UNIVERSITY OF CALIFORNIA

Los Angeles

An Empirical Evaluation of Neural and Neuro-symbolic Approaches on Multimodal Complex Event Detection

A thesis submitted in partial satisfaction
of the requirements for the degree
Master of Science in Electrical and Computer Engineering

by

Liying Han

2023

ABSTRACT OF THE THESIS

An Empirical Evaluation of Neural and Neuro-symbolic
Approaches on Multimodal Complex Event Detection

by

Liying Han
Master of Science in Electrical and Computer Engineering
University of California, Los Angeles, 2023
Professor Mani B. Srivastava, Chair

Over the past decade, deep learning models have gained significant popularity for processing noisy sensor inputs in various real-world scenarios. However, they still face challenges such as data inefficiency, interpretability limitations, and weak reasoning abilities. On the other hand, Symbolic algorithms offer high-level reasoning and interpretability but struggle with uncertainties and high-dimensional sensor data. Neuro-symbolic approaches have emerged as promising solution, combining the strengths of deep learning and symbolic methods. They show potential for Complex Event Processing (CEP) in critical domains like healthcare monitoring and smart city control. This work explores the performance of neuro-symbolic methods in complex event detection tasks from multimodal sensor data, which involves recognizing complex events with complicated temporal patterns that span a wide time range. However, the lack of multimodal and complex event datasets poses a challenge. To address this, we formulate the complex event detection problem, synthesize a multimodal activity dataset, and build a stochastic daily human activity simulator to generate multimodal complex event datasets. We design a two-module system that includes a multimodal fusion module to handle sample inconsistencies and utilize information across modalities, and a backbone module that can be replaced with neural-only and neuro-symbolic models for performance comparison. The evaluation results demonstrate the superiority of the neuro-symbolic approach in complex event detection and provide insights for developing improved neuro-symbolic architectures for effective and applicable complex event detection systems in multimodal settings.

The thesis of Liying Han is approved.

Guy Van den Broeck

Jonathan Chau-Yan Kao

Mani B. Srivastava, Committee Chair

University of California, Los Angeles

2023

To my family . . .

who continue to inspire me in my pursuit of excellence

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

In this new era of information explosion, we now have unprecedented access to vast streams of data generated from diverse modalities, such as text, audio, images, videos, and more. This availability of diverse and extensive data has greatly accelerated the progress of data-driven machine-learning techniques. With the aid of powerful computing resources, the field of machine learning has witnessed a remarkable surge, particularly in the realm of neural network-based approaches. These approaches have demonstrated their prowess in processing, analyzing, and comprehending complex data, as exemplified by the success of large language models that exhibit human-like understanding and communication abilities.

However, the long-term reasoning and memorization capabilities of neural networks remain a critical concern, especially when it comes to enabling machine learning models to handle complex events processing in various industries, ranging from healthcare and finance to manufacturing and entertainment. Due to the inherent structure of neural networks, they have often been criticized for their lack of interpretability. Consequently, researchers have proposed neuro-symbolic approaches to combine neural networks with logical reasoning abilities and safety guarantees. In this paper, our objectives are to investigate and analyze the capabilities of both neural-only and neuro-symbolic approaches in the context of multimodal complex event processing.

1.1 Complex Event Processing

To begin with, let us explore the concept of complex event processing (CEP) and its importance. CEP involves the computational analysis of data streams associated with complex events, which are composed of simpler atomic events following temporal patterns. Building a CEP system requires addressing three key challenges.

Real-time. In today's fast-paced and data-rich world, organizations across various domains face the challenge of extracting meaningful insights and taking swift actions in response to complex events occurring within their operational environments. CEP offers a powerful solution by enabling the identification, correlation, and analysis of events, allowing for the detection of significant patterns, trends, and relationships in real time. For example, in finance, real-time CEP systems can detect complex events such as anomalies in stock prices, sudden market fluctuations, or fraudulent trading patterns. This enables financial institutions to make informed decisions, trigger automated actions, and mitigate risks promptly. In healthcare systems, where real-time monitoring and analysis of patient data are vital, CEP can monitor vital signs, laboratory results, and clinical notes to identify critical events like patient deterioration. It can alert healthcare providers, trigger interventions, or escalate urgent cases to ensure timely and appropriate care. By harnessing the real-time capabilities of CEP, we can gain valuable situational awareness, enhance decision-making processes, and facilitate timely responses to critical events.

Multimodal sensor data. Sensor networks, driven by the proliferation of connected devices and the Internet of Things (IoT), generate vast streams of data from sensors embedded in various environments and present a significant application area for CEP. The sensors employed capture a wide range of information from the physical surroundings, including temperature, humidity, motion, sound, videos, and more, thereby providing rich and real-time insights into the world around us. By integrating sensor data from diverse modalities with CEP, we can acquire a comprehensive understanding of the environment, enabling a robust and effective approach to detect patterns and anomalies.

Complex patterns. One key property shared by different types of complex events is their reliance on complex patterns that span over time. Atomic events serve as the foundational building blocks for higher-level events. An atomic event is considered as the low-level building block to build higher-level events. For example, in a smart city setting, we can define the complex event "traffic jam formation" using the atomic event "vehicle speed drops below 20 mph for 10 consecutive seconds," and specify the temporal pattern as "five or more instances of the atomic event occurring within a 3-minute interval." The complex event is triggered when the atomic events align with the specified temporal pattern, indicating the emergence of significant traffic congestion. Therefore, the ability to reason about temporal relations is a crucial component of CEP systems.

The significance of Complex Event Processing (CEP) in today's data-driven era cannot be overstated. CEP finds applications in diverse fields such as finance, healthcare, environmental monitoring, smart city, and more. These applications, however, come with their challenges, but they hold the promise of advancing and revolutionizing various industries. By effectively addressing the challenges of real-time processing, integrating multimodal sensor data, and reasoning about complex temporal patterns, CEP can enable us to harness valuable insights, enhance decision-making processes, and drive innovation in the data-driven era.

1.2 Neural-only Approach

When it comes to pattern matching, neural networks are commonly employed as a solution. Neural networks designed for processing variable-length sequential inputs are a suitable choice for Complex Event Processing (CEP), which involves data streams as input.

Recurrent Neural Networks (RNNs) are a class of artificial neural networks specifically designed to handle input sequences of varying lengths. RNNs and their variants, such as

Long Short-Term Memory (LSTM) [HS97] and Gated Recurrent Units (GRU) [CMG14], have found applications in diverse fields including time series forecasting, machine translation, and handwriting recognition. These models utilize internal states to store information over time. However, they have limitations in terms of sequential computation and capturing long-term dependencies in practice.

Another class of neural networks for sequential data processing is Temporal Convolutional Networks (TCNs) [BKK18]. TCNs can be viewed as a variant of Convolutional Neural Networks (CNNs) that are tailored to process data with a temporal dimension. While traditional CNNs are primarily used for image analysis, TCNs extend the concept of convolutional operations to capture dependencies and patterns in sequential data. In contrast to RNNs that process temporal inputs sequentially, TCNs are effective at parallelizing computations.

Transformers [VSP17] emerged as a powerful architecture for sequence modeling tasks and have revolutionized the field of natural language processing (NLP). Unlike traditional RNNs and CNNs that rely on sequential or convolutional operations, transformers employ a self-attention mechanism to capture relationships between different elements in a sequence. This self-attention mechanism allows transformers to attend to the entire input sequence simultaneously, capturing global dependencies and interactions between elements. Both TCNs and transformers possess the ability to parallelize computations when processing data across multiple time steps.

Neural network models offer great promise in handling uncertainty and noise present in real-life sensor data, making them suitable for handling complex events in CEP. However, the research on their capabilities to reason over long sequences and process complex event patterns is still an ongoing area of study. Further exploration and investigation are required to fully understand their potential in complex event processing.

1.3 Neuro-symbolic Approach

Complex Event Processing (CEP) deals with sensor data that is often noisy, contains outliers, and may have missing values. As we mentioned before, deep learning models excel in handling such data by being inherently robust to noise and capable of handling uncertainty. However, a drawback of deep learning models is their limited interpretability and logical reasoning abilities. On the other hand, symbolic reasoning approaches are known for their ease of understanding and high-level reasoning on symbols, but they struggle with uncertainties and are not well-suited for handling high-dimensional sensor data.

To address these limitations, neuro-symbolic approaches provide a compelling solution by integrating the strengths of deep learning and symbolic reasoning. These approaches combine the efficient processing capabilities of deep learning with the interpretability and logical reasoning of symbolic methods. By leveraging deep learning to handle noisy sensor data and incorporating symbolic reasoning to provide transparent and interpretable explanations for complex events, neuro-symbolic approaches become an appealing choice for CEP applications.

