DeepSQA framework, SQA-Gen data generation tool, and the access to the OppQA dataset[1]. We believe this will benefit the community of

[1]The dataset and codes for DeepSQA are available at https://github.com/NESL/DeepSQA.

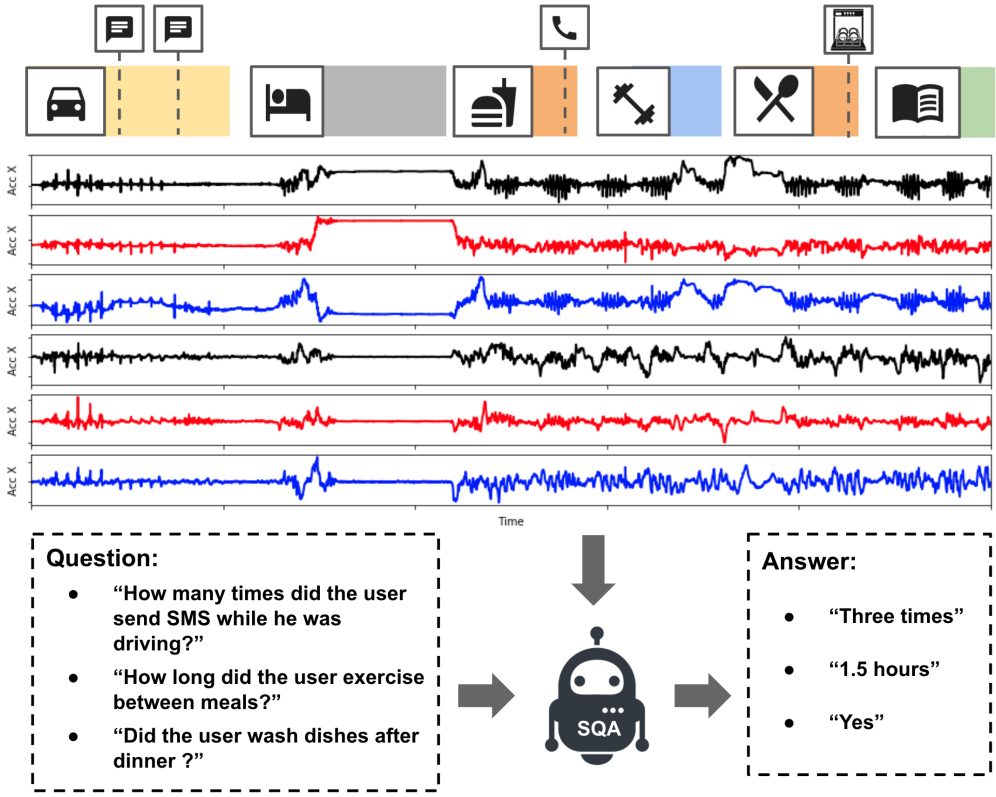

Figure 4.1: Examples of open-ended natural language questions supported by our proposed Sensor Question Answering model.

sensory question answering research.

Contributions. Our contributions are enumerated as follows.

- Firstly, we introduce the DeepSQA framework, the generalized QA framework to address the SQA problem by enabling natural language questions about raw sensory data in distributed and heterogeneous IoT networks.
- Secondly, we propose SQA-Gen, a software framework to generate SQA data using labeled source sensory datasets. Based on SQA-Gen, we create the first SQA dataset on complex human activity question answering, to benchmark SQA models' performance on natural language QA about spatial and temporal properties of raw sensory data streams.
- Thirdly, we evaluate DeepSQA across several state-of-the-art QA models on OppQA,

and enumerate the challenges at the frontier of SQA.

- Lastly, We provide an open-source implementation of DeepSQA, SQA-Gen, as well as the access to the OppQA dataset.

4.2 Related work

In this section, we review the related works on sensory data processing and question answering in different domains.

Machine Learning for Sensory Data. Sensory data processing is not only a critical problem in the signal processing field, but also a hot topic for machine learning applications. Vibrant research has been performed in the field like visual and acoustic domains. However, in this research, we are interested in sensory data that humans cannot easily understand, i.e., sensory data from devices like inertial sensors that are nothing more than a series of scalar physical phenomena. Due to the lack of senses or a standard vocabulary, humans would have a hard time associating these time-series values with the high-level symbolic concepts.

With the advancement of machine learning techniques and compute power, it is possible to make inferences on raw sensory data using a data-driven approach. [66] presents a survey on using wearable sensory data to perform Human Activity Recognition(HAR). Recent research [38, 98, 111] shows that deep learning techniques can better infer human activities. [113] uses the WiFi signal to achieve device-free activity recognition. Besides, time-series sensory data like EEG [10, 114] or GPS [130, 14], are used to make high-level inferences about different activities and events.

However, all the existing works focus on using sensory data to perform fixed predefined tasks, e.g., classifying subject behavior or predicting user sentiment. If the tasks are changed, the data need to be re-labeled, and the models have to be retrained. Despite the inefficiency of data labeling and the cost of model training, there's still no framework that can provide solutions for arbitrary tasks within a specific range.

Question Answering. On the other hand, research in the visual and natural language

processing domains propose a framework that allows task-aware inferencing, in the form of question-answering (QA). In QA, a model is trained to take both a question and its context as input, and answer the given question based on the context data. This QA task requires models to be capable of not only processing raw context data, but understanding natural language questions as well. The questions asked to the model determine the tasks that the model performs. Different tasks, based on either the same or different context data, can be handled by a single QA model. There has been a body of work on question answering applications in different domains, which can be broadly categorized into two groups based on context data.

The first group of QA tasks deals with static data, such as texts, images, charts, and structured data like tables and knowledge bases. As the main focus of this paper is on the question answering with multimodal data, we do not discuss the related work of natural language QA [88] here.

In the Visual Question Answering(VQA) domain, large datasets [4, 55, 49] have be proposed to give fair benchmarks. [55] creates a large VQA reasoning dataset using generated images and questions, which helps reduce data biases and test model generalization ability. In [49], questions are synthesized on real-images to extend the domains further. Different methods are proposed to handle the VQA problem. A major trend is to fuse the image and question features in different ways, such as combining CNN features of images and LSTM features of questions together[4]. Models based on attention mechanism [126, 59] use concatenated feature to first calculate attention weights of the image, and then predict the answer based on the attended area. FiLM[84] proposes a model that interleaves standard CNN layers with linear layers, tilting the layers' activations to reflect the specifics of given questions. More recent work shows that recurrent approaches like [48] with multiple-step reasoning can achieve good performance on datasets where complex reasoning is required. With additional supervision, neural-symbolic methods show better performance on the VQA task. [56, 45] use modular approaches to synthesize models that have the same structure as the questions. [128] uses deep learning model to parse both the image scenes and questions,

and then answers different queries in the semantic space. Instead of using latent feature representations only, a body of work[3] solves VQA problems using additional object-level features and achieves excellent performance. However, this method could not be adapted to the SQA problem, since obtaining the semantic feature is hard for the opaque sensory data.

There are also emerging domains that ask questions on the context of other modalities, like graphs and tables. [57, 58] propose the task of question-answering on graphs and diagrams, where the answer categories are context-dependent, and the answers to questions are sensitive to small variations in the diagrams. [42] uses a BERT[24]-like structure to answer questions based on tabular data.

In summary, all of the work discussed above focus on answering questions with static data, where complex temporal reasoning is not required to get the correct answer.

The second group of QA tasks relies on time series data. [50, 69, 106, 53] solve the problem of video QA with video clips as context data. [2] introduces the task of audio question answering, and uses vision-based models to process the spectrogram features extracted from audio contexts. Although spatio-temporal reasoning is performed in these models, they can only handle context data collected from a single source. Question answering with distributed sensory data from multiple sources with heterogeneous modalities requires the ability to perform sensory fusion and inter-sensor spatial reasoning. Also, it is unclear whether these approaches discussed above are still effective on sensory data that are opaque to humans, such as the IMU data that humans cannot understand.

Semantic Parsing. An alternative approach to handle flexible natural language queries is semantic parsing[11]. Semantic parsing is the process of mapping a natural-language sentence into a logical form, which is a machine-understandable, formal representation of its meaning. After this process, the questions in logic form can be used to query structured or semi-structured knowledge bases. In our SQA problem, to enable a semantic-parsing-based system, effort must be first taken to train a model that processes and maps raw sensory data into a logical semantic space interfacing with logic queries. However, it is impossible to define such an informative semantic space for opaque and unstructured sensory data,

especially when tasks are not provided yet. Also, preparing the annotated data for training the sensory model is a huge burden. Therefore, in this paper, we choose to use an end-to-end approach, which implicitly parses the natural-language questions using neural networks, to tackle the SQA problem. Related work in the semantic parsing field is not discussed here since it is beyond our scope.

4.3 Formalizing SQA problem and Design of DeepSQA

In this section, we formally define the problem of Sensory Question Answering (SQA), and describe the design requirement of SQA systems. Finally, we propose the generalized DeepSQA framework.

4.3.1 Sensory Question Answering (SQA)

Unlike visual, acoustic, or textual data, sensory data collected from sensors, e.g., IMUs and barometers, naturally cannot be understood by humans. All humans are "sensory impaired" to the sensory data due to the non-capability of proper sensory abstraction and the deficiency of a standard vocabulary describing different phenomena and their characteristics. Sensing using state-of-the-art deep learning models proves to be an effective way to comprehend the opaque sensory data. However, outputs of deep learning models are restricted to a pre-defined set of labels, limiting the inferencing flexibility significantly. Also, in order to make various inferences, users need to train different models, which requires large amounts of labeled data and compute power. Based on these limitations, it is essential to have an intelligent system to help humans understand sensory data in the form of question answering. We envision a system that gathers all sensory data from distributed heterogeneous sources. When the user asks arbitrary natural language questions, the system processes the received data, reasons about the spatial-temporal relationships between events, and provides correct answers in the form of natural language to humans.

Here we formally define the Sensory Question Answering (SQA) problem as follows.

Consider a sensor network where data d_i of different modalities are continuously collected by a set of n distributed sensors. At some time point t, the user asks a question q in natural language format to the system, which is expected to output the correct answer a based on the context data D. $D = \{d_1, d_2, ..., d_n\}$, where $d_i = \{d_i^{t-k+1}, ..., d_i^{t-1}, d_i^t\}$ is a sequence of data collected by the i-th sensor. The number k represents the maximum length of history considered when answering the question, and it is usually determined based on different applications and system memory limitation. So this SQA system learns to model the conditional probability of $p(a|q, D)$.

Based on the formulation above, the SQA system needs to satisfy a set of requirements. Firstly, it should have the ability to process and fuse raw, multimodal sensory data collected directly from heterogeneous sensors. Secondly, the SQA system should be capable of analyzing natural language questions to understand what the various tasks are during the inference time. Thirdly, complex temporal and spatial dependencies between different sensory modalities, and more importantly, the correlation between questions and sensory contexts, need to be explored and captured by the SQA system so as to answer the question correctly.

4.3.2 SQA Architecture Design

As shown in Figure 4.2, we propose a generalized framework called DeepSQA. Here, we adopt the idea of question answering in other domains like VQA to our SQA problem. Basically, questions and images are first processed by two different paths to get compact representations, and then an SQA module is applied to analyze these representations and predict answers.

Sensory and Question Representations The question q with M words in natural language form is first embedded into a sequence of M embedding vectors using an embedding matrix, which is either pre-trained on a large text corpus, or learned together with other parameters during the training time. The embedding sequence is then processed by a recurrent-based structure, such as a multi-layer LSTM or bidirectional-LSTM network, to get the question representation.

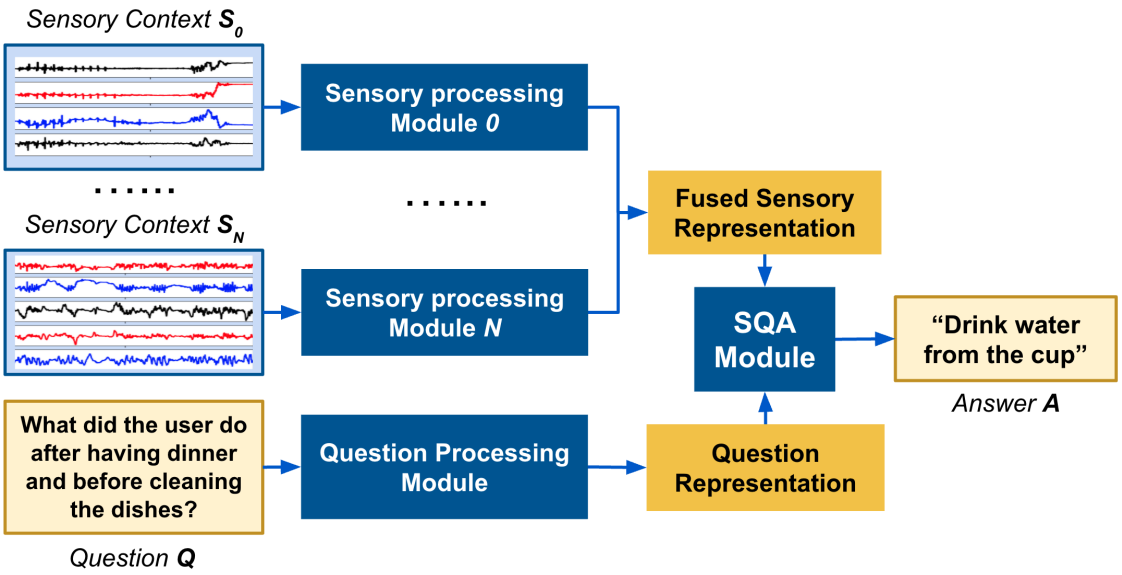

Figure 4.2: The Generalized DeepSQA Framework

In order to calculate the sensory representation, sensory processing modules are required to extract informative features from multimodal raw sensory data.

In the visual domain, a variety of pre-trained models are off-the-shelf. These models are trained on population-scale data of immense size, like the ImageNe[23] and Microsoft COCO[71] datasets. They are capable of processing and extracting informative visual features that can be used to perform a bunch of different downstream tasks, e.g., image classification, object detection, semantic segmentation, etc. As a result, in the task of visual Q & A, these pre-trained models can be used to obtain visual representations at both image-level and object-level directly.

However, for sensory question answering, such a general-purpose model pre-trained on population-scale data is not available, given the absence of a enough large-scale sensory dataset in both the sensing and machine learning communities. Because of the heterogeneity, data from different sensors have different modalities and sampling rates, and data collected in different locations, on different users, might vary significantly. As well-studied in prior work [15], a machine learning model trained on one sensory dataset would not perform well with other sensory data. This makes it impossible for us to use pre-trained models to get sensory representations.

Consequently, in DeepSQA, we use data-specific sensory processing modules instead. The sensory processing modules are designed based on the size and modality of raw sensor data. A popular structure is the Convolutional LSTM network (ConvLSTM), which can be used for extracting relevant sensory features and reasoning about their long-term temporal dependencies. An effective sensory processing module is crucial in the DeepSQA framework, as the downstream question answering task requires accurate context information to make correct predictions.