There is a diverse landscape in the field of neuro-symbolic methods. Building upon the taxonomy proposed by Henry Kautz[Kau22], neuro-symbolic systems can be classified into the following six types:

Symbolic-after-Neural. In this paradigm, neural components are used to process raw inputs into structured data, which are then fed to symbolic programs for further reasoning. This is the most common and general neuro-symbolic architecture, so most works may fall into this paradigm. An example is DUA [MTC22], which is a hierarchical reinforcement learning (RL) framework. It uses neural components that are deep RL agents pre-trained on primitive actions, and a symbolic meta-policy learning module provided with common sense background knowledge will learn the optimal combination of those primitive actions to complete AI tasks. Another example that enables end-to-end training of neural components is

DeepProbLog [MDK18, MDK19]. In DeepProbLog, the inputs are passed to neural network modules for classification tasks, whose outputs are probabilistic predicates used for a logic program together with probabilistic facts to evaluate user-defined logic rules. The neural components are trained using labels only given at the output side of the logic program.

Neural-after-Symbolic. This paradigm is the transposition of the previous one. Systems of this type perform deep learning over inputs processed by symbolic reasoning. The neural component either learns the relations between the symbols or learns to focus on some specific symbols based on needs. One typical example is the Graph Neural Network [SGT09], where graph nodes are encoded or pre-processed before performing inference via deep learning methods.

Neural-Symbolic aggregation. A neuro-symbolic system of this type has a neural model and symbolic model that aggregate their results to achieve more robust inference. In this paradigm, the neural component plays a role in modeling errors that arise due to the uncertainties present in the symbolic component. Alternatively, the symbolic component governs the neural component, ensuring that it adheres to specific constraints or rules. In the work of STLnet [MGF20], a neural student model learns to predict the next output sequence by learning temporal logic relations, while a symbolic teacher model generates an output sequence that is most "similar" to that prediction, but also fully satisfies the given relational constraints. The loss from both parts is summed together to direct the training of the neural model.

Symbolically-constrained Neural. This type of work adds a symbolic component to the learning process of a neural model to follow constraints, norms, or rules. It is somehow similar to the previous type but is slightly different in that this type uses a much smaller symbolic part than the neural model to guide the deep learning, while the type of Neural-Symbolic aggregation usually has two independent neural and symbolic models, both of which can perform inference. A recent example of this type is Pylon [ALT22], which trains deep learning models with user-defined constraints on the output. In Pylon, symbolic constraints are converted to an additional loss added to the traditional error cost.

Symbolically-structured Neural. In this paradigm, the neural network has a special structure generated from symbolic rules. An early work example is Logic Tensor Networks [BGS20], which generates a first-order logic language into TensorFlow computational graphs. Pix2rule [CR21] has a differentiable linear layer in a deep neural network, which is biased to capture the semantics of AND and OR to extract spatial symbolic rules from that layer. A recent work TOQ-net [MLG21] further captures temporal relations. It takes as inputs the properties and relationships of objects, and contains layers that are carefully structured to perform either object or temporal quantification.

Neurally-accelerated Symbolic. This type of neuro-symbolic system takes the advantage of efficient evaluation from neural networks to accelerate the training and inference. A neural model takes the place of a either slow or non-differentiable symbolic program while keeping the latter's functionality. An example is Neuroplex [XGV20], which adopts a knowledge distillation approach to train a neural model that can replace the logic reasoner for complex event pattern detection. Another recent example is the NEAR [SZS20] method in the field of program synthesis, which uses neural relaxation as the heuristic function to guide the search for an optimal program architecture.

The application of neuro-symbolic methods in CEP is an ongoing and promising area of research. In this work, we aim to investigate the potential of specific types of neuro-symbolic approaches in effectively handling multimodal sensor data for complex event detection in CEP. By comparing neural-only methods with neuro-symbolic methods, we seek to gain insights into the differences and advantages of incorporating symbolic reasoning in such a context. This research endeavor will contribute to the advancement of CEP by enhancing our understanding of the benefits of neuro-symbolic approaches and enabling the design of more effective methods for handling complex events across multiple sensor modalities.

CHAPTER 2

Related Works

In this work, we compare neural-only approaches with neuro-symbolic approaches for complex event detection using multimodal sensor data. Two relevant works, Neuroplex[XGV20] and DeepProbCEP[RXT23], propose neuro-symbolic methods for sensor data complex event detection.

Neuroplex. The Neuroplex framework consists of two main components: a neural network module and a knowledge injection module. The neural network module is responsible for learning patterns from raw sensor data, while the knowledge injection module incorporates domain-specific knowledge into the learning process. This combination allows the model to leverage the power of neural networks for pattern recognition while benefiting from the explicit representation of complex event rules.

DeepProbCEP. The DeepProbCEP is an end-to-end system built on top of DeepProbLog [MDK18, MDK19], a probabilistic programming language that makes logic rules differentiable. It comprises perception and reasoning levels, employing neural architectures for simple event classification and detecting complex events based on user-defined rules. It allows the injection of human knowledge to train with sparse data.

However, these works primarily focus on detecting relatively simple and short complex event patterns. In their cases, a complex event is defined by the combination of three or fewer simple events within a limited number of windows. In contrast, our work detects more realistic and complex events in the context of daily health monitoring. We define complex events with longer and more intricate patterns and design a daily human activity simulator to generate stochastic data sequences that follow these patterns. This allows for a wide range of combination possibilities of simple events within each complex event. Additionally, our study specifically explores multimodal settings, an area that has not been extensively covered in previous research.

CHAPTER 3

Methodology

To begin, we outline the complex event detection task, the development of a complex event generator, and the creation of a multimodal complex event dataset. Next, we provide an overview of the complex event detection system, which consists of two modules: the multimodal fusion module and the backbone module.

Subsequently, we delve into the details of the multimodal fusion model, which aims to leverage both audio and IMU sensor data by utilizing fusion embedding techniques. Lastly, we focus on the backbone module, a crucial component for comparing different models in learning complex event patterns. We select three baseline deep learning models: Long Short-Term Memory Networks (LSTMs), Temporal Convolutional Networks (TCNs), and Transformers. Additionally, we consider a neuro-symbolic model that integrates neural networks with finite state machines (FSMs) to facilitate complex event detection.

3.1 Complex Event Detection Task

In this research, we focus on the real-time complex event detection task within the field of CEP. The goal of this task is to not only determine the occurrence of specific complex events but also provide information about the timing of these events. In this section, we provide a comprehensive formulation of the complex event detection problem that we concentrate on. Additionally, we develop the generator of complex events and elaborate on the format of the multimodal complex event dataset used in our study.

3.1.1 Problem Formulation

Before we head toward the formulation of the complex event detection problem, let us first define what is an atomic event and what is a complex event. An atomic event is a foundational building block, a low-level and smallest component that cannot be further divided in a problem domain, whereas a complex event is a higher-level event that is built upon relevant atomic events following some temporal patterns.

For example, we can define the sport "high jump" as a complex event that is composed of the consecutive occurrence of atomic events "standing", "running", "jumping" and "landing". In this example, the temporal pattern is simple as atomic events are consecutive and follow one specific order. Another example is in a home health monitoring system, suppose we have atomic events "walking", "using restroom", "washing hands", "eating" and "drinking", we can define an complex event "unsanitary restroom usage" as "no washing hands after using the restroom before performing any activity other than walking, or simply walking away for more than 1 minute after using restroom". In this case, the pattern of the complex event is more intricate as the temporal dependency of those atomic events is not simply sequential.

We define the complex event detection task as follows. Let us consider a problem domain with a set of atomic events A, consisting of n elements denoted as $A = a_1, a_2, \ldots, a_n$. Additionally, we have a set of complex events of interest with k elements $E = e_1, e_2, \ldots, e_k$, which are composed using these atomic events. Each complex event e_i is defined by a subset of the atomic events A_i, where the combination of these atomic events follows a specific pattern:

$$e_i = R_i(A_i) = R_i(a_{i_1}, a_{i_2}, \ldots, a_{i_{n_i}}), \quad a_{i_j} \in A_i, \quad 1 \leq j \leq n_i, \quad 1 \leq i \leq k. \tag{3.1}$$

Here, R_i represents the temporal pattern function that maps the relevant atomic events into complex event e_i, $A_i \subseteq A$, and n_i is the number of atomic events in the subset A_i.

Without loss of generality, let's assume we receive data streams from two sensors with different modalities to detect complex events. The sensor data from modality M_1 is denoted as $\mathbf{X_1}$, and the sensor data from modality M_2 is denoted as $\mathbf{X_2}$. These sensors have distinct sampling rates: the sampling rate for M_1 sensor is r_1, and for M_2 sensor, it is r_2. At each time step t greater than the smallest sampling rate, we receive a multimodal sensor data pair:

$$\mathbf{D}_t = (\mathbf{X_1}(t), \mathbf{X_2}(t)). \tag{3.2}$$

Here, $\mathbf{X_1}(t)$ is a $(r_1 \times t) \times m_1$ dimensional matrix, where $r_1 \times t$ represents the number of samples within time t, and m_1 is the feature dimension of modality M_1. Similarly, $\mathbf{X_2}(t)$ is a $(r_2 \times t) \times m_2$ dimensional matrix, and m_2 is the feature dimension of modality M_2.

At each time step t, there exists a corresponding ground truth label y_t, representing the complex event class label for e_t occurring at that time step. The objective of a complex event detection system f that takes sensor data as input over time T is to accurately predict the complex event label for each time step, i.e.,

$$\min |\hat{y_t} - y_t|, \quad \text{where } \hat{y_t} = f(\mathbf{D}_t), \quad 1 \leq t \leq T, \tag{3.3}$$

Let $\mathbf{y} = y_1, y_2, \ldots, y_T$ be the ground-truth complex event label sequence. Equation 3.3 can be expressed in vector form as

$$\min ||f(\mathbf{D}) - \mathbf{y}||, \quad \text{where } \mathbf{D} = (\mathbf{X_1}, \mathbf{X_2}). \tag{3.4}$$

3.1.2 Daily Human Activity Simulator

Given the limited availability of datasets for the complex event detection task within the field of CEP, we undertake the task of constructing our dataset. Creating this dataset manually would be extremely laborious because complex events involve more than a fixed sequential combination of atomic events. To address this challenge, we develop a stochastic complex event generator capable of generating event sequences for each complex event class. This generator provides the flexibility to generate complex event sequences of varying lengths. The generator can be used to generate complex event sequences of any length. In particular, we design a complex event generator focused on simulating daily human activities within the context of a healthcare-monitoring system.

Overview of Daily Activity Simulator. The daily human activity simulator is designed with a hierarchical structure, taking into account three levels: stage-level, activity-level, and action-level, when generating complex event sequences. The conceptual design is as follows:

- **Stage-Level**: A day is divided into different stages, such as *Sleep Stage*, *Daytime Stage*, *Meal Stage*, and so on. Each stage contains a distinct distribution of activities.
- **Activity-level**: Each stage is associated with a specific set of activities. For example, the *Sleep Stage* typically includes activities like "Sleep" and "Use restroom," whereas the "Daytime Stage" involves activities such as "Walk," "Sit," "Work," "Drink," and "Use restroom".
- **Action-level**: Each activity is further defined by a combination of atomic actions following specific rules. Consequently, each activity comprises a temporal sequence of atomic actions.

The simulator is designed to generate action sequences that adhere to some daily regular pattern within the specified start and end times. Now, let us delve into a comprehensive explanation of each hierarchy in the simulator.

Stage-level. The simulator is designed to have various predefined stages, each starting randomly within a specific time interval. For example, the *Sleep Stage* starts at some point between 10 p.m. and 12 p.m., following a Gaussian distribution. Fig. 3.1 visually represents the mechanism of the simulator at the stage level. A previous stage terminates when a new stage commences. Certain stochastic stages may not occur. For instance, a *Meal Stage* during the noon could be skipped if someone forgets to have lunch.

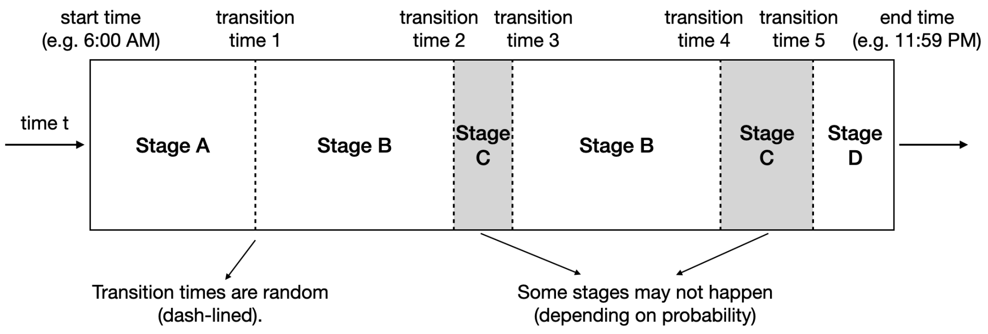


Figure 3.1: **Illustration of the simulator at a stage level.** In this example, the simulator operates with four stages denoted A, B, C, and D. Following the designated start time (e.g., 6:00 a.m.), the simulator initiates in the first stage, A (e.g., a *Sleep Stage*). The moment when the simulator transitions into the subsequent stage B is referred to as the "transition time," indicated by the dashed line in the figure. This transition time is predefined and occurs within a specific time interval, following a Gaussian distribution. The greyed-out blocks represent stages that have a probability of not occurring, as determined by user-defined parameters (e.g., a *Meal Stage* may be skipped).

In the context of the daily human activity simulator, we have designed four distinct stages, each associated with a specific set of activities:

- *Daytime Stage*: Activities include “Walk”, “Sit”, “Work”, “Drink”, and “Use Restroom”.
- *Evening Stage*: Activities include “Walk”, “Sit”, “Brush Teeth”, and “Use Restroom”.
- *Meal Stage*: Activities include “Walk”, “Sit”, “Wash”, “Drink”, and “Eat”.

The subsequent part will delve into the specific workings of the simulator within each stage to generate the corresponding activities.

Activity-level. Within each stage, the simulator generates activities based on predefined probabilities, with random durations that fall within a user-defined interval. These activities are mutually exclusive in the time domain, meaning that only one activity can occur at any given time point. Fig. 3.2 illustrates the activities generated within each stage. In each stage, there is a selection of activities to choose from. Once a previous activity concludes, the simulator randomly selects a new activity from the available set of activities within the stage.

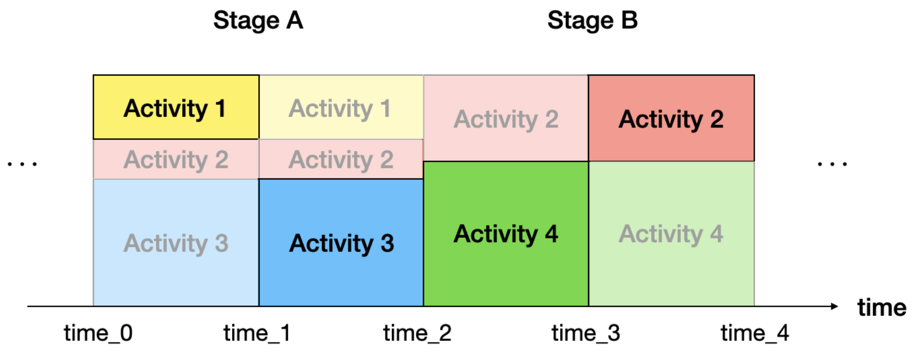


Figure 3.2: **Activities generated in each stage.** This figure showcases the activities generated within *Stage A* and *Stage B*. *Stage A* comprises Activity 1, Activity 2, and Activity 3, while *Stage B* contains Activity 2 and Activity 4. At *time_0*, the simulator is in *Stage A*, and it randomly selects Activity 1. After *time_1*, Activity 1 concludes, and a new activity is randomly chosen with a certain probability. In this example, Activity 3 is selected. Subsequently, after the conclusion of Activity 3 at *time_2*, the simulator transitions to *Stage B*. At this point, the simulator can choose between Activity 2 and Activity 4. In this instance, Activity 4 is generated at *time_2* and persists for a random duration, followed by the generation of Activity 2 upon its conclusion.

Additionally, each activity is defined based on specific rules regarding atomic events. Further details regarding these rules will be discussed in the following part.

Action-level. Each activity is composed of a combination of primitive actions, also referred to as atomic events, following specific rules. Similarly to activities, each atomic event has a random duration within a user-defined time interval. Here are a couple of examples:

- The “Use Restroom” activity is defined as the sequential combination of atomic events: ‘walk’ $\rightarrow$ (‘wash’) $\rightarrow$ ‘sit’ $\rightarrow$ ‘flush-toilet’ $\rightarrow$ (‘wash’) $\rightarrow$ ‘walk’, where the parentheses indicate that the atomic action may or may not occur.
- The “Work” activity randomly switches atomic events between ’sit’, ’type’, ’click-mouse’, and ’drink’..

Fig. 3.3 presents an example of the distribution and definition of activities within the *Daytime Stage*.

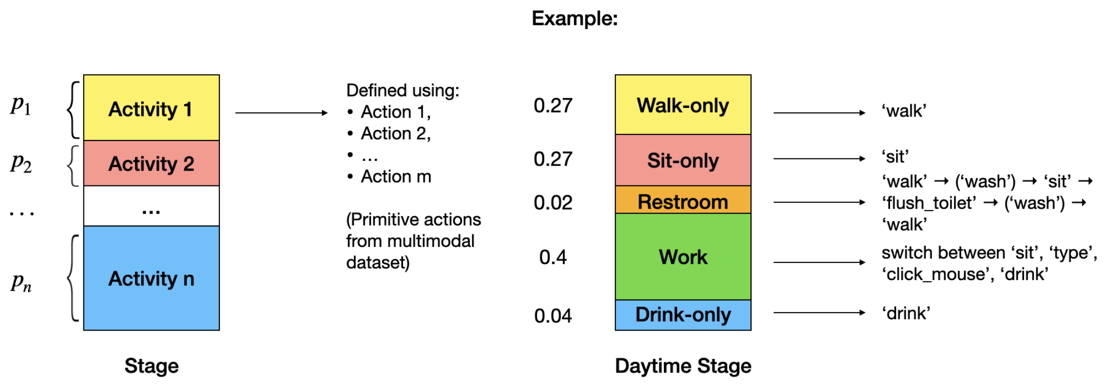


Figure 3.3: **The simulator in an action-level view.** (**Left**) Within each stage, we have a set of n activities that occur according to a predefined distribution, where Activity i has a probability p_i of taking place when the simulator is in that stage. Each activity is defined by a temporal combination of relevant primitive actions (atomic events). (**Right**) In this example, the simulator is in the *Daytime Stage*. It encompasses the activities “Walk-only”, “Sit-only”, “Restroom,” “Work”, and “Drink-only”, with corresponding probabilities Prob $= [0.27, 0.27, 0.02, 0.4, 0.04]$ respectively. Each activity is defined by the pattern displayed on the right side.