We also investigate using an auto-encoder-based approach to extract sensory features. The auto-encoders can be trained in a self-supervised manner with no annotation required. The intermediate bottleneck layer, which squeezes the original high-dimensional time-series sensory data into low-dimensional vectors, can be used as the sensory representations. However, because of the lack of training guidance, the sensory representations generated by the auto-encoders are not informative enough and can lead to huge performance sacrifice. Therefore, in this paper, we omit this method, and focus on the ConvLSTM based sensory processing module.

Reasoning with SQA Module In the DeepSQA, an SQA module is designed to find the inter-correlation between sensory and question data, perform reasoning, and predict final answers to complex compositional questions. Instead of explicitly decomposing the reasoning into multiple semantic sub-tasks as humans do, we adapt a neural-network-based approach to perform complex reasoning implicitly. Different techniques can be used in this SQA module, such as the simple convolutional RNN with multimodal data fusion, bi-linear pooling, and the attention mechanism. We will detail different models based on DeepSQA framework in Section 4.5.

Although there are some other techniques [3] that show superior performance than the purely neural approach in the VQA domain, they either require object-level or semantic features as input, or need additional supervision to decompose questions into semantic sub-tasks. These methods are not suitable in our case of SQA, simply because sensory data are not human-understandable, and the usage of semantic knowledge is not an option.

All the different modules in DeepSQA are stacked and connected together, and the entire system can be trained in an end-to-end fashion.

To train and evaluate different SQA systems along with a set of baselines, a large and diverse SQA dataset is required. However, because of the opacity of sensory data, humans cannot provide answers to question based on sensory contexts directly, and hence it is impossible to create an SQA dataset using the crowd-sourcing method as the VQA does. In this work, we propose a method SQA-Gen, which can generate SQA data based on a labeled source sensory dataset. Using this tool, we generate the OppQA dataset, that questions human activities and their temporal relationships. In the next section, we detail our tool for transforming a sensory dataset into an SQA dataset.

4.4 SQA-Gen: SQA Dataset Generation Tool

In this work, we introduce a tool SQA-Gen for creating SQA data from a source sensory dataset. Using SQA-Gen, we generate the OppQA, a human activity sensory question answering dataset, based on the OPPORTUNITY [93] data. In this section, we first describe the data generation method we use in SQA-Gen, and then show the statistics about our generated OppQA dataset. We also need to mention that this generalized tool SQA-Gen can also be used to create new SQA datasets with other labeled source sensor data. We make the OppQA dataset and SQA data generation tool SQA-Gen available online to facilitate future research in the SQA field.

The context of the SQA dataset needs to have the characteristics of regular sensory data, showing the necessity of being multimodal, and involving multiple users with multiple devices. Besides, the sensory context needs to be real time-series data collected by sensors. Synthesized context data are unrealistic [55, 2], and the SQA models trained on that would not have satisfactory performance when deployed in real scenarios. Because of sensory data's opacity to humans, it is difficult to ask a human generator to create questions and answers based on sensory context manually. Therefore, in this work, we are using an automatic

approach to generate sensory questions and answers.

Although the automatic generation may not provide many linguistic variations on natural language questions and answers, it has several other benefits. Firstly, it always provides objective and correct answers to even complex and compositional questions. In comparison, humans sometimes fail to give right answers [55], especially to complicated questions. Secondly, automatic generation is a scalable approach that can be applied to different source datasets at a minimal cost. In Section 4.6, we use SQA-Gen to generate multiple variants of OppQA with different parameter configurations, and also apply SQA-Gen to a new sensory dataset, ExtraSensory [108], to obtain new SQA data in a different domain. The scalability of SQA-Gen helps evaluate and train SQA models efficiently. It allows us to investigate SQA models' performance under different settings and also provides abundant amounts of data for model training. Thirdly, it is evident that the automatic QA generation shows higher efficiency with lower costs than human creation.

4.4.1 Source Data Selection

In this work, we choose the domain of human activity analysis to evaluate the SQA systems, and select the OPPORTUNITY Dataset[93] as our source sensory dataset. The OPPORTUNITY is a dataset used for benchmarking human activity recognition (HAR) tasks. The data are collected with wearable, object, and ambient sensors. Here we are only using the seven body-worn inertial measurement units (IMU) sensors to make inference on user activities. These sensors are located at different parts of the body: Left lower arm (LLA), Left upper arm (LUA), Right lower arm (RLA), Right Upper arm (RUA), Back of the torso(BACK), and Left/Right shoes(L-SHOE/R-SHOE). These inertial measurement units provide readings of: 3D acceleration, 3D rate of turn, 3D magnetic field, and orientation of the sensor with respect to a world coordinate system in quaternions. The IMU data are collected at a fixed sampling rate of 30Hz.

OPPORTUNITY also provides a rich set of annotations, including three different hierarchies: high-level activities, modes of locomotion, and low-level activities. Here we are using

the two hierarchy of labels: locomotion activities(sit, stand, walk, lie, other) and low-level activities, including 17 different micro activities such as opening and closing doors, shelves, drawers, drinking tea, etc. This enables us to ask questions that reason about the interactions between different levels of activities, for example, "Did the user drink tea while he is standing?". OPPORTUNITY collects data on four distinct subjects. For each subject, six separate runs were recorded. The total length of the data is about 8 hours.

4.4.2 SQA Data Generation

With the labeled source sensory data, we are able to generate an SQA dataset. Inside SQA-GEN, the sensory context, the question, and the answer are all accompanied with a semantic representation to enable automatic machine generation. The entire data generation pipeline is illustrated in Figure 4.3, which we describe in detail below.

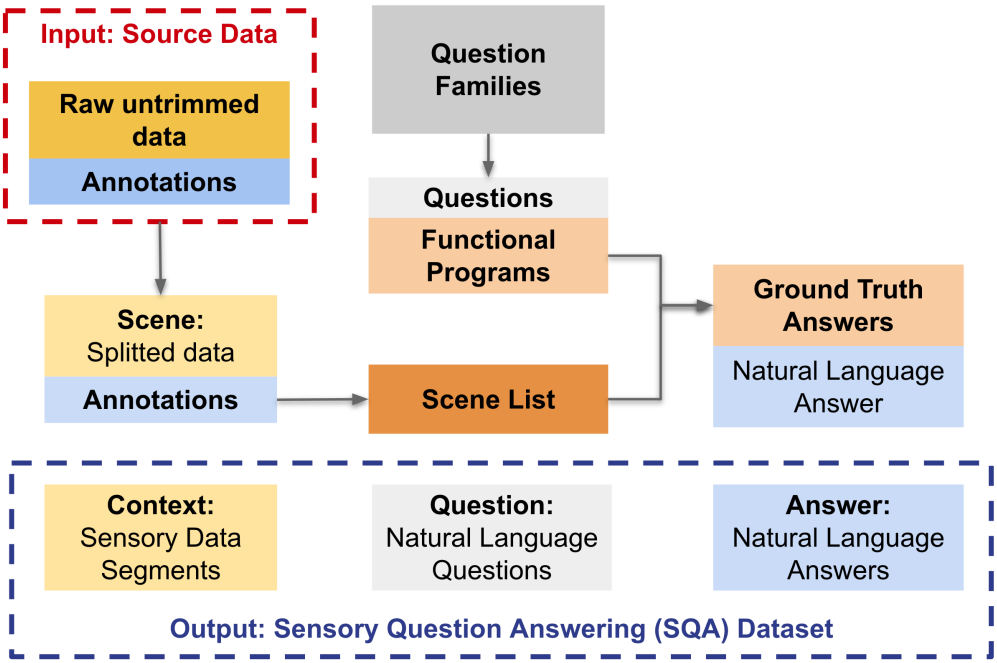

Figure 4.3: Data generation Pipeline

Sensory Context Generation. The untrimmed source OPPORTUNITY data are first split into data sequences with appropriate length, which are then used as the sensory scene (context) for the SQA task. The context length is an adjustable parameter of SQA-GEN,

which determines the maximum length of history considered when answering the question in real-time. In OppQA, we choose the context length to be one minute, which means that the past 1800 data samples are used to answer the question. The data splitting is performed using a sliding window approach on the original time-series data. The length of the sliding window is one minute (1800 timesteps), and we use a stride of 20 seconds (600 timesteps) to avoid having two consecutive sensory contexts with too much information in common.

Scene Representation. For each sensory scene, we have its associate label sequence obtained from OPPORTUNITY's two hierarchies of annotations. The label sequence provides information about what the user is doing at every timestep. Corresponding to the two-hierarchy annotations, we create two scene lists using the label sequences as the semantic representation for each sensory context. A scene list $\mathbf{A} = \{A_1, A_2, ..., A_n\}$ contains n activities happening in a sequential order. For each activity $A_i = \{y_i, d_i, s_i\}$, the activity type y_i, duration d_i and starting time s_i is stored.

To create the scene lists, we traverse the label sequences and aggregate the consecutive data samples with the same label to a single activity. The "other" activity is omitted here. With the semantic representation of the sensory scene, we can easily reason about the relationships between different activities in an automatic manner.

We formally define the temporal relationships between different activities. We say activity A_i is after/before A_j if $i > j$ or $i < j$, and A_i is right after/before A_j if $i = j+1$ or $i = j-1$. And for the "While" relationship: suppose A and B are two scene list of different hierarchies, then the user did A_i while doing B_j only when the starting and ending point of A_i is within the range of $[B_j.s, B_j.s + B_j.d]$.

Question Generation. Inspired by [55, 49], SQA questions are generated automatically using a template-based method. In the SQA-Gen tool, we create a set of 16 different question families, covering different question types: Action Query, Time Query, Existence, Counting, Action Comparison, Value Operation, etc. For each question family, a functional program template is used for constructing functional programs. With this program, several text templates exist for creating questions in different natural language forms. We need to

mention that new question families and text templates can be easily added to SQA-GEN to generate new questions based on diverse requirements.

A functional program is composed of a set of functional building blocks. Similar to [55], we have a function catalog that deals with different operations for SQA reasoning. These functions can be combined in different ways with different input parameters to create an infinite number of questions with arbitrary complexity.

For example, the question "What did the user do before opening the fridge and after closing the drawer?" is generated using the text template "What did the user do [Relation] [Activity] [Combinator] [Relation] [Activity]", while filling in "before", "after" as relations, "and" as the Combinator, "open the fridge", "close the drawer" as activities. The functional program "query_action_type(AND(relate(before, open the fridge), relate(after, close the drawer)))" is instantiated using program template "query_action_type(Combination(relate(Relation, Activity), relate(Relation, Activity)))".

To generate questions using text templates, we need to take care of the tense, person, and plurality to avoid grammar mistakes. We take these three elements as additional parameters of the activity in the text templates. After generating a valid question, we also apply a synonym change in order to increase language diversity.

The functional program can be directly applied to the scene lists of the sensory scene to get the correct answer, which is then translated into natural language. At this stage, the sensory context, the question, and the answer are all generated.

Question Type and Answer Balancing. We slide the time window across the untrimmed time-series sensory data to get various sensory scenes. For each sensory scene, we construct its semantic representation *scene lists*, and based on the activities involved in this scene, we apply all possible functional programs to the scene lists to get the correct answers. Sometimes the questions are ill-posed, and the answers could not be obtained. For example, the question "What did the user do after closing the door?" would be *ill-posed* if the user closed the door more than once in the time window, and we could not identify which "close the door" activity the question is referring to. These ill-posed questions are rejected.

After traversing all the time-series data and iterating all the possible functional programs, we generate an initial SQA dataset, which includes around 35 million question & answer pairs and 1362 sensory contexts. However, the current SQA dataset has an unbalanced question type distribution and biased answer distributions. This would lead to inefficient training and possibly performance degradation. Therefore, we downsample the questions based on their types to control the question data type composition.[2]

More importantly, we balance the global answer distribution of the dataset, to avoid question-conditional biases, which allow learners to make educated guesses without understanding the sensory contexts. To do this, we first calculate the global answer distribution for each question type, and then downsample the questions with the most frequent answer to a pre-defined ratio of 80%. We repeat this operation iteratively, until the stopping criteria are met: the question numbers, or the ratio of the most frequent answer is below certain pre-set thresholds.

After question resampling and answer balancing, the SQA dataset has more balanced answer distributions and question type composition. Details are illustrated in Figure 4.4.

4.4.3 Summary of Generated OppQA Dataset.

To create the OppQA dataset, we use a total of 16 question families and more than 110 text templates. Table 4.1 shows examples of generated questions for different types, and also their possible answers.

Table 4.2 shows the statistics of OppQA dataset. It includes 1362 unique sensory scenes, and a total of 91 thousand questions, 72 thousand of which are unique. Among them, 39 thousand unique queries are used. The question query represents the functional program used to generate the question. Diverse questions with the same semantic meaning share the same query. As shown in Figure 4.4 (B), OppQA has an average question length of 18 words, and is complicated enough compared with other popular VQA datasets. It can be

[2]We exclude the questions of "Time Query" type in OppQA dataset, since these questions have non-categorical answers. SQA models with regression tasks are left to future work.

Q Type	Question Example	Possible Answer
Existence	Is it true that the user closed the door after opening the door?	Yes, No
Counting	How many times did the user close the door?	<integer >
Action Query	What did the subject do after cleaning the table?	open the door, wash dishes, drink water, ...
Time Query	How long did the user wash dishes?	<float >
Action Compare	Confirm if the user performed the same action proceeding and following opening the fridge?	Yes, No
Number Compare	The user toggled the switch for the same times before and after drinking water?	Yes, No

Table 4.1: Question Examples and Possible Answers in OppQA.