The action sequences generated by the simulator are then used to create the multimodal complex event dataset. The definitions of the complex event classes used for our complex event detection task are described in Section 4.2.

3.2 System Pipeline Overview

In this section, we present the complex event detection system that we have designed to facilitate a comprehensive comparison between neural-only approaches and neuro-symbolic approaches. The design of the system takes into account several key factors: (1) How to handle the unequal sampling rates of data obtained from different modality sensors? (2) How to effectively utilize information from multiple modalities? (3) How to ensure a fair comparison of the long-term reasoning and memorization capabilities between neural-only and neuro-symbolic approaches, while excluding the influence of different pre-processing abilities of the sensor data by different models? (4) How to design neural-only and neuro-symbolic models that are well-suited for the complex event detection task?

To address these challenges, we propose a two-module system, as shown in Fig. 3.4. The first module is the multimodal fusion module m, which takes sensor data as input and produces a fusion embedding as output. The second module is the backbone module g, which takes the fusion embedding from the multimodal fusion module and predicts real-time complex event labels. Thus, the task objective, as defined in Eq. 3.3, becomes:

$$\min |\hat{y_t} - y_t|, \quad \text{where } \hat{y_t} = g\left(m\left(\mathbf{D}_t\right)\right), \quad 1 \leq t \leq T, \tag{3.5}$$

Here, $\mathbf{D}_t$ represents the multimodal data pair obtained at time step t, and y_t represents the ground-truth complex event class label occurring at time t.

In this system, we specifically focus on two modalities: audio and IMU sensor data. The multimodal fusion module m remains consistent, while the backbone module g is varied to include different neural and neuro-symbolic architectures, ensuring a fair comparison between them. In the upcoming sections, we will provide detailed explanations of the model architectures and training methods employed for each approach.

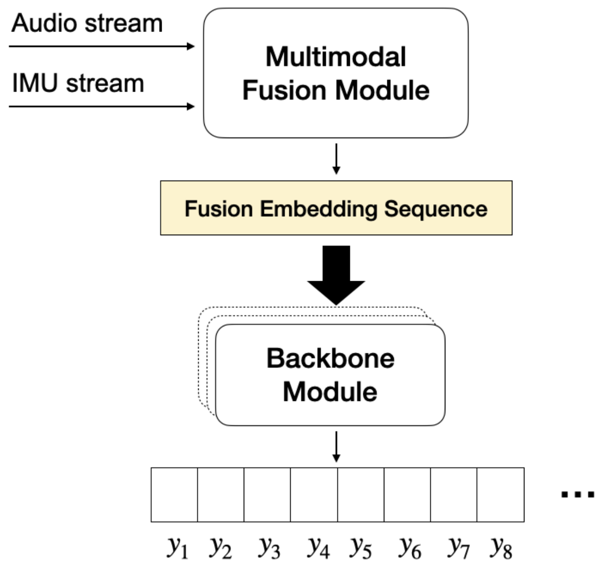

Audio stream
IMU stream
Multimodal Fusion Module
Fusion Embedding Sequence
Backbone Module
...
y_1 y_2 y_3 y_4 y_5 y_6 y_7 y_8

Figure 3.4: **System overview.** Our proposed system consists of two modules: a multimodal fusion module and a backbone module. The multimodal fusion module is responsible for integrating information from multiple modalities, while the backbone module handles the detection of complex events. To evaluate the performance of different approaches, the backbone module can be replaced with either neural-only models or neuro-symbolic models. The output of the system is a sequence of complex event labels (y) for each input sample data.

3.3 Multimodal Fusion Module

This module addresses the first two challenges discussed in Section 3.2: handling sample frequency inconsistency and leveraging both modalities for complex event processing. Additionally, it helps mitigate the third challenge by minimizing the impact of different sensor data feature extraction capabilities on the comparison results. This is achieved by ensuring that all backbone models receive the same data embedding processed by the multimodal fusion module. Consequently, the models are mainly evaluated based on their long-term complex event memorization and learning abilities.

The multimodal fusion module is responsible for processing audio and IMU sensor data

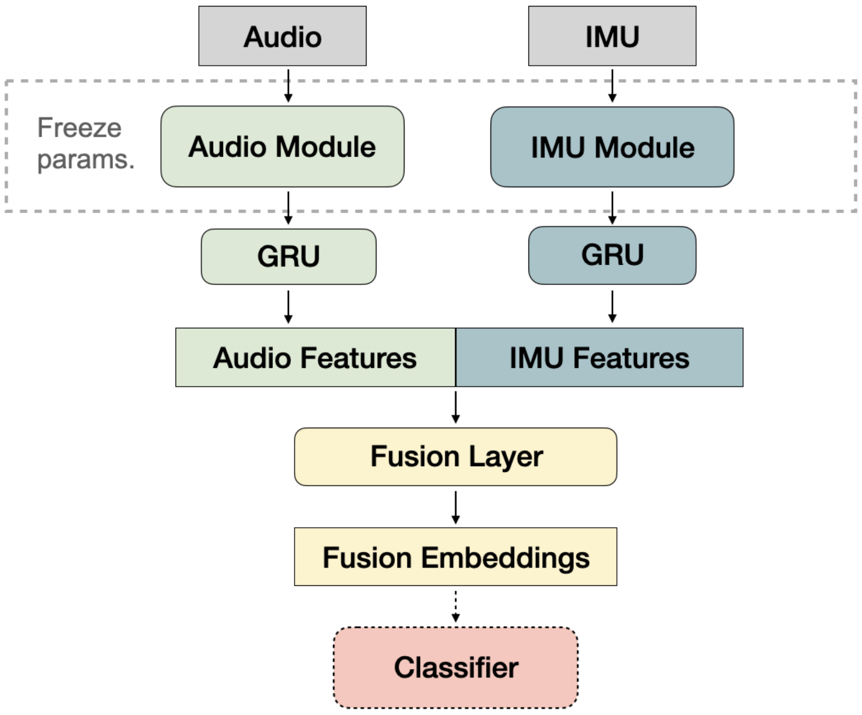


Figure 3.5: **Overview of the multimodal fusion module.** In the figure, a rectangular block represents data or an embedding vector, while a rounded corner rectangular block represents a neural network. The dashed-lined block indicates that it is omitted during the inference phase.

within fixed window sizes and generating the fused data embedding for each window. We have set the sliding window length to be the same as the window size so that there is no overlap between windows during the fusion process. In our design, the window size is set to 5 seconds. Therefore, the model initially processes the first 5 seconds of both audio and IMU data, and subsequently slides the window by 5 seconds to process the next set of data. To fuse the information from 5-second clips of both modalities, various approaches can be considered, such as early fusion, late fusion, and hybrid fusion [BAM17]. In our system, we employ early fusion, which involves integrating the features from both modalities immediately after they are extracted. Specifically, we concatenate the extracted features for both audio and IMU data to create a joint representation.

Fig. 3.5 provides an overview of the multimodal fusion module's structure. In the initial step of the multimodal fusion module, the model performs feature extraction on the 5-second audio and IMU data streams separately. This is accomplished by utilizing pre-trained audio and IMU modules dedicated to their respective modalities. Following this, the audio and IMU embeddings undergo individual Gated Recurrent Unit (GRU) layers. The outputs of these GRUs are then concatenated to create a comprehensive feature embedding. Next, the fusion embedding is passed through a Fusion Layer, which is trained alongside a downstream classifier specifically designed for atomic event classification. Notably, during the inference phase, the classifier is omitted, and only the output from the last hidden layer of the fusion layer is employed as the fusion embedding. We provide detailed explanations of each layer in the forthcoming sections.

3.3.1 Fusion Embedding

The multimodal fusion module in Figure 3.5 utilizes the BEATs model [CWW22] as the pre-trained audio module for processing raw audio sensor data. BEATs is an audio Transformer model that learns bidirectional encoder representations of audio data in a self-supervised manner. For the IMU module, we employ the LIMU-BERT model architecture [XZT21] and pre-train it on our own IMU dataset. LIMU-BERT is a self-supervised model designed to effectively capture temporal relations and feature distributions in IMU sensor measurements. In our multimodal fusion module, we freeze the parameters of both pre-trained models.