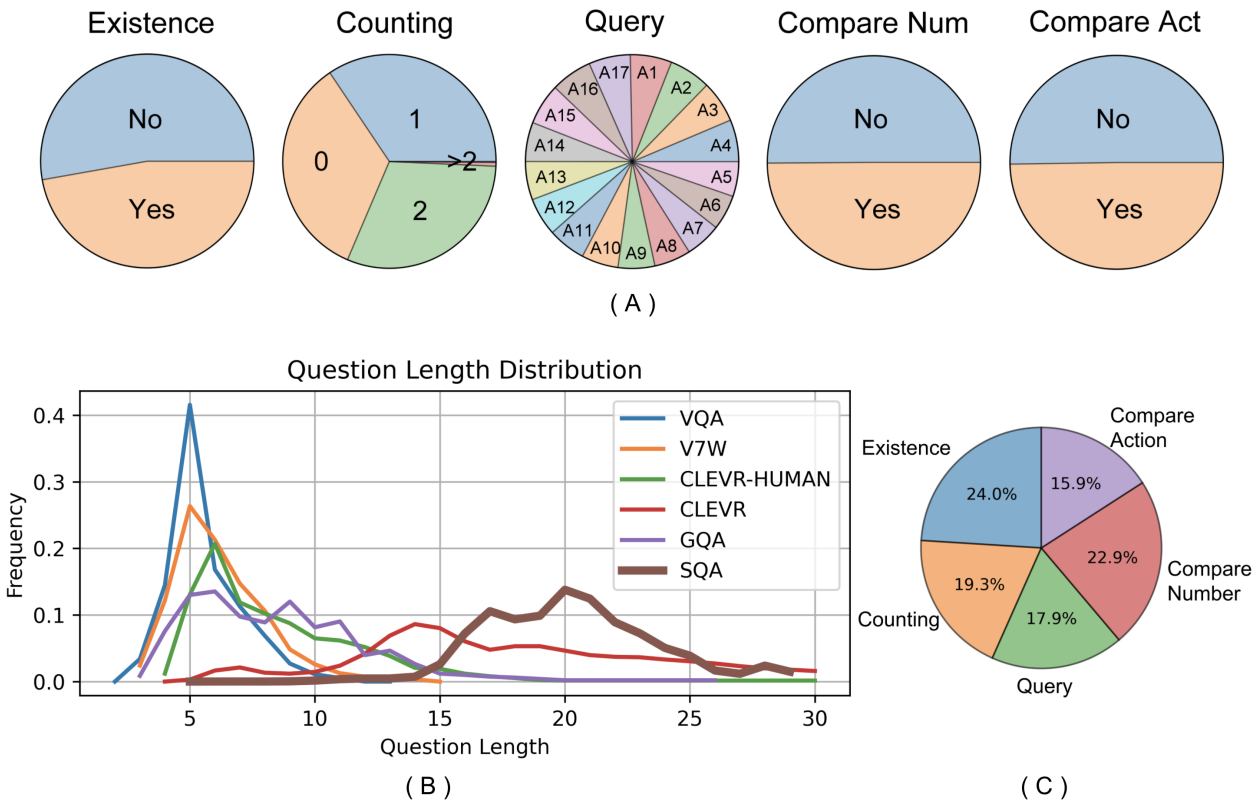

Figure 4.4: Statistics of OppQA dataset. (A):Global answer distribution of different question types. (B): Question length distribution compared with other VQA datasets. (C): Question type composition. Question type "Time Query" with non-categorical answers is excluded in this figure.

used as a benchmark for evaluating SQA models.

The OppQA data is split into a training set and a testing set. The training set contains SQA data generated on the first two ADL runs and a drill run of users 1-4, and the rest of the runs are used to generate testing data. The training data and testing data do not share sensory context, but they have overlapping question queries. Table 4.2 lists the details of OppQA.

Split	Sensory Contexts	Questions	Unique Questions	Unique Queries
Total	1,362	91,412	72,936	38,922
Training	730	74,470	62,789	35,262
Testing	632	16,942	14,643	6,275

Table 4.2: Statistics for the OppQA dataset.

4.5 DeepSQA Models and Implementations

In this section, we describe the structures and implementation details of the proposed SQA models, along with a set of baselines.

All of the models that process context sensory data first use a convolutional-LSTM network to get the sensory representations. The ConvLSTM network is composed of two convolutional modules(which contain a convolution layer with 1×3 kernel, ReLU activation and a maxpooling layer), followed by an LSTM layer and a fully-connected layer to generate a 128-dimension dense embedding vector. Both the LSTM layer and fully-connected layer have 128 hidden units. Instead of processing sensory data from different sources using different neural networks, we perform an early fusion on 77 channels of sensory reading from all the seven distributed sensors, which leads to better performance on this dataset. Therefore, the size of input sensory data is 77×1800, with 1800 specifying the window length of a one-minute window. This parameter is also changed in later simulations when we evaluate the SQA performance with respect to the task complexity.

All the models, that process questions, use LSTM-based models to get question representations. Every word in the question is first embedded into a 300-dimensional representation using a pre-trained GloVe[83] word embedding matrix, unless otherwise noted. The GloVe word vectors are pre-trained with six billion tokens with a vocabulary of 400K words. We use in total two different LSTM-based structures to extract question representations. Details will be discussed below.

In the evaluation, we exclude the questions asking about the duration of activities, which give various non-integer numbers as answers. We then formulate the SQA task as a multi-class classification problem[4], using the top 27 answers as the classification labels. Therefore, the output of every model should be a distribution of predicted scores of these candidate answers.

4.5.1 Baseline Models

We adopt a representative subset of methods from VQA: baselines that predict answers based on statistics of training data (Prior and Prior-Q), baselines using only question data (LSTM), or using only sensory data as input (ConvLSTM). Since sensory data are opaque to humans, we cannot use the crowd-sourced method to collect human answers and evaluate human performance on this task. As an alternative, we use a Neural-Symbolic approach with pre-trained native sensory classifiers, and perfect question logic knowledge to mimic human performance. These methods are described in details as follow:

- **Prior:** As [4], this baseline answers the most frequent answer in the training dataset for all questions, which is 'No' in OppQA dataset.
- **Prior-Q:** Similar to the *Prior* method, this model predicts answers based on training data statistics. For each question type, it predicts the most frequent training-set answer.
- **LSTM:** Similar to the "LSTM Q" in [4], questions are first processed with learned word embeddings, and analyzed by a word-level two-layer LSTM model, where each layer contains 128 hidden units. The output question representation, which is the final state of LSTM, is then fed into an MLP network using two hidden layers with 128 units to predict the distribution over answers. This method uses no context information as it is "sensory-blind," so it can only model question-conditional bias.
- **ConvLSTM:** Inspired by [55] and [4], this "question-blind" model only uses sensory context. Sensory data are first processed using the ConvLSTM network, and then the

answer is predicted by an MLP with one hidden layer that has 128 units and a softmax output layer.

- **Neural-Symbolic:** In this method, we employ two activity classification models recognizing user activities and locomotion using sensory data at every time step. We then use 100% correct, hard-coded logic rules to analyze the classification result and answer the question. These rules are functional modules in Python, hosting all logic operations, and are selected based on different questions. The activity classification models are ConvLSTM structures that contain two convolutional layers, followed by an LSTM layer and a fully-connected layer to map the sensory data to the predicted activity. The convolutional kernel size is 1×3, and two convolutional layers are followed by batch normalization layers. Sixty-four feature maps are used in each convolutional layer, and 32 hidden nodes in the LSTM layer. The activity classification models are trained natively on the original OPPORTUNITY dataset, with an accuracy of 80.13% and 74.13% on the activity classification and locomotion classification tasks, respectively. These numbers could represent the performance of state-of-the-art sensory models on this dataset.

4.5.2 DeepSQA-based models

Based on the DeepSQA framework, we propose and evaluate three different models that process both sensory contexts and questions together, analyze and reason about the spatial-temporal relationships, and draw the final answers. These models are chosen as representatives, as they employ the state-of-the-art mechanisms and structures in the multimodal deep learning and the question answering domains. The model DeepSQA-ConvLSTM, uses a simple but effective elementwise multiplication to fuse the question and sensory representations together and predict answer based on it. The model DeepSQA-SA, learns how to search for the features in sensory contexts that are related to the answer using attention weights calculated based on the questions. The model DeepSQA-CA, decomposes the questions into multiple explicit reasoning steps, customizes the weighting of the context feature

vector for each step, and performs iterative reasoning processes to get the answer. Details of these models are discussed as follows.

DeepSQA-ConvLSTM: This model combines the ability of both sensory processing and question reasoning in the LSTM and ConvLSTM baselines. Since the representations of both modalities have the same dimension of 128, the model first fuses the sensory and question representations using an element-wise multiplication, and then feeds the combined feature to a two-layer MLP (with 128 hidden nodes) to get the prediction of the answer distribution.

DeepSQA-SA: In this model, we use the Stack Attention(SA) [126, 59] mechanism to fuse the sensory and question information, and then predict answers based on new attended features.

Specifically, the questions and sensory contexts are first processed using an LSTM and a ConvLSTM models to get compact representations in the latent space. These two representations are then concatenated together as a combined feature. In the Stack Attention network, we use a two-layer CNN with 1×1 kernels to calculate the spatial attention weights. The first and second convolutional layers' activation functions are ReLU and Softmax, to calculate the normalized probabilities over all the spatial locations.

The calculated attention weights are then multiplied with the sensory representation to get the attended sensory feature, where the information relevant to the question is highlighted.

We use a glimpse number of two to get two sets of different attended sensory features. These features are concatenated with the question representation again and fed to a 2-layer MLP network with 1024 hidden nodes. The final answer is predicted by the last softmax layer.

DeepSQA-CA: This method uses the Compositional Attention (CA) [48] mechanism to perform multi-step reasoning over the sensory and question data, by employing a recurrent structure which strings together p MAC cells, each responsible for performing one reasoning step. A MAC cell is the basic module in this recurrent architecture.

In this model, the sensory contexts are processed using the same ConvLSTM network, while the question data are processed differently. The original question with word tokens is first converted into a sequence of word embeddings. Instead of using pre-trained GloVe word embeddings, we learn the embedding matrix with other network parameters during the training time. The embedding sequence is then processed by a bidirectional LSTM network, which trains two instead of one LSTMs on the input sequence in two opposite directions. The question representation is then the final hidden states' concatenation from the forward and backward LSTM passes. In addition to the question representation, we also generate a contextual word sequence, by storing the sequence of output states for each word in the biLSTM network.

After getting the sensory and question representations, we feed them to the recurrent MAC structure to perform multi-step reasoning. Each MAC cell is composed of a control, read, and write unit, which operate over control and memory hidden states. The control unit attends to different parts of the question, to update the control state that represents the reasoning operation at each time step. The read unit extracts relevant information out of the sensory context with the guidance of the control state. The write unit integrates extracted information into a memory state, which becomes the input of the next MAC cell.

The input of a MAC cell are the previous control and memory states, together with the sensory context representation, question representation, and contextual words. The initial control and memory states are initialized learned parameters.

At last, the final memory and question representation are concatenated and fed to a 2-layer MLP with a softmax classifier to predict the distribution over candidate answers. An overview of this model is demonstrated in Figure 4.5.

In our implementation, we set the reasoning step value p as 12, which means that the recurrent structure has 12 MAC cells. The dimension of hidden states (control and memory) is 512, and the final MLP network has two layers with 1536 and 512 hidden nodes.

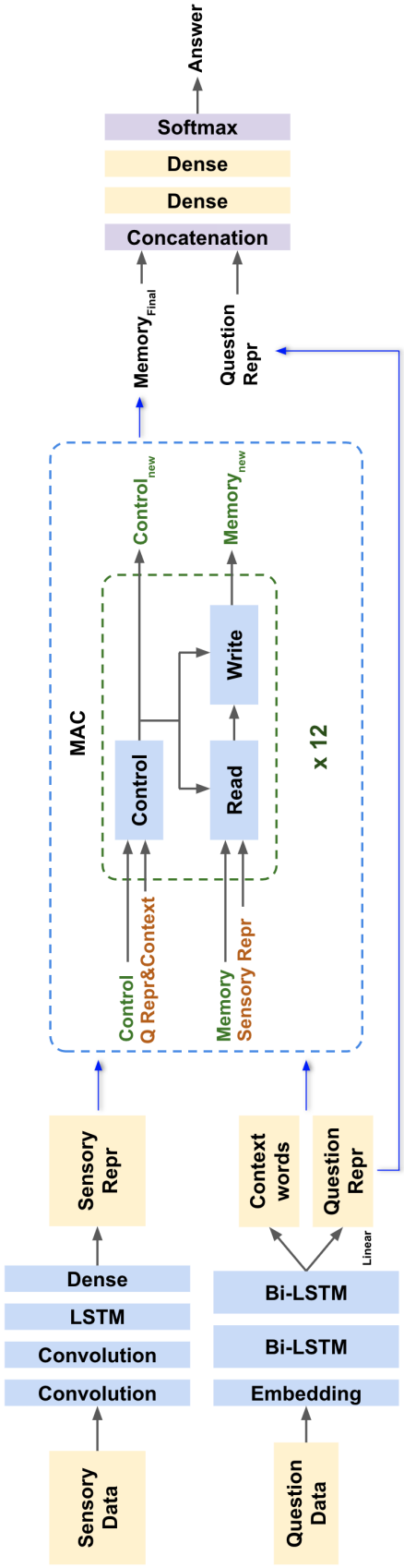

Figure 4.5: Overview of DeepSQA-CA Model

4.6 Evaluation

In this section, we empirically evaluate the proposed DeepSQA framework by comparing the model performance with baseline methods. We use the created OppQA dataset and its variants to test the effectiveness and robustness of SQA models on reasoning about the spatial-temporal dependencies of human activities.

4.6.1 Implementation Details

In our experiment, the DeepSQA-CA model is implemented in Pytorch, and the other models are implemented with TensorFlow and Keras frameworks. All the models are trained and tested on a desktop machine with two Nvidia RTX Titan GPUs. During the training, a 0.15 dropout rate is applied to the convolutional, dense, and LSTM-based layers for the ConvLSTM and MLP models, and a $1e-4$ weight decay is applied to the DeepSQA-CA model to avoid overfitting. We train all the models for 40 epochs using Adam optimizer with a learning rate equal to $1e-4$ and a batch size of 64.

4.6.2 General Observations

We first evaluate the DeepSQA models and baselines on the standard OppQA dataset, which covers questions of 15 types, and reasons about human activities over a time window of one minute (1800 samples). The maximum question length is 31 words, and the number of unique candidate answer is 27, with non-integer answers excluded. The overall performance of each method is shown in Table 4.3. We break down the performance based on different question types: existence, counting, action query, number comparison, and action comparison. Among them, counting and query are further classified into open-ended questions and the remaining into binary questions.

The *Prior* method in the first column predicts "No" for all the questions, and gets a 0% accuracy on open-ended questions and 53.27% on binary questions, which is a little bit higher than 50% of a random guess. This indicates that the answer distribution in the OppQA

	Baselines					DeepSQA		
	Prior	**PriorQ**	**Neural Symbolic**	**ConvLSTM**	**LSTM**	**SA**	**ConvLSTM**	**CA**
Overall	41.57%	54.82%	42.75%	44.92%	65.04%	59.74%	67.63%	**72.38%**
Binary	53.27%	65.26%	51.10%	57.56%	68.33%	61.78%	71.81%	**76.51%**
Open	0.00%	17.74%	13.09%	0.00%	53.35%	52.49%	52.78%	**57.67%**
Existence	66.63%	66.34%	46.15%	39.97%	66.99%	67.20%	69.76%	**72.69%**
Counting	0.00%	35.99%	30.86%	0.00%	60.71%	58.94%	59.06%	**63.31%**
Action Query	0.00%	4.29%	0.00%	0.00%	47.92%	47.74%	48.16%	**53.52%**
Num Comparison	53.59%	72.45%	66.61%	63.21%	70.73%	63.57%	71.91%	**76.61%**
Act Comparison	37.99%	37.99%	0.00%	55.64%	61.02%	49.57%	73.62%	**80.21%**

Table 4.3: Overall results of models trained and tested on OppQA dataset

dataset is balanced with minimum global bias. Taking question types into consideration, the *PriorQ* method gets better performance, but it still could not predict answers to open-ended questions well. On the "Action Query" question type, *PriorQ* can only get an accuracy of 4.29%, proving that it is indeed a challenging task to predict the correct answer out of 27 candidate answers.