Following the audio and IMU modules, we include separate GRU layers. This step ensures that the dimensionality of the extracted audio feature embeddings and IMU feature embeddings are unified for symmetry. Since audio data is collected at a frequency of 16,000 Hz and IMU data is collected at a frequency of 20 Hz, for a window size of 5 seconds, we obtain audio sensor data of 80,000 samples and IMU sensor data of 100 samples. With audio data having 1 feature and IMU data having 6 features (acceleration on three axes and gyroscope on three axes), the shapes of the audio sensor data and IMU sensor data are $(80,000 \times 1)$ and (100×6), respectively. After passing through the corresponding pre-trained modules, the audio embedding becomes (248×768), and the IMU embedding becomes (100×72). The GRUs are then employed to process these embeddings into (1×128) vectors for both audio and IMU features.

To create a joint representation of audio and IMU features, we utilize a Fusion Layer. We concatenate the audio and IMU feature embeddings, resulting in a (1×256) vector. This vector is then passed through the Fusion Layer, which consists of a multi-layer perceptron (MLP) comprising a fully connected (FC) layer followed by a softmax activation layer. The output of the Fusion Layer is the fusion embedding, with a shape of (1×128). During training, the Fusion Layer is trained along with an MLP network to classify multimodal atomic events, such as ”walk,” ”sit,” and ”wash” (see Section 4.2 for a complete list of atomic action classes).

During inference, we remove the MLP classifier in the multimodal fusion module, and only use the output of the last hidden layer of the Fusion Layer as the learned fusion embedding. This fusion embedding serves as the input to the backbone modules, which are responsible for learning the complex event detection task

3.3.2 Training

The BEATs model used for the audio module is already pre-trained, while the LIMU-Bert model for IMU module is pre-trained on our dataset (see Section 4.1.2) following the training framework described in the original paper. After pre-training, we freeze the parameters of both models and focus on training the other components of the multimodal fusion module. For training and testing the fusion module, we utilize the multimodal atomic action dataset introduced in Section 4.2. The training process involves minimizing the cross-entropy loss, aiming to optimize the following objective:

$$\min_{\theta} L_m = -\min_{\theta} \frac{1}{N} \left(\sum_{i=1}^{N} c_i \cdot \log\left(\hat{c_i}\right) \right), \quad \text{where } \hat{c_i} = m_\theta(\mathbf{d_i}) \tag{3.6}$$

where $\mathbf{d}$ is the multimodal sensor data of a 5-second window size, $\hat{c_i}$ is the predicted label of the atomic action in that window, c_i is the corresponding ground truth label, and N represents the size of the multimodal atomic action dataset.

3.4 Backbone Module

This module plays a crucial role in the complex event detection system as it handles the learning of complex event patterns, and also enables comparison between neural-only and neuro-symbolic models. The system takes a multimodal data stream and divides it into 5-second segments. Within each segment, the multimodal fusion module extracts the fusion embedding using the learned fusion module. These fusion embeddings from all segments are then concatenated in time sequence and passed to the backbone module, which is trained to perform the complex event detection task.

The design of this module addresses the last two challenges discussed in Section 3.2. By employing a consistent fusion approach and using appropriate neural baseline and neuro-symbolic models, it ensures a fair comparison between different approaches. This allows for a comprehensive evaluation of the system's performance and facilitates the identification of the most effective approach for complex event detection.

3.4.1 Baseline Neural-only Models

We utilize three baseline deep learning models, namely LSTM, TCN, and Transformer, specifically designed to process variable-length sequential inputs. LSTMs are one type of RNNs that employ internal memory states to retain information over time, while the TCN model is a variant of CNNs designed for time sequence data. In the case of the Transformer model, we only employ the Transformer Encoder since our complex event detection task is a multi-label multi-class classification problem, eliminating the need for the Decoder typically used in translation tasks. TCNs and Transformers allow parallelizing computations, whereas LSTMs are limited to sequential processing and evaluation.

During complex event detection, the system predicts the current complex event occurring at each time step (outputting "0" if no interesting complex event is detected), so the output of the system should be a real-time sequence of complex event labels. Additionally, complex events are said to be detected only when the full pattern is observed at the current time. Therefore, when the system processes time sequence data, the backbone model at each timestamp, denoted as t, only requires knowledge of events from previous timestamps (0 to $t-1$) to infer the complex event occurring at the current time. Information from future timestamps ($t+1$, $t+2$, etc.) should not be used during training for complex event detection. Last but not least, the longest temporal dependency within a complex event pattern must be taken into consideration. For instance, if a complex event involves "brushing teeth in less than 2 minutes," the longest temporal dependency is 2 minutes. Consequently, when designing models with hyper-parameters that may impact the accessible context, it is essential to ensure that the model's "visible context" is at least as long as the duration of the complex event patterns.

To address these requirements, we design baseline neural-only models using a many-to-many architecture and adopt a causal structure. The many-to-many architecture enables the models to generate multi-label multi-class outputs, and the causal structure imposes a constraint on the models, restricting them to access only the data from previous timestamps in a real-time fashion. For TCN models, which have a receptive field, we ensure that the model's context length is greater than the longest complex event pattern range to capture the necessary information for accurate complex event detection.

Long Short-Term Memory. We use a Unidirectional LSTM instead of a Bidirectional LSTM. The former one works in a causal way, because it takes the input sequence and processes it iteratively, updating its internal memory state based on the current input and the accumulated information from previous time steps. The LSTM model has 2 LSTM layers with a hidden dimension of 128.

Temporal Convolutional Network. We adopt the TCN model with causal masking so that when performing convolutional operations all information from future timestamps are masked out. Similar to CNNs, a TCN model also has a receptive field, which refers to the region in the input space that influences the output of a particular neuron or unit in a neural network. In other words, it is the maximum number of steps back in time from the current sample at time t that a neuron "sees" or receives information from [BKK18]. By definition, the receptive field of a TCN model is calculated as

$$\text{Receptive field} = \#\text{ stacks of residuals blocks} * \text{kernel size} * \text{last dilation rate}. \tag{3.7}$$

We use a TCN model with one stack of residual blocks of 5 levels and a kernel size of 2, then the dilation rates are [1, 2, 4, 8, 16]. The number of filters for each level is 128, 128, 128, 256, and 256, respectively. Then the receptive field is

$$\text{Receptive field} = 1 \times 2 \times 16 = 32. \tag{3.8}$$

This is greater than the longest context length required by our complex events (see Section 4.2). For a 2-min pattern length, we require the TCN model to have a receptive field greater than $2 \times 60 \div 5 = 24$ (the number of seconds divided by the system window size).

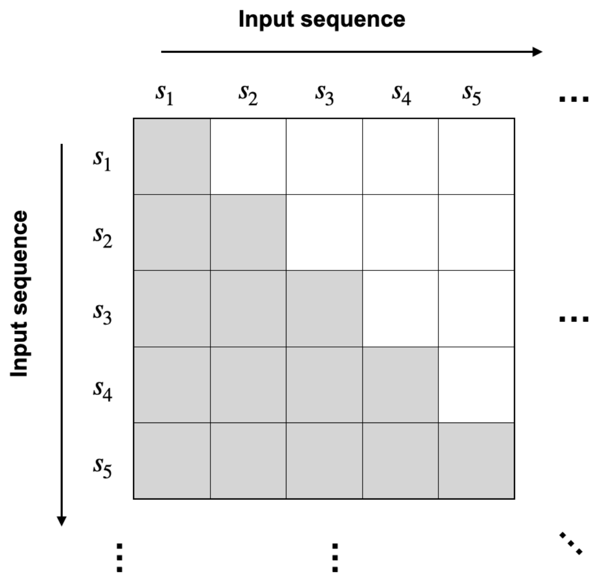

Input sequence
s_1 s_2 s_3 s_4 s_5 ...
Input sequence
s_1 s_2 s_3 s_4 s_5

Figure 3.6: **Self-attention mask of Transformer Encoder.** This figure illustrates the mask applied to the self-attention matrix of a transformer encoder. Given an input sequence s, s_i corresponds to the i-th token in s. The gray blocks are positions allowed to have attention weights applied, and the white blocks are positions that are masked out.

In our Transformer Encoder, we employ various techniques to capture temporal dependencies and ensure effective processing of the input sequence.

Firstly, we utilize an attention mask that restricts the model's attention to previous timestamps only, excluding information from future timestamps. This mask ensures a causal relationship between the attended positions. Both during training and inference, we apply a triangular attention mask on the encoder self-attention matrix, as depicted in Fig. 3.6. Consequently, the Encoder model can attend to input sequences causally, considering the temporal order of the data.

Additionally, to encode the positional information of each element within the sequence, we incorporate positional encoding into the embedding inputs. This technique assigns a unique representation to each position and enables the Transformer model to learn and understand temporal dependencies [VSP17]. Given the nature of the complex event detection task, which heavily relies on capturing temporal dependencies, the inclusion of positional encoding is crucial.

To enhance the model's attention mechanism, we employ multi-head attention with 8 heads. This enables the model to capture different types of relationships and patterns within the input data simultaneously, enhancing its capacity for complex event detection. Moreover, we choose 6 encoder layers in the Transformer Encoder, providing a deep architecture that can effectively capture and process complex event patterns.