Albeit using pre-trained activity classification networks and perfect reasoning logic, the *Neural Symbolic* method show bad performance on OppQA. The primary reason is that the deep learning models cannot make accurate inferences on the Opportunity dataset (with an accuracy around 70-80%). The errors in the noisy inferences accumulate and then dramatically reduce the answer correctness. On the other hand, the end-to-end neural network approaches do not suffer from this problem. One possible explanation is that the neural network could learn to use reliable features, and compensate for the possible error to make the right prediction.

Generally, the models with DeepSQA framework show better performance than the baselines, especially the *DeepSQA-CA* model, which uses the compositional attention mechanism, gets the best performance on all of the question types, with an overall accuracy of 72.38%. The *DeepSQA-SA* model based on stacked attention shows inferior performance than some of the baselines. This proves that, although demonstrating good performance on visual QA tasks, the stacked attention does not work well on reasoning about temporal dependencies between human activities.

Need to mention, the *LSTM* model could get an over 60% overall accuracy without using any sensory context data, and it could even get good performance closer to some of the DeepSQA models. It is because that for some particular questions, the local bias still exists. For example, when the question is "What did the user do after opening the door?" the answer would be "Close the door" with a 90% probability. Since we are using real data, it is inevitable to contain bias induced by natural human behavior patterns.

Performance on Prime Dataset. Because of the unavoidable local bias in the dataset, we construct a "Prime Testing Set" based on the original testing data. In this *prime dataset*,

we abandon the questions with only a single answer, and all the questions have more than one answer. This makes it impossible for the models to predict answers using only question data and without sensory context. This prime set is actually more challenging than the original testing set. In Table 4.4, we list the performance comparison of all the deep-learning-based models on the testing dataset and prime dataset. Generally, the accuracies on the prime dataset are lower than those on the testing set. DeepSQA-CA has the minimum performance degradation while maintaining good accuracy. Its accuracy loss is less than 2%. The baseline *ConvLSTM* model observes a performance improvement on the prime dataset. It is because that the non-prime questions with a single answer are answered incorrectly by this model, and removing those questions could passively increase the accuracy.

	Baselines		**DeepSQA**		
	ConLSTM	**LSTM**	**SA**	**ConvLSTM**	**CA**
Testing	44.92%	65.04%	59.74%	67.63%	**72.38%**
Prime	56.56%	60.39%	53.46%	64.90%	**70.48%**

Table 4.4: Overall performance on prime testing set

Robustness Against Linguistic Variations. The key motivation of proposing the sensory question answering task is to improve the inferencing flexibility of sensing systems, in a way that users can make arbitrary inferences during the run time in the form of natural language. However, for questions with the same semantic meanings, the natural language representations might be dramatically different. In order to test the robustness against linguistic variations of SQA systems, we create a "Rephrasing Testing set." This dataset contains different rephrasings for all the questions in the original testing set, generated using our question generation tool. We evaluate the four DL-based models which take natural language question as input on the rephrasing dataset, and list the corresponding accuracies in Table 4.5. Basically, the performance on the rephrasing set is similar to the testing accuracy, indicating that DL-based SQA models are robust to question rephrasing. In addition, we

define a consistency score to further measure the model robustness on every query:

$$Consistency = \frac{\sum_{i=1}^{m} \mathbb{1}(a_i = a_{major})}{m}$$

The m represents the number of rephrasings for each question query, and a_{major} is the majority answer predicted by the SQA model. Table 4.5 shows that the consistency scores for all the models are greater than 95%, which proves their robustness to linguistic variations.

	LSTM	**DeepSQA SA**	**DeepSQA ConvLSTM**	**DeepSQA CA**
Testing Acc	65.03%	59.73%	67.63%	**72.38%**
Rephrasing Acc	65.86%	60.51%	68.56%	**72.86%**
Consistency	99.13%	98.97%	99.06%	96.28%

Table 4.5: SQA robustness to linguistic variations

4.6.3 Analysis by Question Complexity

To access the SQA model performance with respect to question complexity, we analyze the evaluation result of deep learning models by the question length, as shown in Figure 4.6. The models that do not take the question as input are omitted here. Intuitively, longer questions should contain more query information, and would be more challenging to answer than shorter ones. Surprisingly, we do not see a clear correlation between question length and SQA performance. One possible reason is that the number of words in questions cannot effectively reflect the questions' complexity. Some questions are more verbose than the others, and contain redundant or even useless information, while representing simple queries. This would lead to the result that SQA models have better accuracies on some longer questions than those on shorter questions.

Instead of using the number of words as a question complexity measure, we use the number of required query operations to answer the questions to better capture the complexity. For example, questions like "What did the user do after washing dishes?" can be answered

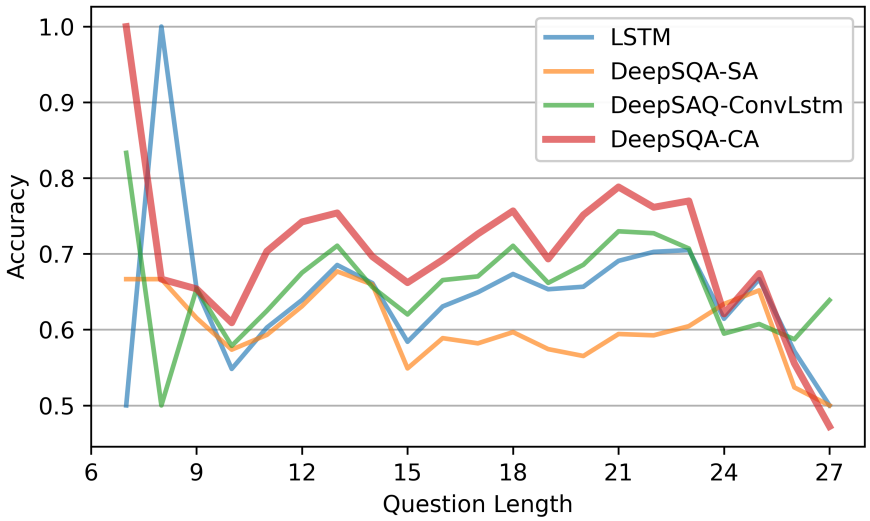

Figure 4.6: Models performance w.r.t. question length.

using three operations: 1. detect and localize the "washing dishes" activity; 2.Filter the activities after the "washing dishes"; 3. Query the type of filtered activity.

Since the binary and open-ended questions have different complexity, we first categorize the questions into two groups, and analyze them separately. As shown in Figure 4.7 (A), for binary questions, there is a linear decrease in SQA model performance with the number of operations increasing from one to three. However, the accuracies of questions requiring seven query operations are pretty high. It is probably because the number of this type of question is larger than others in the training dataset. Models trained on this training dataset learn to perform well on this type of questions.

Figure 4.7 (B) shows the SQA model performance changes on open-ended questions. Specifically, the accuracy of the DeepSQA-CA model decreases with the required query operations increasing. However, the other models show poor performance on questions requiring two operations. We notice that the SQA models are sensitive to the amount of training data. A larger training dataset would generally lead to a better performance. Apart from these, we can find that the DeepSQA-CA model shows the best performance consistently regardless of the complexity of input questions.

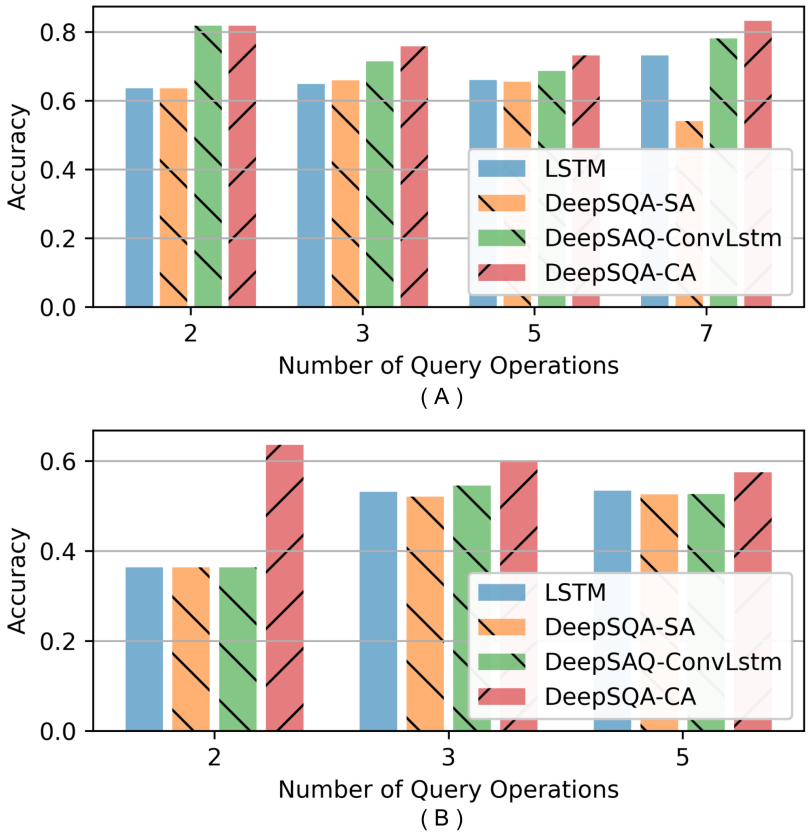

Figure 4.7: Models performance w.r.t. question complexity: (A) Binary and (B) Open-ended.

4.6.4 Analysis by Context Complexity

We also perform a set of experiments using new datasets generated in the same way as the original OppQA, but with different context window lengths. Here, we generate four variant datasets, using a window length of 500, 750, 1000, and 1500 time steps.

Apparently, the SQA task with shorter context length would be simpler, since fewer activities are involved in the scene, and reasoning about them is easier. In Figure 4.8, we plot the bar chart of performance for all SQA models that take the sensory context as input. When the sensor context window is only 500-time-step long, the models like DeepSQA-CA and DeepSQA-ConvLSTM could get excellent performance with over 90% accuracies. The performance of DeepSQA-ConvLSTM is even slightly better than DeepSQA-CA. It is because within a short period, the complex SQA reasoning module is not necessary.

With the length of sensory context window increasing, the accuracy of SQA models keeps decreasing, which means that the length of sensory context is a crucial factor affecting the SQA task's complexity. Generally, the DeepSQA-CA model shows the best performance consistently, especially when the sensory context is long and the task is challenging. This illustrates the effectiveness of the compositional attention mechanism in processing complex spatial-temporal dependencies on human activity reasoning.

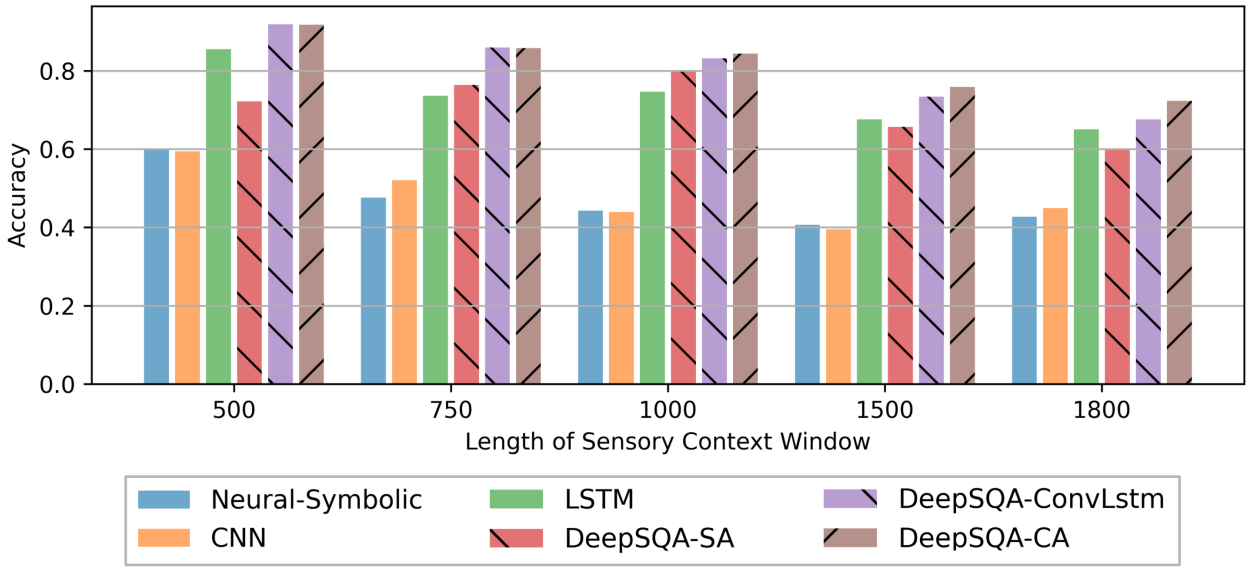

Figure 4.8: Model performance w.r.t. SQA task complexity.

4.6.5 Generalization to new questions

Another interesting characteristic of sensory question answering is its data-efficient training and generalization ability to new data. During the training, instead of learning to answer each specific question, SQA models learn to perform basic logical operations and predict answers based on operation results. At the inference time, SQA models are able to answer novel questions that they have never seen in the training dataset, but are composed of familiar activities and logic operations.

In this evaluation, we construct a new dataset OppQA-*generalize* to test the generalization ability of different SQA models. Specifically, this dataset adapts the same configuration as the original OppQA: with 77-channel data from 7 distributed sensors. We set the context

window length to 1800, and the stride to 600 during generation. We split the data based on sensory context and question query to construct the training and testing set. The training set covers 80% of the sensory contexts, and 50% of all the unique question queries. The testing set contains the rest 20% of the sensory contexts, and all the unique question queries. The testing set is further divided into *Test-familiar* and *Test-novel* sets. The *Test-familiar* has question queries overlapping with the training set, and *Test-novel* has queries different from the training set. The details of this dataset are described in Table 4.6.

Split	**Sensory Contexts**	**Questions**	**Unique Queries**
Train	726	38,121	17,587
Test-Familiar	625	8,715	3,420
Test-Novel	627	8,227	3,305

Table 4.6: Statistics for the OPPQA-*generalize* dataset.

We train different SQA models on the training set, and then test the model performance on *Test-familiar* and *Test-novel* sets separately. The accuracies are listed in Table 4.7.

	Test-Familiar			**Test-Novel**		
	Binary	**Open**	**Overall**	**Binary**	**Open**	**Overall**
Prior	46.90%	0.00%	36.64%	46.55%	0.00%	36.28%
PriorQ	70.11%	18.51%	58.82%	68.86%	19.02%	57.87%
Neural Symbolic	51.12%	12.69%	42.71%	51.08%	13.51%	42.80%
CNN	58.09%	0.00%	45.38%	58.01%	0.00%	45.22%
LSTM	67.80%	53.54%	64.68%	66.91%	46.86%	62.49%
DeepSQA(san)	69.89%	53.38%	66.28%	67.72%	49.01%	63.60%
DeepSQA(convlstm)	71.96%	53.49%	67.92%	71.71%	48.90%	66.68%
DeepSQA(mac)	**73.03%**	**58.42%**	**69.83%**	**72.70%**	**54.30%**	**68.64%**

Table 4.7: Model performance generalization to novel questions

As shown in the table, most models have a performance degradation when facing the

unseen question queries in the *Test-novel* dataset. The degradation is more evident for open-ended questions, where the accuracies drop by around $4-7\%$, compared with $0-2\%$ for binary questions. The DeepSQA-CA model continues to show the best performance on both *Test-familiar* and *Test-novel* testing sets, and it has the lowest performance degradation. This result demonstrates that the DeepSQA model has better generalization ability to new question queries.

4.6.6 Discussion

Generation of Natural Language Answer. In this work, we formulate the SQA problem as a classification problem, where the output classes are the top K answers appearing in the training dataset. The DL-based SQA models predict the answer distribution on the K classes, and the class with the highest probability is selected as the predicted answer. Several limitations exist in this approach. Firstly, the regression-type questions, such as Query-Time, are not enabled by the SQA models. Secondly, although the SQA models support answering diverse questions of different types by enumerating the possible answers in the training set, the output is still constrained to a limited set. In future work, we will augment the DeepSQA with an answer generator, which takes processed information from the proceeding reasoning module and creates natural language answers.

Neural-Symbolic DeepSQA framework. At the current stage, we use an end-to-end neural network structure for modeling the SQA task. So the prediction made by the model is not explainable. A significant drawback of this purely data-driven approach is that it's data-intensive during training, and also memory-consuming for storing the learned dependencies. Based on the evaluation result, we can observe that model performance degrades when facing longer sequences (e.g., longer sensory time windows, complex question queries). In future work, we would like to introduce the idea of neural-symbolic system to the SQA task, to get a more compact symbolic representation, and robustness on complex scenarios.

Evaluation on Different Sensory Datasets. In addition to OppQA, we also generated another SQA dataset using ExtraSensory[108] as source data. However, most of the SQA

models show less satisfied performance on it. The poor performance is due to the challenge of deep learning models extracting useful information from ExtraSensory data. The model trained on ExtraSensory natively could only get around $< 60\%$ accuracy on the activity classification task. Consequently, the sensory representation could not provide much information to the DeepSQA. Admittedly, the vibrancy of visual question answering research largely depends on the mature visual information processing ability. We believe that the proposed DeepSQA framework and SQA-Gen tool in this paper could help facilitate the SQA research.

Generalization to Complex & Distributed Scenarios. Thus far we have only discussed an evaluation on OppQA, where multimodal data from different sensors are fused together such that a single sensory processing module is used in DeepSQA models. We believe that it can be extended to SQA scenarios with multimodal data dispersed far apart. The complex spatial-temporal relationship between sensory data needs to be explored and reasoned to get the correct answer. In future work, we will deploy DeepSQA system in complex real sensor network scenarios, with more rising challenges like time synchronization and sensory fusion.

4.7 Conclusion

In this work, we present DeepSQA, a generalized framework for sensory question answering. By taking both sensory data and natural language questions simultaneously, DeepSQA is able to identify the tasks specified by questions and perform reasoning on sensory data accordingly. It reduces the laborious work of training new deep learning models when new tasks are introduced, and improves the inferencing flexibility. To evaluate SQA models, we introduce SQA-Gen, which automatically generates SQA datasets using labeled source sensory data. Based on this tool, we propose the OppQA dataset for benchmarking SQA model performance. Results on OppQA prove the effectiveness, reliability, and robustness of DeepSQA. Future work will focus on evaluating DeepSQA in real-world scenarios with distributed and multimodal sensory data.

CHAPTER 5

Enabling Edge Devices that Learn from Each Other: Cross Modal Training for Activity Recognition

5.1 Introduction

Edge devices are typically equipped with a wide variety of sensing modalities for tracking environmental markers. To provide insights and enable context-aware applications (e.g. user activity recognition [125], workout tracking [96], speech recognition [35]) the data collected on these devices are used to train deep neural network models. However, to fully realize the learning-at-the-edge paradigm, several challenges still needs to be addressed. In particular, the model training process needs to handle insufficient labeled data, and the heterogeneity in inter-device sensing modalities.

As a step towards addressing the above concerns, we propose RecycleML– a mechanism to transfer knowledge between edge devices. Our approach is guided by the observation that application-specific semantic concepts can be better associated with features in the higher layers (close to the output side) of a network model [26]. This observation allows us to conceptualize the layers of the different networks as an hourglass model, as shown in Figure 5.1. The lower half of the hourglass correspond to the lower layers (close to the input side) of the individual models (trained on specific sensing modalities). The narrow waist is the common layer (latent space) into which the lower layers project their data for knowledge transfer. The upper half of the hourglass comprises of the task-specific higher layer features which are trained in a targeted fashion for task-specific transfer.

To evaluate RecycleML, we emulate edge devices with three sensing modalities - vision,

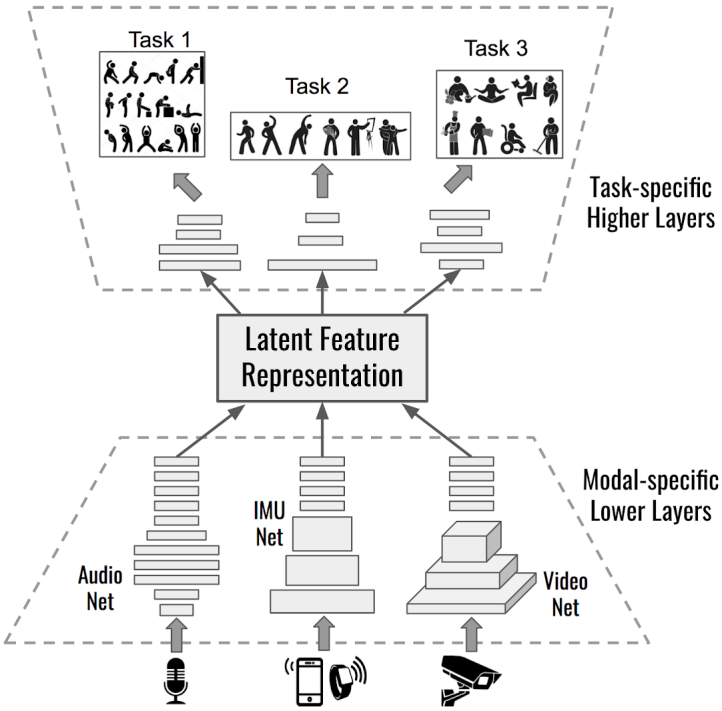

Figure 5.1: Shared representation between edge devices.

audio and inertial (IMU) sensing as shown in Figure 5.2. We perform zero-shot learning [101], i.e. use zero training labels, across different sensing modalities when they are performing the same classification task. We achieve this by training the target edge device model to have the same latent space as the source model. RecycleML can also learn to expand the classification tasks of the transferred model with very few training examples.

Our results across a mix of sensory substitutions and task transfers show that, over our collected CMActivity dataset, RecycleML reduces the amount of labeled data required to train edge devices by at least 90% and speeds up the training process by up to 50 times after doing knowledge transfer using unlabeled data.

Our contributions are as follows:

1. We combine the idea of transfer learning (lower layers transfer) with sensory substitution (higher layers transfer) together and propose a unified framework, where the knowledge in every part of a network could be transferred.

2. We introduce a new dataset CMActivity that have synchronized data of three modalities: vision, audio, and inertial.

3. For activity recognition task, we verify that the shared representation exists for time series sensory data, and it can help transfer knowledge from ambience edge devices to wearable edge devices and vice versa. The code for our experiment is available on-line.[1].

5.2 Method Overview

5.2.1 Conceptual Scenario

Suppose Alice has an edge device D_{V1} with camera in her living room, and it is trained to do activity recognition. Alice wants to replicate the inferencing ability of D_{V1} on other devices: a smart watch D_W which she wears regularly, an acoustic device D_{A1} in her living room to turn off D_{V1} whenever needed due to privacy reasons, and a camera D_{V2} and a voice assistant D_{A2} in her office, Our objective is to transfer activity recognition knowledge of D_{V1} to D_{A1} and D_W (Video→Audio and IMU), and later, transfer activity recognition knowledge of D_W to D_{A2} and D_{V2} (IMU→Audio and Video).

5.2.2 RecycleML Description

RecycleML uses the same latent feature representation across edge devices of different modalities to do knowledge transfer. Knowledge transfer uses *synchronous unlabeled data* to map the input of untrained model to the shared latent feature representation of the pre-trained model (details in Section 5.2.2.1). Later edge devices can either reuse the upper layer across models or do task transfer on the upper layers if needed (details in Section 5.2.2.2).

[1]https://github.com/nesl/RecycleML

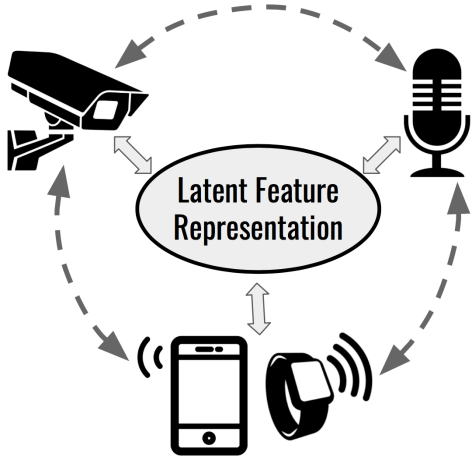

Figure 5.2: Knowledge transfer across edge devices with different sensing modalities.

5.2.2.1 Knowledge Transfer

For simplicity, let us consider two edge devices D_X and D_Y, each with different sensing modality capturing data X and Y respectively. Suppose D_X has a pre-trained model M_X and performs task T_X. Our goal is to train a new model M_Y for D_Y to perform task T_Y. To transfer knowledge from D_X to D_Y, we collect data X and Y from both devices while observing the same event. X and Y need not be labeled. An important requirement is the time synchronization in devices D_X and D_Y so as to capture the same event in their data X and Y. Synchronization is natural in different sensing modalities. For example, vision, audio and inertial sensors observing the same event of human motion can capture it in different signals (see Section 5.3.1 for details).

We input data X to the pre-trained model M_X, and instead of getting the final output value, we calculate the activation values $f(X)$ of an intermediate layer that acts as our shared latent feature representation. f is the transformation of all the early layers before the specific activation. We use $f(X)$ as the training value for the model M_Y of device D_Y. Specifically, we choose a new network g, specialized for input modality Y, and train the network $g(Y)$ so that it maps Y to the same shared latent feature representation by minimizing $|g(Y)-f(X)|^2$ as our loss function. We generate the model M_Y for device D_Y by adding the task specific output layers to g. In this way, model M_X teaches the new model M_Y in a teacher-student

data distillation manner [43].

5.2.2.2 Task Transfer

Transferring knowledge from device D_X to D_Y does not need any ground-truth labels. However, the new model M_Y for device D_Y may need additional information before performing any classification or regression task. Therefore, three different scenarios arise when devices D_X and D_Y performing tasks T_X and T_Y in classification settings respectively: (i) Devices D_X and D_Y are performing same tasks T_X, (ii) Devices D_X and D_Y are performing related tasks T_X and T_Y, e.g. where T_X and T_Y are both human activity inferencing but with different numbers of categories, and (iii) Devices D_X and D_Y are performing completely different tasks T_X and T_Y. In this paper, we study how to transfer knowledge between devices in the first two scenarios.

We explore two different methods of task transfer:

- *PureTransfer* directly uses the higher layers of model M_X for new model M_Y. In this case no further training is needed and no labeled data is required.
- *Transfer+LimitedTrain* freezes the network g and adds higher layers to M_Y and retrains only the higher layers using limited labeled data.

In the first scenario, since the tasks are same we can use both methods. In the second and the third scenarios, direct transfer of higher layers from model M_X to model M_Y does not work as M_X does not give the same desired output. Hence, we use the second method. In our experiments, we evaluate scenario (i) of task transfer using both methods of *PureTransfer* and *Transfer+LimitedTrain* and scenario (ii) using *Transfer+LimitedTrain*.

In our experiments, we used the output of last hidden layer after removing the final output layer from model M_X as the f transformation. Here f and g serve as shared latent representations across modalities. We add a single task specific layer to g to generate model M_Y. In future, we will explore the different choices of f and addition of multiple task specific output layers to g.

5.3 Evaluation

5.3.1 Dataset

For our experiments, we collected a new dataset, called CMActivities, composed of videos for vision and audio modality, and corresponding IMU data (accelerometer and gyroscope) from sensors on left and right wrist. We collected 767 videos of roughly 10 second each from 2 users[2] doing 7 different activities at 6 locations. Every video contains a single activity and is used to label the vision, audio and IMU data. The total duration of collected data for each modality is 125 minutes.

Table 5.1: Description of CMActivities dataset

Activity	Number of Videos	Duration (sec)
Go Upstairs	162	1338
Go Downstairs	161	1113
Walk	119	1143
Run	115	891
Jump	73	995
Wash Hand	73	1070
Jumping Jack	90	958

We collected the videos of the user using an observer smartphone. The wrist sensors communicate the data to the smartphone of the user doing the activities. The IMU data was timestamped by user's smartphone and the video by the observer smartphone. Time synchronization between vision and audio is naturally present because both are extracted from the same videos. However, time synchronization between the user smartphone and the observer smartphone is needed so as to synchronize video and IMU data. In our data collection, we used the default smartphone timestamps synchronized through the Network

[2]The data is collected from the authors and thus does not require approval from IRB.

Time Protocol (NTP) [80] service, and observed a maximum time difference of 0.5 seconds between the observer smartphone and the user smartphone. We leave it for future to explore the effect of poor time synchronization across devices in observing the same event. We expect the knowledge transfer capabilities of RecycleML to degrade as the time difference between devices increases.

The details of CMActivities are shown in Table 5.1. The data collection was done at different locations with two users wearing separate set of clothes at each location so as to make sure that the trained classifier learns the activity features and is least affected by the environmental factors. We split 767 videos and IMU sessions into three parts: training dataset (624), testing dataset (71) and personalization dataset (72). Training and testing datasets contain 7 activities at 5 different locations and personalization dataset contains 5 activities at 6^{th} location. We don't have *Go Upstairs* and *Go Downstairs* activities in the personalization dataset.