3.4.2 Neuro-symbolic Model

In our comparison with the neural baseline models, we design a neuro-symbolic model that combines a neural layer with a finite state machine (FSM) architecture to leverage the strengths of both neural networks and symbolic reasoning.

The neural layer is the MLP classifier of the multimodal fusion module (as shown in Fig. 3.5) that was only used during fusion module training. It takes the fusion embedding extracted from each 5-second window as input and predicts the atomic action class. The predicted sequence of atomic actions is then passed to the FSMs that are specifically designed for each complex event.

The FSMs receive the predicted atomic action labels from the neural layer and generate complex event labels every 5 seconds. These FSMs are designed to capture the temporal dependencies and patterns of each complex event, and output the complex event class when a sequence of atomic actions matches its rule.

3.4.3 Training

The training mechanism for the neural-only baseline models is our main focus here, as the training of the MLP classifier used in the neuro-symbolic model has been discussed in the previous multimodal fusion module section.

A major challenge that we encounter during training neural models is the class imbalance issue. Due to the temporal sparsity property of complex events, the ground truth label sequence is almost all “0”, which means no complex events happen at current time t. Since this is very similar to the issue in the object detection field, where the number of negative classes (e.g. background objects) overwhelms the number of positive classes (e.g. foreground objects), leading to a class imbalance issue during training time. Therefore, we adopt a measure in the field of object detection, which is to use Focal Loss (FL) [LGG17].

FL is a variant of the cross-entropy loss that penalizes more heavily on mistakes made on less frequent class examples. Without loss of generality, we only consider a binary classification problem for notation simplicity. Let p_y be the estimated probability by the classifier for the class to be y. Then FL is written as

$$L\left(p_y\right) = -\alpha_y \left(1 - p_y\right)^{\gamma} \log\left(p_y\right) \tag{3.9}$$

where γ is the focusing parameter that smoothly adjusts the rate at which easy examples (classes that occur much more frequently) are down-weighted, and α_y is the class weight coefficient used to further adjust weights of each class on training loss.

In our case, given a complex event dataset with N training examples, and each example sequence y_i has length T, the training objective is to optimize the following loss:

$$\min_{\theta} L_{CE}\left(\theta\right) = -\sum_{i=1}^{N}\sum_{t=1}^{T} \alpha_{y_i(t)} \left(1 - p_{y_i(t)}\right)^{\gamma} \log\left(p_{y_i(t)}\right). \tag{3.10}$$

We choose $\gamma = 2$ as recommended by the original paper, and choose $\alpha_0 = 0.005$ for the most frequent complex event class “0” (though it means no complex events of interest happening), and the class weights for other classes are set to $\alpha_y = 0.45$.

CHAPTER 4

Dataset

4.1 Synthetic Multimodal Activity Dataset

To synthesize the multimodal complex event dataset, we begin by creating the multimodal activity dataset. This is done by combining activity classes from existing audio and IMU datasets. These activities later serve as the atomic action blocks from which we can build complex events based on specific patterns.

4.1.1 Audio Dataset

For the audio component of the multimodal dataset, we utilize the ESC-70 dataset. It is a combination of the ESC-50 dataset [Pic15] and the Kitchen20 dataset [MOF19]. The ESC-50 dataset consists of 2000 5-second labeled environmental audio recordings, with 50 different classes of natural, human, and domestic sounds. The Kitchen20 dataset is a collection of 20 labeled kitchen-related environmental sound clips. To include silent sound recordings, which are important for building complex event examples that involve empty sound, we also collected 40 additional 5-second clips. The original sampling rate of these recordings is 44.1 kHz, but we downsample them to 16 kHz for our dataset.

4.1.2 IMU Dataset

To incorporate the IMU component, we use the WISDM dataset. This dataset contains raw accelerometer and gyroscope sensor data collected from smartphones and smartwatches, at a sampling rate of 20 Hz. It was collected from 51 test subjects as they performed 18 activities for 3 minutes each [WYH19]. Based on the findings in a survey paper [ONW21], we choose to use only the data collected from the smartwatch, as it results in better classification accuracy compared to using data from both the smartphone and smartwatch. We segment the original data samples into non-overlapping 5-second clips, aligning them with the duration of the audio clips.

4.1.3 Multimodal Activity Class Definition

To synthesize the multimodal activity dataset, we define nine multimodal activity classes and generate 500 data samples per class. These classes are derived from the audio and IMU datasets. We select nine activity classes from the audio dataset, including "footsteps", "no sound", "brushing teeth", "mouse click", "drinking sipping", "eating", "keyboard typing", "toilet flush" and "water-flowing". From the IMU dataset, we choose nine corresponding activity classes: "walking", "sitting", "teeth", "sitting", "drinking", "eating pasta", "typing", "standing" and "standing". Each multimodal activity class is defined by pairing an audio class with an IMU class, as shown in Table 4.1. To generate data samples for each multimodal activity class, we randomly pair samples from the corresponding audio and IMU classes.

4.2 Synthetic Multimodal Complex Event Dataset

To create the multimodal complex event dataset, we utilize the atomic events defined by the activity classes in the multimodal activity dataset. We design complex event rules based on these atomic events and employ a stochastic simulator (as discussed in Section 3.1.2) along with Finite State Machines (FSMs) to generate the dataset.

Multimodal class	**Audio class**	**IMU class**
walk	footsteps	walking
sit	no sound	sitting
brush teeth	brushing teeth	teeth
click mouse	mouse click	sitting
drink	drinking sipping	drinking
eat	eating	eating pasta
type	keyboard typing	typing
flush toilet	toilet flush	standing
wash	water-flowing	standing

Table 4.1: **Definition of multimodal activity classes.**

4.2.1 Complex Event Class Definition

We define four complex events related to health violations in the context of a home health monitoring system, which is a meaningful and interesting application field of complex event processing. These complex events are defined as shown in Table 4.2.

4.2.2 Generating Multimodal Complex Event Dataset

To generate the multimodal complex event dataset with the defined complex event types, we first use the stochastic simulator (discussed in Section 3.1.2) to create activity sequences that span 5 minutes in duration.

Next, we construct real-time complex event labels for these activity sequences by designing Finite State Machines (FSMs) that incorporate the specific symbolic rules for the four complex events. At each time step t, the FSMs take the t-th activity class as input and output the current complex event label based on their current state. Therefore, given a activity sequence with T time steps $A = a_1, a_2, \ldots, a_T$, the FSMs generate a complex event

Complex event	Definition
e_0	No interesting complex events detected. The default class if none of the following complex events happen.
e_1	No hand washing after using the restroom, followed by engaging in other activities, or walking away for more than 1 minute.
e_2	No hand washing before meals (reset the complex event state if no eating occurs within 2 minutes).
e_3	Brushing teeth for less than 2 minutes (if no brushing occurs within 10 seconds, the timing for brushing teeth stops).

Table 4.2: **Definition of multimodal activity classes.** Note that e_0, e_1, e_2, and e_3 correspond to complex event class labels 0, 1, 2, and 3, respectively. If an event does not belong to classes e_1, e_2, or e_3, it is categorized as the default event e_0.

label sequence of the same length $\mathbf{y} = y_1, y_2, \ldots, y_T$. A visualization of a label sequence with length T is as follows:

$$\mathbf{y} = [0, 0, 0, 0, 0, 0, 1, \ldots, 0, 0, 0, 0, 2, 0]$$

Here, $\mathbf{y}_t = 1$ (or $\mathbf{y}_t = 2$) indicates that complex event e_1 (or complex event e_2) is detected at timestamp t, while $\mathbf{y}_t = 0$ indicates that no interesting complex events (including e_1, e_2, e_3) occur at timestamp t.

Finally, to create the multimodal complex event data, we map each activity sequence to a multimodal sensor data sequence. This is achieved by replacing each activity in the sequence with the corresponding multimodal activity data sample.

Since we are using a window size of 5 seconds, each complex event data example consists of 60 timestamps (as there are $5 \times 60/5 = 60$ timestamps in a 5-minute duration). The dimension of each timestamp is 128, which corresponds to the dimension of the fusion embedding vector. We generate 5000 examples for the multimodal complex event dataset.

CHAPTER 5

Evaluation

We evaluate three baseline neural-only models and one neuro-symbolic model using the multimodal complex event dataset built in the previous section.

5.1 Experiment Setup

For training, we utilize two Nvidia GPUs (1080-Ti). We use the SGD optimizer with a learning rate of $1e-3$ for all training processes. The multimodal fusion module is trained for 100 epochs, while the baseline neural-only models are trained for 2000 epochs.

5.2 Evaluation Metrics

In our experiments, calculating the mean accuracy alone is not informative enough to assess how well the models learn complex event patterns. This is because the complex event label sequence contains a large number of "0"s due to the temporal sparsity of complex events. To address this, we adopt common metrics used in the field of object detection to evaluate the performance of our backbone models.