The training dataset is further split into 3 parts: Pre-Training set, Transfer set and LimitTrain set. The personalization dataset is split into PersonalTrain and PersonalTest sets. The testing dataset is used only for evaluation. The frame rate of video is 29 and the sampling frequency of audio and IMU is 22050 Hz and 25 Hz respectively. We use a window of 2 seconds to extract vision, audio and IMU features from dataset with sliding window of 0.4 seconds between consecutive windows. In case of vision and IMU, we use raw features directly as input to the models. We extracted features from the raw audio data using Librosa [79] and use it as the input features. Specifically, we extract mel-frequency cepstral coefficients (MFCC) [74], power spectrogram [27], mel-scaled spectrogram, spectral contrast [52] and tonal centroid features (tonnetz) [40].

In total, we have 11976 samples in training (5000 samples for Pre-Training set, 6000 samples for Transfer set, and 976 samples for LimitedTrain set), 1377 samples in test and 1592 samples in personalization (475 samples for PersonalTrain set and 1117 samples for PersonalTest set) for each modality.

Table 5.2: Testing accuracy of baseline models

Input Modality	Video	Audio	IMU
Accuracy	90.92%	92.81%	90.99%
Number of parameters	4.6M	0.8M	57K

5.3.2 Baselines

To compare the results of RecycleML, we trained Video, Sound and IMU models using Pre-Training dataset individually to do activity recognition. The models we use are the state-of-the-art deep learning architectures that are generally adapted in a wide range of applications:

(a) Video Network is a reduced version of C3D [107] network. It includes four 3D-convolutional modules combined with 3D-maxpooling layers, followed by 3 fully-connected layers and one output layer. The total number of parameters are about 4.6 million.

(b) Audio Network is a multi-layer perceptron model. It has 10 fully-connected layers and a total of 810 K parameters. We add drop-out to avoid overfitting.

(c) IMU Network is a CNN network. It has 2 convolutional modules (convolution layer + maxpooling layer), 3 fully-connected layers and a output layer. 57K parameters are trainable in this network.

Table 2 shows the summary of the individual models. The models are trained using the training dataset and tested on testing dataset. These baseline models are trained using SGD [13] and Adam [61] optimizers with a learning rate of 0.001. We save the models with best test accuracy after training for 500 epochs.

5.3.3 Knowledge Transfer Results

Knowledge transfer results are presented in Table 5.3. In the first and second experiment, vision device D_{V1} is trained while acoustic device D_{A1} and wearable device D_W are untrained respectively. In the third and fourth experiment wearable device D_W is trained while vision

Table 5.3: Comparison of knowledge transfer between devices.

Significance tests (compared to the training from scratch) are carried out using t-test with P¡0.005 in most cases.

Transfer	Trained-Device	Pure-Transfer	Transfer+LimitedTrain	Training from Scratch
Video(D_{V1}) to Audio(D_{A1})	90.92%	90.20%	**90.36%**	84.12%
Video(D_{V1}) to IMU(D_W)	90.92%	94.19%	**94.37%**	70.73%
IMU(D_W) to Video(D_{V2})	90.99%	74.00%	**75.13%**	72.26%
IMU(D_W) to Audio(D_{A2})	90.99%	84.82%	**87.82%**	84.28%

device D_{V2} and acoustic device D_{A2} are untrained. For each of these four transfers, we follow the same procedure. Taking vision device D_{V1} to acoustic device D_{A1} as an example, we first train the vision model of D_{V1} from scratch using the Pre-Training set (5000 samples) of training dataset. We use the standard SGD optimizer with a learning rate of 0.001. The training is finished in 500 epochs. We then use D_{V1} as a pre-trained device to transfer knowledge to a D_{A1} following the procedure described in Section 5.2.2.1. In the knowledge transferring process, we use Adam optimizer with a learning rate of 0.001, and run it for 500 epochs. The data used in transfer process are the synchronized unlabeled vision and sound data from Transfer set (6000 samples) of training dataset. After transfer, the higher layers of audio model can be created using two methods *Pure-Transfer* and *Transfer+LimitedTrain* discussed in Section 5.2.2.2 when both D_{V1} and D_{A1} are doing the same task. In *Pure-Transfer* method audio model uses the output layer of vision model directly. In *Transfer+LimitedTrain*, we train the new output layer for audio model. We select a small labeled set of 500 samples randomly out of 976 samples from LimitedTrain set of training dataset and name it LimitTrainSet. We use the LimitTrainSet to train the output layer of audio model for 100 epochs using Adam optimizer. As a comparison, we also trained an audio model from scratch using the same LimitTrainSet for 500 epochs. We use more epochs for training from scratch as it takes more time to converge. The other three transfers are tested in the same way. The Audio and IMU models which are trained from scratch use Adam optimizer.

Note: In Video to IMU transfer, it takes more time to transfer the knowledge, so we perform the knowledge transfer for 1000 epochs. In real implementations, the knowledge transfer process for edge devices can either be done in background or at the server using unlabeled data, so as to avoid the overhead.

Table 5.3 shows the knowledge transfer results between devices doing the same task of activity recognition. Model performance is measured by test accuracy. Considering row 1, *Trained-Device* is the accuracy of pre-trained device D_{V1}. *Pure-Transfer* and *Transfer+LimitedTrain* are the accuracy of device D_{A1} using both methods respectively. The last

cell shows the accuracy of audio model trained from scratch using LimitTrainSet. As we can see both methods *Pure-Transfer* and *Transfer+LimitedTrain* achieve better accuracy than training from scratch. This shows that shared latent feature representation is successful in doing knowledge transfer across devices of different modalities. We also observe that *Transfer+LimitedTrain* usually gives the best performance.

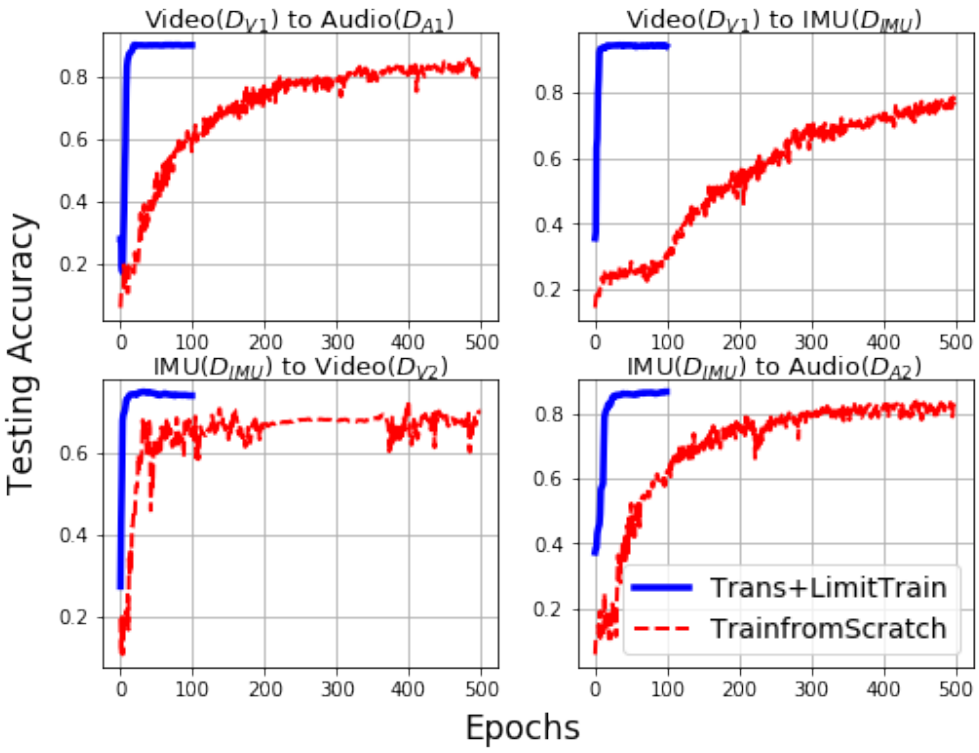

Figure 5.3: *Transfer+LimitedTrain* converges in 10 epochs whereas Training from scratch requires training for around 500 epochs.

In our experiment, we train every model for 10 times to preclude the effect of randomness. Based on the results, significance tests (compared to training from scratch) are carried out using t-test. We find that the *Transfer+LimitedTrain* can outperform training from scratch ($p < 0.005$) in three cases (Video to Audio, Video to IMU, IMU to Audio); and $p < 0.4$ for the case of IMU to Video transfer. This is because video model is complicated and sensitive, and the performance of video model trained from scratch fluctuates.

5.3.4 RecycleML Reduces Training Time

We further compare the effect of number of epochs between *Transfer+LimitedTrain* method and training from scratch using LimitedTrainSet (500 samples). Figure 5.3 shows our results in all the 4 transfers. Clearly, *Transfer+LimitedTrain* method trains model with accuracy greater than 80% in most of the cases with less than 10 epochs, while training from scratch can not achieve comparable accuracy after 500 epochs. This makes RecycleML even more suitable to be deployed on edge devices: it reduces the training time by 50x. The reason for this huge gain is the knowledge transfer using unlabeled data and *Transfer+LimitedTrain* trains only the output layer so it requires very less number of epochs.

5.3.5 RecycleML Reduces Required Labeled Data

To study the effect of number of labeled data samples on model accuracies, we change the size of training data for *Transfer+LimitedTrain* and training from scratch. All the training samples were selected randomly from LimitedTrain set (976 samples) of training dataset. Although methods converge at different speeds (*Transfer+LimitedTrain* converges in 10 epochs, while Training from scratch takes about 500 epochs), in this experiment, we only compare the converged performance of all the models. Figure 5.4 shows our results for four device transfers. Consider Video (D_{V1}) to Audio (D_{A1}), *Transfer+LimitedTrain* is compared with training Audio (D_{A1}) from scratch. Using *Transfer+LimitedTrain*, the model achieve best achievable accuracy using only 50 data samples. While training model from scratch cannot get comparable results even if we increase the size of available data to 976 samples as shown in upper left figure. The testing was performed on entire test dataset. So RecycleML reduces labeled data requirement by at least 90%. However, in ideal scenario, when abundant labeled data samples are available, training from scratch slowly converges and can outperform *Transfer+LimitedTrain*. For IMU (D_{IMU}) to Video (D_{V2}), when more than 750 labeled data are available, training from scratch can outperform the method of *Transfer+LimitedTrain*.

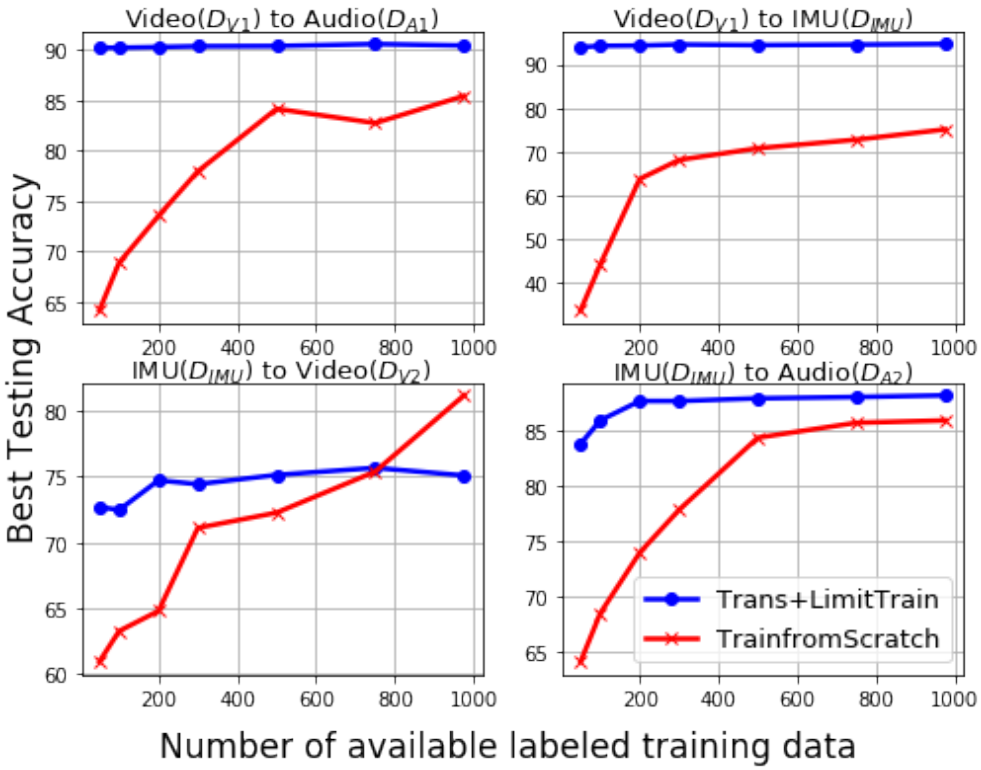

Figure 5.4: With different sizes of labeled data, *Transfer+LimitedTrain* converges better than Training from scratch.

5.3.6 Related Task Transfer Using RecycleML

We tested knowledge transfer from video device to IMU device with video model doing activity recognition task with 7 categories while goal of IMU model is to do activity recognition task with 5 categories in a totally different location.

We did knowledge transfer as described in Section 5.2.2.1 and finally used *Transfer+LimitedTrain* method to train the output layer of IMU model using PersonalTrain set (475 samples). The trained models are tested on PersonalTest set (1117 samples). In Figure 5.5, we plot the learning curve on *Transfer+LimitedTrain* and training from scratch trained using PersonalTrain . When transferring knowledge to a relevant task, RecycleML still learns faster: it converges in 10 epochs and gets a testing accuracy of 91.58%, while training from scratch takes 500 epochs and only gets an accuracy of 61.86%.

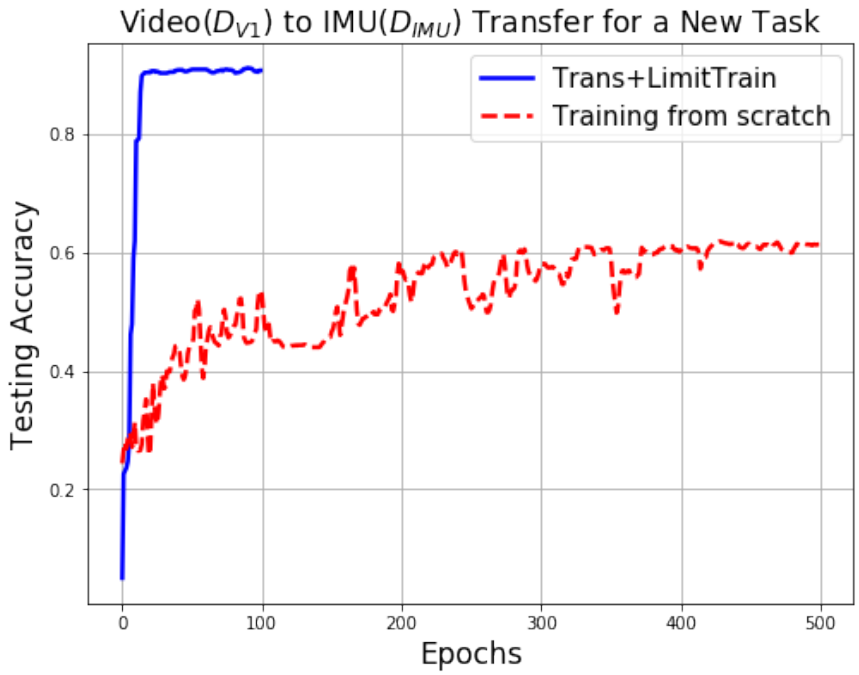

Figure 5.5: Transferring knowledge to a new task: *Transfer+LimitedTrain* learns faster and better than Training from Scratch.

5.4 Related Work

RecycleML is inspired from prior works in machine learning for multimodal data. Previous works [47, 87, 81, 86] combine lower layers from multiple modalities to develop a unified model that outperforms the individual modalities. Radu et al. [87, 86] study combining modalities for human activity recognition on mobile devices. We use the idea of representing multiple modalities in the same latent space in intermediate layers of a deep network, but our focus is on knowledge transfer for machine learning models across multi-modal edge devices.

Ba et al. [8], Hinton et al. [43] present knowledge transfer between the same modality. Ngian et al. [82] use shared representations to improve visual speech classification. Aytar et al. [6] learn shared representations that connect multiple forms of image and text data. Frome et al. [32] show knowledge transfer from text to vision for object classification. Gupta et al. [36] present knowledge transfer between labeled RBG images and unlabeled depth and optical flow images. Aytar et al. [7] show that visual knowledge can be transfer from vision

to sound.

The prior works either focus on image and text data, or take two modalities (vision and audio) from the same source into consideration. In RecycleML, we consider three commonly available sensing modalities on edge devices from multiple sources, and create a unified representation that bridge them. This allows edge devices to use multimodal knowledge transfer across different sensing modalities of ambient sensors (vision and audio) and wearables sensors (IMU) for the first time.

5.5 Discussion

While RecycleML shows promise in terms of handling both paucity of labeled data and also speeds up model training across multiple modalities, the ability of the approach to generalize to different applications for larger datasets needs further investigation. Furthermore, our experiments indicate that while the trained models can be personalized to a specific environment, they need regularization to generalize to new settings.

For cross modal knowledge transfer using RecycleML, we need unlabeled but synchronized data. In our experiments, since audio and video data are captured by the same device, they are naturally synchronized. In addition, we used the default smartphone timestamps, synchronized through the Network Time Protocol (NTP) [80] service, to synchronize IMU device with video and sound device. In real settings, however, edge devices have to be time synchronized in order to observe the same event at the same time.

In our experiments, we chose the fully connected layer (immediately prior to the output layer) as the common latent space. In future, we plan to explore different choices for the shared representation layer, for efficient sensory substitution and task transfer on edge devices.

5.6 Conclusion

Heterogeneity in sensing modality of the edge devices, together with lack of labeled training data, represent two of the most significant barriers to enabling the learning-on-the-edge paradigm. Towards this end, we presented RecycleML, a system that enables multi-modality edge devices to perform knowledge transfer between their models by mapping their lower layers to a shared latent space representation. RecycleML further allows task-specific transfer between models by targeted retraining of the higher layers beyond the shared latent space – reducing the amount of labeled data needed for model training. Our initial experiments, performed using multi-modality data (vision, audio, IMU) for activity recognition, show that transfer model trained using RecycleML leads to reduced training time and results in increased accuracy compared to an edge model trained from scratch using limited labeled data.

CHAPTER 6

Concluding Remarks

6.1 Conclusion

In this dissertation, we investigate the deployment of deep learning systems for IoT devices in complex and dynamic environments. More specifically, we focus on addressing complex tasks using a neural-symbolic framework with the help of prior knowledge, achieving flexible inferencing at the run-time while reducing retraining effort, and adapting deep learning models to dynamic IoT settings by cross-modal knowledge transfer.

For the complex event detection, we design and implement the DEEPCEP framework, which combines the logical reasoning ability of CEP and the inference power of Deep learning models to detect complex events for unstructured, distributed multimodal data. We use the scenario of detecting an unattended bag in a sensitive area to illustrate how DEEPCEP can be instrumented for an application. In particular, we highlighted how DEEPCEP provides a framework to propagate the uncertainty from a detected simple event to a composed complex event using ProbLog. In addition, we presented Neuroplex, a neural-symbolic framework for detecting complex events. Using semantic knowledge to guide the learning, Neuroplex can learn to detect complex events with much fewer and sparse annotations. Results on different datasets prove the effectiveness, reliability, and robustness of Neuroplex.

To enable flexible inferencing and understand opaque data better, we introduce DeepSQA, a generalized framework for sensory question answering. By taking both sensory data and natural language questions simultaneously, DeepSQA can identify the tasks specified by questions and perform reasoning on sensory data accordingly. It reduces the laborious work of training new deep learning models when new tasks are introduced, and improves the in-

ferencing flexibility. To evaluate SQA models, we introduce SQA-Gen, which automatically generates SQA datasets using labeled source sensory data. Based on this tool, we propose the OppQA dataset for benchmarking SQA model performance. Results on OppQA confirm that DeepSQA is effective and robust in sensory QA tasks.

By proposing RecycleML, we provide a unified framework for knowledge transfer of deep learning models. The introduced sensory substitution learning allows knowledge transfer even when the input modality is changing. RecycleML also allows task-specific transfer between models by retraining the higher layers beyond the shared latent space, which effectively reduces the amount of labeled data needed for model training. This framework increases the adaptability of deep learning models of IoT devices in dynamic environments. The evaluation result on the cross-modal activity dataset demonstrates the performance of RecycleML in both learning speed and data efficiency.

We release our datasets and codes on public GitHub repositories for all the work above to help facilitate future research in this area.

Bibliography

[1] Antlr.

[2] Jerome Abdelnour, Giampiero Salvi, and Jean Rouat. Clear: A dataset for compositional language and elementary acoustic reasoning. *arXiv preprint arXiv:1811.10561*, 2018.

[3] Peter Anderson, Xiaodong He, Chris Buehler, Damien Teney, Mark Johnson, Stephen Gould, and Lei Zhang. Bottom-up and top-down attention for image captioning and visual question answering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6077–6086, 2018.

[4] Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. Vqa: Visual question answering. In *Proceedings of the IEEE international conference on computer vision*, pages 2425–2433, 2015.

[5] Asra Aslam and Edward Curry. Towards a generalized approach for deep neural network based event processing for the internet of multimedia things. *IEEE Access*, 6:25573–25587, 2018.

[6] Yusuf Aytar, Lluis Castrejon, Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. Cross-modal scene networks. *IEEE transactions on pattern analysis and machine intelligence*, 2017.

[7] Yusuf Aytar, Carl Vondrick, and Antonio Torralba. Soundnet: Learning sound representations from unlabeled video. In *Advances in Neural Information Processing Systems*, pages 892–900, 2016.

[8] Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *Advances in neural information processing systems*, pages 2654–2662, 2014.

[9] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[10] Pouya Bashivan, Irina Rish, Mohammed Yeasin, and Noel Codella. Learning representations from eeg with deep recurrent-convolutional neural networks. *arXiv preprint arXiv:1511.06448*, 2015.

[11] Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1533–1544, 2013.

[12] Pedro Bizarro. Bicep-benchmarking complex event processing systems. In *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2007.

[13] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[14] Ella Browning, Mark Bolton, Ellie Owen, Akiko Shoji, Tim Guilford, and Robin Freeman. Predicting animal behaviour using deep learning: Gps data alone accurately predict diving in seabirds. *Methods in Ecology and Evolution*, 9(3):681–692, 2018.

[15] Diane Cook, Kyle D Feuz, and Narayanan C Krishnan. Transfer learning for activity recognition: A survey. *Knowledge and information systems*, 36(3):537–556, 2013.

[16] Luc De Raedt and Angelika Kimmig. Probabilistic (logic) programming concepts. *Machine Learning*, 100(1):5–47, 2015.

[17] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. Problog: A probabilistic prolog and its application in link discovery. In *IJCAI*, volume 7, pages 2462–2467. Hyderabad, 2007.

[18] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. ProbLog: A probabilistic Prolog and its application in link discovery. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 2462–2467, 2007.

[19] Gert Dekkers, Steven Lauwereins, Bart Thoen, Mulu Weldegebreal Adhana, Henk Brouckxon, Toon van Waterschoot, Bart Vanrumste, Marian Verhelst, and Peter Karsmakers. The SINS database for detection of daily activities in a home environment using an acoustic sensor network. In *Proceedings of the Detection and Classification of Acoustic Scenes and Events 2017 Workshop (DCASE2017)*, pages 32–36, November 2017.

[20] Gert Dekkers, Lode Vuegen, Toon van Waterschoot, Bart Vanrumste, and Peter Karsmakers. Dcase 2018 challenge-task 5: Monitoring of domestic activities based on multi-channel acoustics. *arXiv preprint arXiv:1807.11246*, 2018.

[21] Thomas Demeester, Tim Rocktäschel, and Sebastian Riedel. Lifted rule injection for relation embeddings. *arXiv preprint arXiv:1606.08359*, 2016.

[22] Alan J Demers, Johannes Gehrke, Biswanath Panda, Mirek Riedewald, Varun Sharma, Walker M White, et al. Cayuga: A general purpose event monitoring system. In *Cidr*, volume 7, pages 412–422, 2007.

[23] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[24] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[25] Ivan Donadello, Luciano Serafini, and Artur D'Avila Garcez. Logic tensor networks for semantic image interpretation. *arXiv preprint arXiv:1705.08968*, 2017.

[26] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *International conference on machine learning*, pages 647–655, 2014.

[27] Dan Ellis. Chroma feature analysis and synthesis. *Resources of Laboratory for the Recognition and Organization of Speech and Audio-LabROSA*, 2007.

[28] Miquel Espi, Masakiyo Fujimoto, Keisuke Kinoshita, and Tomohiro Nakatani. Exploiting spectro-temporal locality in deep learning based acoustic event detection. *EURASIP Journal on Audio, Speech, and Music Processing*, 2015(1):26, 2015.

[29] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, 33(4):917–963, 2019.

[30] Daan Fierens, Guy Van den Broeck, Joris Renkens, Dimitar Shterionov, Bernd Gutmann, Ingo Thon, Gerda Janssens, and Luc De Raedt. Inference and learning in probabilistic logic programs using weighted Boolean formulas. *Theory and Practice of Logic Programming*, 15(03):358–401, May 2015.

[31] Ioannis Flouris, Nikos Giatrakos, Antonios Deligiannakis, Minos Garofalakis, Michael Kamp, and Michael Mock. Issues in complex event processing: Status and prospects in the big data era. *Journal of Systems and Software*, 127:217–236, 2017.

[32] Andrea Frome, Greg Corrado, Jonathon Shlens, Samy Bengio, Jeffrey Dean, Marc'Aurelio Ranzato, and Tomas Mikolov. Devise: A deep visual-semantic embedding model. In *Neural Information Processing Systems (NIPS)*, 2013.

[33] Lajos Jenő Fülöp, Gabriella Tóth, Róbert Rácz, János Pánczél, Tamás Gergely, Arpád Beszédes, and Lóránt Farkas. Survey on complex event processing and predictive analytics. In *Proceedings of the Fifth Balkan Conference in Informatics*, pages 26–31. Citeseer, 2010.

[34] Chuang Gan, Naiyan Wang, Yi Yang, Dit-Yan Yeung, and Alex G Hauptmann. Devnet: A deep event network for multimedia event detection and evidence recounting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2568–2577, 2015.

[35] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE, 2013.

[36] Saurabh Gupta, Judy Hoffman, and Jitendra Malik. Cross modal distillation for supervision transfer. In *Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*, pages 2827–2836. IEEE, 2016.

[37] Daniel Gyllstrom, Eugene Wu, Hee-Jin Chae, Yanlei Diao, Patrick Stahlberg, and Gordon Anderson. Sase: Complex event processing over streams. *arXiv preprint cs/0612128*, 2006.

[38] Nils Y Hammerla, Shane Halloran, and Thomas Plötz. Deep, convolutional, and recurrent models for human activity recognition using wearables. *arXiv preprint arXiv:1604.08880*, 2016.

[39] Md Nazmul Haque, Mahir Mahbub, Md Hasan Tarek, Lutfun Nahar Lota, and Amin Ahsan Ali. Nurse care activity recognition: A gru-based approach with attention mechanism. In *Adjunct Proceedings of the 2019 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2019 ACM International Symposium on Wearable Computers*, pages 719–723, 2019.

[40] Christopher Harte, Mark Sandler, and Martin Gasser. Detecting harmonic change in musical audio. In *Proceedings of the 1st ACM workshop on Audio and music computing multimedia*, pages 21–26. ACM, 2006.

[41] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[42] Jonathan Herzig, Paweł Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Martin Eisenschlos. Tapas: Weakly supervised table parsing via pre-training. *arXiv preprint arXiv:2004.02349*, 2020.

[43] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[44] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[45] Ronghang Hu, Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Kate Saenko. Learning to reason: End-to-end module networks for visual question answering. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 804–813, 2017.

[46] Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard Hovy, and Eric Xing. Harnessing deep neural networks with logic rules. *arXiv preprint arXiv:1603.06318*, 2016.

[47] Jing Huang and Brian Kingsbury. Audio-visual deep learning for noise robust speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 7596–7599. IEEE, 2013.

[48] Drew A Hudson and Christopher D Manning. Compositional attention networks for machine reasoning. *arXiv preprint arXiv:1803.03067*, 2018.

[49] Drew A Hudson and Christopher D Manning. Gqa: A new dataset for real-world visual reasoning and compositional question answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6700–6709, 2019.

[50] Yunseok Jang, Yale Song, Youngjae Yu, Youngjin Kim, and Gunhee Kim. Tgif-qa: Toward spatio-temporal reasoning in visual question answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2758–2766, 2017.

[51] I-Hong Jhuo and DT Lee. Video event detection via multi-modality deep learning. In *2014 22nd International Conference on Pattern Recognition*, pages 666–671. IEEE, 2014.

[52] Dan-Ning Jiang, Lie Lu, Hong-Jiang Zhang, Jian-Hua Tao, and Lian-Hong Cai. Music type classification by spectral contrast feature. In *Multimedia and Expo, 2002. ICME'02. Proceedings. 2002 IEEE International Conference on*, volume 1, pages 113–116. IEEE, 2002.

[53] Lu Jiang, Junwei Liang, Liangliang Cao, Yannis Kalantidis, Sachin Farfade, and Alexander Hauptmann. Memexqa: Visual memex question answering. *arXiv preprint arXiv:1708.01336*, 2017.

[54] Yu-Gang Jiang, Zuxuan Wu, Jun Wang, Xiangyang Xue, and Shih-Fu Chang. Exploiting feature and class relationships in video categorization with regularized deep neural networks. *IEEE transactions on pattern analysis and machine intelligence*, 40(2):352–364, 2018.

[55] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2901–2910, 2017.