We use precision, recall, and F1 score as the performance metrics. Given the number of true positives (tp), false positives (fp), and false negatives (fn), precision and recall scores are calculated as follows:

$$precision = \frac{tp}{tp+fp}, \quad recall = \frac{tp}{tp+fn} \tag{5.1}$$

The F1 score is the harmonic mean of precision and recall:

$$F_1 = \frac{2}{recall^{-1} + precision^{-1}} = 2 \cdot \frac{precision \cdot recall}{precision + recall} = \frac{2tp}{2tp + fp + fn} \tag{5.2}$$

In other words, a higher precision score reflects the model's ability to correctly classify negative samples, while a higher recall score reflects the model's ability to correctly identify all positive samples. Since we are particularly interested in detecting complex event classes 1, 2, and 3, we consider them as positive classes, while complex event class 0 is treated as the negative class, which constitutes the majority of the dataset.

5.3 Experiment Results

First, we test the multimodal fusion model on the multimodal activity dataset to assess the accuracy of the classifier, which serves as the neural component of the neuro-symbolic model. The classifier achieves 91% accuracy on the test set.

Next, we calculate the metric scores of the baseline neural models and the neuro-symbolic model for complex event detection. We evaluate precision, recall, and F1 scores for each class and calculated average scores across multiple classes. The results are summarized in Table 5.1 and Table 5.2.

Different performances on various complex events. We observe that different neural-only or neuro-symbolic models had varying scores on different complex events. This could be attributed to different pattern types of complex events introducing different learning challenges. For example, e_3 requires continuous counting capability to detect violations of spending enough time brushing teeth, while e_1 has a more complex rule, being triggered by either performing activities other than washing hands and walking right after using the restroom or walking away for more than 1 minute without washing hands after using the restroom. To some extent, e_1 can be considered a combination of e_2, and e_3 in terms of event types.

	precision					*recall*				
	e_0	e_1	e_2	e_3	Avg.	e_0	e_1	e_2	e_3	Avg.
LSTM	0.98	0.0	0.0	0.0	0.25	**1.0**	0.0	0.0	0.0	0.25
TCN	**0.999**	0.17	0.16	0.06	0.35	0.87	**0.97**	**1.0**	0.92	**0.94**
Transformer	0.99	0.0	0.0	0.07	0.27	0.94	0.0	0.0	0.72	0.42
Neural + FSM	0.996	**0.79**	**0.99**	**1.0**	**0.94**	0.999	0.28	0.99	**1.0**	0.82

Table 5.1: **Precision and recall scores of each complex event.** The precision and recall scores are calculated separately for each complex event class. We also calculate the macro-average precision and recall scores (denoted as Avg.) as an overall measure.

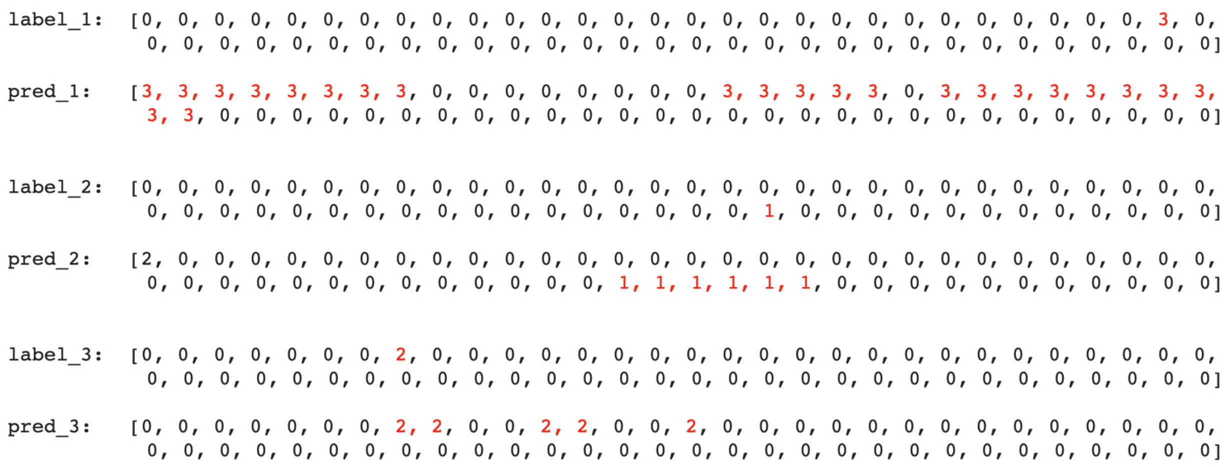
	Avg. $F1$ score (all)	Avg. $F1$ score (positive)
LSTM	0.25	0.0
TCN	0.40	0.23
Transformer	0.28	0.057
Neural + FSM	**0.85**	**0.80**

Table 5.2: **Macro-average $F1$ scores in all classes and positive classes.** The macro-average $F1$ scores are calculated separately on two scales. The first mean $F1$ score is evaluated on all four complex event classes, while the second mean $F1$ score is calculated only using the positive classes. The positive classes include meaningful complex events that happen rarely, namely e_1, e_2 and e_3, while the default complex event class e_0 is considered as the negative class.

Performance of LSTM and Transformer models. The LSTM model achieves almost perfect precision and recall scores on the default complex event class e_0, but has zero precision and recall scores for all other event classes e_1, e_2, and e_3. This indicates a complete failure in learning complex event rules, as it consistently generates "0" labels for all instances. Similarly, the Transformer model exhibits almost zero precision scores on positive classes (e_1, e_2, and e_3), and a very high precision score on e_0. It is slightly better than the LSTM model in that the recall score of e_3 is 0.72. Recall that the precision score indicates the ability to avoid labeling negative samples as positive, and the recall score implies the ability to correctly recognize positive samples. Having a high recall score and low precision score (but not equal to zero) for e_3 means that the Transformer model generates a lot of "3" labels for e_3 blindly along with numerous "0" for e_0. Overall, both the LSTM and Transformer models struggle to effectively learn complex event rules.

Performance of the TCN model. Among the neural-only baseline models, the TCN model stands out as the best one. It achieves the highest recall scores for each class and also has the highest macro-average recall score, indicating its ability to accurately identify the type of complex event occurring. However, the model exhibits low precision scores for all classes except e_0, suggesting that it struggles to capture the precise timing of each complex event. This limitation becomes more evident when examining the predicted label sequences generated by the TCN model, as illustrated in Figure 5.1. In these examples, the predicted positive labels ("1", "2", and "3") occur within a certain range around the positions of the ground-truth positive labels. Rather than generating a single positive label, the TCN tends to produce consecutive positive labels, leading to numerous false positives. In summary, while the TCN model learns the complex event pattern to some extent, it still falls short in detecting the precise timing of events.

Limitations of neural-only models. In addition to the aforementioned performance issue, the TCN and Transformer models suffer from a significant limitation regarding the longest temporal patterns they can learn. This limitation is determined by the design of



```
label_1:  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

pred_1:   [3, 3, 3, 3, 3, 3, 3, 3, 0, 0, 0, 0, 0, 0, 0, 0, 3, 3, 3, 3, 3, 0, 3, 3, 3, 3, 3, 3, 3, 3,
           3, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


label_2:  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

pred_2:   [2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


label_3:  [0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

pred_3:   [0, 0, 0, 0, 0, 0, 0, 2, 2, 0, 0, 2, 2, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Figure 5.1: **Example complex event label sequences predicted by TCN.** Here are three examples of complex event label sequences predicted by the TCN model. Each example consists of a ground truth sequence (`label_i`) and a corresponding prediction sequence (`pred_i`). Positive class labels are highlighted in red in both the ground truth and prediction sequences.

the models themselves. For instance, if we aim to detect a complex event with a pattern spanning 24 hours, we would need to employ an exceptionally large TCN or Transformer model capable of processing a 24-hour input sequence at once, which is impractical. Alternatively, if we use a normal-sized TCN or Transformer model, we would have to segment the 24-hour sequence into smaller parts and feed them to the model sequentially. However, since TCN and Transformer models lack memory states to retain information across these segments, the information within each segment is considered independent. Consequently, these models cannot effectively detect complex events with patterns longer than the size of the segmentation. While LSTMs theoretically can relate information across segments, they suffer from gradient vanishing problems, as evidenced by the LSTM baseline model's failure to learn complex events even with a 5-minute duration.

The neuro-symbolic model has the best overall performance. The Neural + FSM model achieves the best precision and F1 scores, along with a reasonably good recall score, indicating its superior capability to detect complex events. However, it is worth noting that the model exhibits a relatively low recall score for e_1. This could be attributed to the fact that, by design, a classification error made on an atomic event is more likely to result in an incorrect detection of e_1 compared to other complex events. Therefore, the complex event detection performance of the Neural + FSM model heavily relies on the accuracy of its neural component, which classifies atomic activities in 5-second windows. Nevertheless, the advantage of this model in the context of CEP is evident as it can detect complex events in sequences of infinite length through the integration of finite-state machines (FSMs), which play a crucial role in keeping track of complex events by maintaining their states.

CHAPTER 6

Discussion

In this study, we evaluate of neural-only and neuro-symbolic approaches for complex event detection with multimodal data. We synthesize a multimodal activity dataset and a multimodal complex event dataset within the context of daily human activity and health monitoring. We design a two-module system, where the first module processes multimodal data and the second module focuses on learning complex event patterns. Baseline neural-only and neuro-symbolic models are used as replacements for the second module to facilitate evaluation. However, there are several limitations in our current approach, as well as avenues for future work, which we discuss below:

Multimodal activity dataset. One flaw in our dataset design is that the multimodal classes can be classified solely based on audio classes. This limitation arises from the fact that different multimodal classes correspond to different audio classes, as defined in Table 4.2. Consequently, the learning of the multimodal fusion model may be biased towards audio features, potentially disregarding the importance of IMU features in classifying multimodal classes. To address this, it would be beneficial to design more diverse multimodal classes and utilize additional real-life multimodal datasets for evaluation.

Multimodal processing model. Our current multimodal fusion model employs a straightforward approach of early fusion, where audio and IMU features are concatenated and mapped to a shared space. To improve the model's performance, alternative backbone models such as cross-modality attention models could be explored. Additionally, considering scenarios with missing modality data and developing strategies to handle such cases would enhance the model's robustness. Furthermore, the current window size of 5 seconds for the fusion model may be relatively large for human activity classification using sensor data. Exploring smaller window sizes could potentially capture finer temporal dynamics.

Complex event design. When designing complex events, different types of patterns may present varying levels of difficulty for pattern learning. It would be valuable to refine our evaluation experiments by categorizing complex events based on their patterns. Additionally, testing models on complex events of various lengths, beyond the 5-minute sequences examined in this paper, would provide a more comprehensive assessment of their capabilities.

Training and evaluation. We employ the focal loss to address class imbalance introduced by the complex event label sequence. However, focal loss requires hyper-parameter tuning, and finding optimal choices can be challenging. Exploring alternative loss functions that better address this class imbalance issue could be beneficial. Moreover, while we currently define the complex event detection task as a multi-label multi-class classification problem, there may be alternative approaches that better suit the nature of the task. Additionally, designing better evaluation metrics that more effectively capture the extent to which models learn complex event patterns, rather than relying solely on precision, recall, and F1 scores, would provide more intuitive insights.

Neuro-symbolic approach. The neuro-symbolic model we implement in this study integrates a neural module with FSMs in an intuitive manner. However, the performance of complex event detection heavily relies on the accuracy of the neural component in classifying atomic events. This approach may not represent the best solution for complex event detection. Therefore, it is crucial to explore and develop more robust neuro-symbolic methods that can effectively leverage both neural and symbolic components in the context of CEP.

In conclusion, our evaluation of neural-only and neuro-symbolic approaches for complex event detection with multimodal data provides valuable insights. However, addressing the limitations mentioned above and pursuing future work in these areas would enhance the effectiveness and applicability of complex event detection systems in multimodal settings.

CHAPTER 7

Conclusion

In this study, we evaluated of neural-only and neuro-symbolic approaches for complex event detection with multimodal data. Our findings indicate that the neuro-symbolic approach is better suited for the complex event detection task. We began by formalizing the complex event detection task and developed a daily human activity simulator to generate stochastic sample sequences for complex events. Through this process, we synthesized a multimodal activity dataset and a multimodal complex event dataset within the context of daily health monitoring.

To facilitate evaluation, we designed a two-module complex event detection system. The first module processed multimodal data, while the second module focused on learning complex event patterns. As part of our evaluation, we replaced the second module with baseline neural-only and neuro-symbolic models. These models allowed us to compare and assess their performance in complex event detection.

Future work will focus on evaluating complex events with various types and lengths, developing more robust and effective neuro-symbolic methods, advancing multimodal sensor data processing techniques, and so on.

REFERENCES

[ALT22] Kareem Ahmed, Tao Li, Thy Ton, Quan Guo, Kai-Wei Chang, Parisa Kordjamshidi, Vivek Srikumar, Guy Van den Broeck, and Sameer Singh. “PYLON: A PyTorch Framework for Learning with Constraints.” *Proceedings of the AAAI Conference on Artificial Intelligence*, **36**(11):13152–13154, Jun. 2022.

[BAM17] Tadas Baltrusaitis, Chaitanya Ahuja, and Louis-Philippe Morency. “Multimodal Machine Learning: A Survey and Taxonomy.” *CoRR*, **abs/1705.09406**, 2017.

[BGS20] Samy Badreddine, Artur d’Avila Garcez, Luciano Serafini, and Michael Spranger. “Logic Tensor Networks.” *CoRR*, **abs/2012.13635**, 2020.

[BKK18] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. “An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling.” *CoRR*, **abs/1803.01271**, 2018.

[CMG14] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. “Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation.” *CoRR*, **abs/1406.1078**, 2014.

[CR21] Nuri Cingillioglu and Alessandra Russo. “pix2rule: End-to-end Neuro-symbolic Rule Learning.” *CoRR*, **abs/2106.07487**, 2021.

[CWW22] Sanyuan Chen, Yu Wu, Chengyi Wang, Shujie Liu, Daniel Tompkins, Zhuo Chen, and Furu Wei. “BEATs: Audio Pre-Training with Acoustic Tokenizers.”, 2022.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory.” *Neural computation*, **9**(8):1735–1780, 1997.

[Kau22] Henry Kautz. “The Third AI Summer: AAAI Robert S. Engelmore Memorial Lecture.” *AI Magazine*, **43**(1):93–104, Mar. 2022.

[LGG17] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár. “Focal Loss for Dense Object Detection.” *CoRR*, **abs/1708.02002**, 2017.

[MDK18] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. “DeepProbLog: Neural Probabilistic Logic Programming.” In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[MDK19] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. “DeepProbLog: Neural Probabilistic Logic Programming.” *CoRR*, **abs/1907.08194**, 2019.

[MGF20] Meiyi Ma, Ji Gao, Lu Feng, and John Stankovic. "STLnet: Signal Temporal Logic Enforced Multivariate Recurrent Neural Networks." In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pp. 14604–14614. Curran Associates, Inc., 2020.

[MLG21] Jiayuan Mao, Zhezheng Luo, Chuang Gan, Joshua B. Tenenbaum, Jiajun Wu, Leslie Pack Kaelbling, and Tomer D. Ullman. "Temporal and Object Quantification Networks." *CoRR*, **abs/2106.05891**, 2021.

[MOF19] Marc Moreaux, Michael Garcia Ortiz, Isabelle Ferrané, and Frederic Lerasle. "Benchmark for Kitchen20, a daily life dataset for audio-based human action recognition." In *2019 International Conference on Content-Based Multimedia Indexing (CBMI)*, pp. 1–6, 2019.

[MTC22] Ludovico Mitchener, David Tuckey, Matthew Crosby, and Alessandra Russo. "Detect, Understand, Act: A Neuro-Symbolic Hierarchical Reinforcement Learning Framework (Extended Abstract)." In Lud De Raedt, editor, *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pp. 5314–5318. International Joint Conferences on Artificial Intelligence Organization, 7 2022. Sister Conferences Best Papers.

[ONW21] Bolu Oluwalade, Sunil Neela, Judy Wawira, Tobiloba Adejumo, and Saptarshi Purkayastha. "Human Activity Recognition using Deep Learning Models on Smartphones and Smartwatches Sensor Data.", 2021.

[Pic15] Karol J. Piczak. "ESC: Dataset for Environmental Sound Classification." In *Proceedings of the 23rd Annual ACM Conference on Multimedia*, pp. 1015–1018. ACM Press, 2015.

[RXT23] Marc Roig Vilamala, Tianwei Xing, Harrison Taylor, Luis Garcia, Mani Srivastava, Lance Kaplan, Alun Preece, Angelika Kimmig, and Federico Cerutti. "DeepProbCEP: A neuro-symbolic approach for complex event processing in adversarial settings." *Expert Systems with Applications*, **215**:119376, 2023.

[SGT09] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. "The Graph Neural Network Model." *IEEE Transactions on Neural Networks*, **20**(1):61–80, 2009.

[SZS20] Ameesh Shah, Eric Zhan, Jennifer J. Sun, Abhinav Verma, Yisong Yue, and Swarat Chaudhuri. "Learning Differentiable Programs with Admissible Neural Heuristics." *CoRR*, **abs/2007.12101**, 2020.

[VSP17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. "Attention Is All You Need." *CoRR*, **abs/1706.03762**, 2017.

[WYH19] Gary M. Weiss, Kenichi Yoneda, and Thaier Hayajneh. "Smartphone and Smartwatch-Based Biometrics Using Activities of Daily Living." *IEEE Access*, **7**:133190–133202, 2019.

[XGV20] 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*, SenSys '20, p. 489–502, New York, NY, USA, 2020. Association for Computing Machinery.

[XZT21] Huatao Xu, Pengfei Zhou, Rui Tan, Mo Li, and Guobin Shen. "LIMU-BERT: Unleashing the Potential of Unlabeled Data for IMU Sensing Applications." In *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*, SenSys '21, p. 220–233, New York, NY, USA, 2021. Association for Computing Machinery.