[56] Justin Johnson, Bharath Hariharan, Laurens Van Der Maaten, Judy Hoffman, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Inferring and executing programs for visual reasoning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2989–2998, 2017.

[57] Kushal Kafle, Brian Price, Scott Cohen, and Christopher Kanan. Dvqa: Understanding data visualizations via question answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5648–5656, 2018.

[58] Kushal Kafle, Robik Shrestha, Scott Cohen, Brian Price, and Christopher Kanan. Answering questions about data visualizations using efficient bimodal fusion. In *The IEEE Winter Conference on Applications of Computer Vision*, pages 1498–1507, 2020.

[59] Vahid Kazemi and Ali Elqursh. Show, ask, attend, and answer: A strong baseline for visual question answering. *arXiv preprint arXiv:1704.03162*, 2017.

[60] Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. Character-aware neural language models. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[61] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[62] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[63] Paula Lago, Sayeda Shamma Alia, Shingo Takeda, Tittaya Mairittha, Nattaya Mairittha, Farina Faiz, Yusuke Nishimura, Kohei Adachi, Tsuyoshi Okita, François Charpillet, et al. Nurse care activity recognition challenge: summary and results. In *Adjunct Proceedings of the 2019 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2019 ACM International Symposium on Wearable Computers*, pages 746–751, 2019.

[64] An Ngoc Lam and Øystein Haugen. Complex event processing in thingml. In *International Conference on System Analysis and Modeling*, pages 20–35. Springer, 2016.

[65] Martin Längkvist, Lars Karlsson, and Amy Loutfi. A review of unsupervised feature learning and deep learning for time-series modeling. *Pattern Recognition Letters*, 42:11–24, 2014.

[66] Oscar D Lara and Miguel A Labrador. A survey on human activity recognition using wearable sensors. *IEEE communications surveys & tutorials*, 15(3):1192–1209, 2012.

[67] Phong Le and Willem Zuidema. Compositional distributional semantics with long short term memory. *arXiv preprint arXiv:1503.02510*, 2015.

[68] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[69] Jie Lei, Licheng Yu, Mohit Bansal, and Tamara L Berg. Tvqa: Localized, compositional video question answering. *arXiv preprint arXiv:1809.01696*, 2018.

[70] Yitao Liang and Guy Van den Broeck. Learning logistic circuits. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4277–4286, 2019.

[71] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[72] Kun Liu, Wu Liu, Chuang Gan, Mingkui Tan, and Huadong Ma. T-c3d: temporal convolutional 3d network for real-time action recognition. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

[73] Xiaochen Liu, Pradipta Ghosh, Ulutan Oytun, B.S. Manjunath, Kevin Chan, and Ramesh Govindan. Caesar: Cross-camera complex activity recognition. In *17th ACM Conference on Embedded Networked Sensor Systems, SENSYS 2019*. Association for Computing Machinery, Inc, 2019.

[74] Beth Logan et al. Mel frequency cepstral coefficients for music modeling. In *ISMIR*, volume 270, pages 1–11, 2000.

[75] Dimitrios Lymberopoulos, Abhijit S Ogale, Andreas Savvides, and Yiannis Aloimonos. A sensory grammar for inferring behaviors in sensor networks. In *Proceedings of the 5th international conference on Information processing in sensor networks*, pages 251–259, 2006.

[76] Meiyi Ma, Ji Gao, Lu Feng, and John Stankovic. Stlnet: Signal temporal logic enforced multivariate recurrent neural networks. *Advances in Neural Information Processing Systems*, 33, 2020.

[77] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. In *Advances in Neural Information Processing Systems*, pages 3749–3759, 2018.

[78] Mohsen Marjani, Fariza Nasaruddin, Abdullah Gani, Ahmad Karim, Ibrahim Abaker Targio Hashem, Aisha Siddiqa, and Ibrar Yaqoob. Big iot data analytics: architecture, opportunities, and open research challenges. *IEEE Access*, 5:5247–5261, 2017.

[79] Brian McFee, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. librosa: Audio and music signal analysis in python. In *Proceedings of the 14th python in science conference*, pages 18–25, 2015.

[80] David L Mills. Internet time synchronization: the network time protocol. *IEEE Transactions on communications*, 39(10):1482–1493, 1991.

[81] Sebastian Münzner, Philip Schmidt, Attila Reiss, Michael Hanselmann, Rainer Stiefelhagen, and Robert Dürichen. Cnn-based sensor fusion techniques for multimodal human activity recognition. In *Proceedings of the 2017 ACM International Symposium on Wearable Computers*, pages 158–165. ACM, 2017.

[82] Jiquan Ngiam, Aditya Khosla, Mingyu Kim, Juhan Nam, Honglak Lee, and Andrew Y Ng. Multimodal deep learning. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 689–696, 2011.

[83] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[84] Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. *arXiv preprint arXiv:1709.07871*, 2017.

[85] Siyuan Qi, Siyuan Huang, Ping Wei, and Song-Chun Zhu. Predicting human activities using stochastic grammar. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1164–1172, 2017.

[86] Valentin Radu, Nicholas D Lane, Sourav Bhattacharya, Cecilia Mascolo, Mahesh K Marina, and Fahim Kawsar. Towards multimodal deep learning for activity recognition on mobile devices. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct*, pages 185–188. ACM, 2016.

[87] Valentin Radu, Catherine Tong, Sourav Bhattacharya, Nicholas D Lane, Cecilia Mascolo, Mahesh K Marina, and Fahim Kawsar. Multimodal deep learning for activity and context recognition. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 1(4):1–27, 2018.

[88] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

[89] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[90] D Robins. Complex event processing. In *Second International Workshop on Education Technology and Computer Science. Wuhan*, pages 1–10. Citeseer, 2010.

[91] Tim Rocktäschel, Sameer Singh, and Sebastian Riedel. Injecting logical background knowledge into embeddings for relation extraction. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1119–1129, 2015.

[92] Mohammad Sabokrou, Mohsen Fayyaz, Mahmood Fathy, and Reinhard Klette. Deep-cascade: Cascading 3d deep neural networks for fast anomaly detection and localization in crowded scenes. *IEEE Transactions on Image Processing*, 26(4):1992–2004, 2017.

[93] Hesam Sagha, Sundara Tejaswi Digumarti, José del R Millán, Ricardo Chavarriaga, Alberto Calatroni, Daniel Roggen, and Gerhard Tröster. Benchmarking classification techniques using the opportunity human activity dataset. In *2011 IEEE International Conference on Systems, Man, and Cybernetics*, pages 36–40. IEEE, 2011.

[94] Taisuke Sato. A statistical learning method for logic programs with distribution semantics. In *Proceedings of the 12th International Conference on Logic Programming (ICLP-95)*, 1995.

[95] Murat Sensoy, Lance Kaplan, and Melih Kandemir. Evidential deep learning to quantify classification uncertainty. In *Advances in Neural Information Processing Systems*, pages 3183–3193, 2018.

[96] Chenguang Shen, Bo-Jhang Ho, and Mani Srivastava. Milift: Efficient smartwatch-based workout tracking using automatic segmentation. *IEEE Transactions on Mobile Computing*, 2017.

[97] Baoguang Shi, Xiang Bai, and Cong Yao. An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition. *IEEE transactions on pattern analysis and machine intelligence*, 39(11):2298–2304, 2016.

[98] Muhammad Shoaib, Stephan Bosch, Ozlem Durmaz Incel, Hans Scholten, and Paul JM Havinga. Complex human activity recognition using smartphone and wrist-worn motion sensors. *Sensors*, 16(4):426, 2016.

[99] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[100] Anastasios Skarlatidis, Alexander Artikis, Jason Filippou, and Georgios Paliouras. A probabilistic logic programming event calculus. *Theory and Practice of Logic Programming*, 15(2):213–245, 2015.

[101] Richard Socher, Milind Ganjoo, Christopher D Manning, and Andrew Ng. Zero-shot learning through cross-modal transfer. In *Advances in neural information processing systems*, pages 935–943, 2013.

[102] Russell Stewart and Stefano Ermon. Label-free supervision of neural networks with physics and domain knowledge. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[103] Sriskandarajah Suhothayan, Kasun Gajasinghe, Isuru Loku Narangoda, Subash Chaturanga, Srinath Perera, and Vishaka Nanayakkara. Siddhi: A second look at complex event processing architectures. In *Proceedings of the 2011 ACM workshop on Gateway computing environments*, pages 43–50. ACM, 2011.

[104] Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*, 2015.

[105] Mohammed Yassine Kazi Tani, Adel Lablack, Abdelghani Ghomari, and Ioan Marius Bilasco. Events detection using a video-surveillance ontology and a rule-based approach. In *European Conference on Computer Vision*, pages 299–308. Springer, 2014.

[106] Makarand Tapaswi, Yukun Zhu, Rainer Stiefelhagen, Antonio Torralba, Raquel Urtasun, and Sanja Fidler. Movieqa: Understanding stories in movies through question-answering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4631–4640, 2016.

[107] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *Computer Vision (ICCV), 2015 IEEE International Conference on*, pages 4489–4497. IEEE, 2015.

[108] Yonatan Vaizman, Katherine Ellis, Gert Lanckriet, and Nadir Weibel. Extrasensory app: Data collection in-the-wild with rich user interface to self-report behavior. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2018.

[109] Marc Roig Vilamala, Liam Hiley, Yulia Hicks, Alun Preece, and Federico Cerutti. A pilot study on detecting violence in videos fusing proxy models. In *Fusion*, 2019.

[110] Marc Roig Vilamala, Harrison Taylor, Tianwei Xing, Luis Garcia, Mani Srivastava, Lance Kaplan, Alun Preece, Angelika Kimming, and Federico Cerutti. A hybrid neuro-symbolic approach for complex event processing (extended abstract). In *EPTCS proceedings of ICLP*, 2020.

[111] Jindong Wang, Yiqiang Chen, Shuji Hao, Xiaohui Peng, and Lisha Hu. Deep learning for sensor-based activity recognition: A survey. *Pattern Recognition Letters*, 119:3–11, 2019.

[112] Po-Wei Wang, Priya L Donti, Bryan Wilder, and Zico Kolter. Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. *arXiv preprint arXiv:1905.12149*, 2019.

[113] Wei Wang, Alex X Liu, Muhammad Shahzad, Kang Ling, and Sanglu Lu. Device-free human activity recognition using commercial wifi devices. *IEEE Journal on Selected Areas in Communications*, 35(5):1118–1131, 2017.

[114] Xiao-Wei Wang, Dan Nie, and Bao-Liang Lu. Emotional state classification from eeg data using machine learning approach. *Neurocomputing*, 129:94–106, 2014.

[115] Eugene Wu, Yanlei Diao, and Shariq Rizvi. High-performance complex event processing over streams. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 407–418. ACM, 2006.

[116] Zuxuan Wu, Yu-Gang Jiang, Xi Wang, Hao Ye, and Xiangyang Xue. Multi-stream multi-class fusion of deep networks for video classification. In *Proceedings of the 24th ACM international conference on Multimedia*, pages 791–800. ACM, 2016.

[117] Zuxuan Wu, Xi Wang, Yu-Gang Jiang, Hao Ye, and Xiangyang Xue. Modeling spatial-temporal clues in a hybrid deep learning framework for video classification. In *Proceedings of the 23rd ACM international conference on Multimedia*, pages 461–470. ACM, 2015.

[118] Yaqi Xie, Ziwei Xu, Kuldeep Meel, Mohan S Kankanhalli, and Harold Soh. Semantically-regularized logic graph embeddings. *arXiv preprint arXiv:1909.01161*, 2019.

[119] Tianwei Xing, Sandeep Singh Sandha, Bharathan Balaji, Supriyo Chakraborty, and Mani Srivastava. Enabling edge devices that learn from each other: Cross modal training for activity recognition. In *Proceedings of the 1st International Workshop on Edge Systems, Analytics and Networking*, pages 37–42. ACM, 2018.

[120] Tianwei Xing, Marc Roig Vilamala, Luis Garcia, Federico Cerutti, Lance Kaplan, Alun Preece, and Mani Srivastava. Deepcep: Deep complex event processing using distributed multimodal information. In *2019 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 87–92. IEEE, 2019.

[121] Yuanjun Xiong, Kai Zhu, Dahua Lin, and Xiaoou Tang. Recognize complex events from static images by fusing deep channels. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1600–1609, 2015.

[122] Dan Xu, Elisa Ricci, Yan Yan, Jingkuan Song, and Nicu Sebe. Learning deep representations of appearance and motion for anomalous event detection. *arXiv preprint arXiv:1510.01553*, 2015.

[123] Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Van den Broeck. A semantic loss function for deep learning with symbolic knowledge. *arXiv preprint arXiv:1711.11157*, 2017.

[124] Yan Yan, Yi Yang, Deyu Meng, Gaowen Liu, Wei Tong, Alexander G Hauptmann, and Nicu Sebe. Event oriented dictionary learning for complex event detection. *IEEE Transactions on Image Processing*, 24(6):1867–1878, 2015.

[125] Jianbo Yang, Minh Nhut Nguyen, Phyo Phyo San, Xiao Li Li, and Shonali Krishnaswamy. Deep convolutional neural networks on multichannel time series for human activity recognition. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[126] Zichao Yang, Xiaodong He, Jianfeng Gao, Li Deng, and Alex Smola. Stacked attention networks for image question answering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 21–29, 2016.

[127] Wen Yao, Chao-Hsien Chu, and Zang Li. Leveraging complex event processing for smart hospitals using rfid. *Journal of Network and Computer Applications*, 34(3):799–810, 2011.

[128] Kexin Yi, Jiajun Wu, Chuang Gan, Antonio Torralba, Pushmeet Kohli, and Josh Tenenbaum. Neural-symbolic vqa: Disentangling reasoning from vision and language understanding. In *Advances in neural information processing systems*, pages 1031–1042, 2018.

[129] Xishan Zhang, Hanwang Zhang, Yongdong Zhang, Yang Yang, Meng Wang, Huanbo Luan, Jintao Li, and Tat-Seng Chua. Deep fusion of multiple semantic cues for complex event recognition. *IEEE Transactions on Image Processing*, 25(3):1033–1046, 2016.

[130] Yu Zheng, Like Liu, Longhao Wang, and Xing Xie. Learning transportation mode from raw gps data for geographic applications on the web. In *Proceedings of the 17th international conference on World Wide Web*, pages 247–256, 2008.

[131] Xiaodan Zhu, Parinaz Sobihani, and Hongyu Guo. Long short-term memory over recursive structures. In *International Conference on Machine Learning*, pages 1604–1612, 2015.

[132] Michael Zoumboulakis and George Roussos. Escalation: Complex event detection in wireless sensor networks. In *European Conference on Smart Sensing and Context*, pages 270–285. Springer, 2007.

[133] Michael Zoumboulakis and George Roussos. Complex event detection in extremely resource-constrained wireless sensor networks. *Mobile Networks and Applications*, 16(2):194–213, 2011.