DEPARTMENT OF DIGITAL SYSTEMS

NCSR DEMOKRITOS
INSTITUTE OF INFORMATICS AND
TELECOMMUNICATIONS

Universuty of Pireaus and National Center for Scientific Research Demokritos

MSc in Artificial Intelligence

Department of Digital Systems and Institute of Informatics and Telecommunications

Postgraduate Thesis

The DeepProbCEP system for Neuro-Symbolic Complex Event Recognition

Varsou Panagiota
mtn2002

Supervisor:

Nikos Katzouris
PhD, Researcher in NCSR

Athens, January, 2024

Approved by the selection board on February 2024.

Artikis Alexander Paliouras George

Varsou Panagiota
Department Digital Systems and Institute of Informatics and Telecommunications
University of Pireaus and NCSR Demokritos

Στην ψυχική μου ηρεμία, και τα ελεύθερα σαββατοκύριακα.

Acknowledgements

Ευχαριστώ τους γονείς μου, τον αδερφός μου, τον Αναστάση και τις φίλες μου. Καθώς και όλους όσουν με στήριξαν και με βοήθησαν κατά την διάρκεια αυτού του μεταπτυχιακού αλλά και κατά την συγγραφή της διπλωματικής εργασίας.

Ιανουάριος 2024
Βάρσου Παναγιώτα

Περίληψη

Αυτή η έρευνα εξερευνά τη συμβολή των νευρωνικών δικτύων και της συμβολικής συλλογιστικής, επικεντρώνοντας ιδιαίτερα στην εφαρμογή της νευρο-συμβολικής μάθησης και συλλογιστικής στην Αναγνώριση Σύνθετων Συμβάντων (CER). Κεντρικό στοιχείο αυτής της μελέτης είναι η εξερεύνηση του DeepProbLog, ενός κορυφαίου νευρο-συμβολικού πλαισίου. Το DeepProbLog διακρίνεται από την επιδέξια ενσωμάτωση της λογικής εκφραστικότητας με τη στατιστική δύναμη των νευρωνικών δικτύων και του προγραμματισμού πιθανοτήτων. Ενώ το πλαίσιο Neuroplex προσφέρει επίσης έναν αξιόπιστο συνδυασμό νευρωνικών δικτύων και λογικού προγραμματισμού, το DeepProbLog επιλέχθηκε για τη βελτιωμένη του ικανότητα να μοντελοποιεί απευθείας την αβεβαιότητα και να μαθαίνει αποτελεσματικά από σπάνια δεδομένα. Η δύναμη του DeepProbLog βρίσκεται στην ικανότητά του να συλλογίζεται πάνω σε υψηλού επιπέδου έννοιες και σχέσεις χρησιμοποιώντας πιθανοτικό λογικό προγραμματισμό, σε συνδυασμό με την επάρκειά του στο χειρισμό σπάνιων δεδομένων μέσω πιθανοτικού συλλογισμού. Αυτό το πλαίσιο επιτρέπει τον δηλωτικό ορισμό πολύπλοκων μοτίβων συμβάντων, διευκολύνοντας την περίπλοκη CER μέσω του άμεσου μοντελοποιητικού και συλλογιστικού εργαστηρίου πάνω σε διάφορες αναπαραστάσεις δεδομένων. Αυτή η διατριβή παρουσιάζει μια λεπτομερή αξιολόγηση του DeepProbCEP, μιας επέκτασης του DeepProbLog, σε διάφορες εργασίες CER στο σύνολο δεδομένων MNIST. Περιλαμβάνει μια λεπτομερή συγκριτική ανάλυση έναντι μοντέλων που βασίζονται αποκλειστικά σε νευρικές ή συμβολικές προσεγγίσεις, επισημαίνοντας τα εγγενή τους πλεονεκτήματα και περιορισμούς. Η έρευνα προσφέρει πολύτιμες πληροφορίες για πιθανές μελλοντικές προόδους στη CER, επικεντρωμένες στη χρήση του DeepProbCEP. Ενώ το κύριο επίκεντρο αυτής της έρευνας είναι το DeepProbCEP λόγω των ικανοτήτων του στη μοντελοποίηση της αβεβαιότητας και της μάθησης από σπάνια δεδομένα, αναγνωρίζεται επίσης το δυναμικό του Neuroplex εντός του τομέα CER. Αυτή η μελέτη συμβάλλει στον τομέα όχι μόνο επιβεβαιώνοντας την υπόσχεση του DeepProbCEP ως πλαισίου για την CER αλλά και θέτοντας τα θεμέλια για μελλοντικές εξερευνήσεις και προόδους στη νευρο-συμβολική μάθηση και συλλογιστική.

Abstract

This research delves into the intersection of neural networks and symbolic reasoning, particularly focusing on the application of neural-symbolic learning and reasoning in Complex Event Recognition (CER). Central to this study is the exploration of DeepProbLog, a cutting-edge neural-symbolic framework. DeepProbLog distinguishes itself by adeptly integrating logical expressiveness with the statistical strength of neural networks and probabilistic programming. While the Neuroplex framework also offers a robust blend of neural networks and logical programming, DeepProbLog was chosen for its enhanced ability to model uncertainty directly and to learn efficiently from sparse data. The strength of DeepProbLog lies in its ability to reason over high-level concepts and relationships using probabilistic logic programming, combined with its proficiency in handling sparse data through probabilistic reasoning. This framework enables the declarative definition of complex event patterns, facilitating nuanced CER by directly modeling and reasoning across diverse data representations. This thesis presents a thorough evaluation of DeepProbCEP, an extension of DeepProbLog, across various CER tasks in MNIST dataset. It includes a detailed comparative analysis against models rooted exclusively in either neural or symbolic approaches, highlighting their intrinsic strengths and limitations. The research offers valuable insights into potential future advancements in CER, focusing on the utilization of DeepProbCEP. While the primary focus of this research is on DeepProbCEP due to its capabilities in modeling uncertainty and learning from sparse data, the potential of Neuroplex within the CER domain is also acknowledged. This study contributes to the field by not only substantiating the promise of DeepProbCEP as a framework for CER but also by setting a foundation for future explorations and advancements in neural-symbolic learning and reasoning.

Keywords: Neural-Symbolic Learning, Complex Event Recognition, DeepProbLog, DeepProbCEP, Neural Networks, Symbolic Inference, Temporal Reasoning, Knowledge Representation, Deep Learning, MNIST

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Complex Event Recognition

The notion of event processing has been widely recognised as a universal computational paradigm across diverse domains of application. Events reports provide information on the alterations in the condition of a system and its surrounding environment. Complex Event Recognition (CER) pertains to the detection and categorisation of complex events that are composed of simple, derived events meeting specific patterns. This process enables the implementation of responsive and anticipatory actions [**Giatrakos et al.**, **2020**, 19].

Complex Event Recongition pertains to the process of identifying patterns within streams of event data that are continuously being received from various distributed sources, including those with geographical diversity. CER plays a crucial role in numerous modern Big Data applications, where the processing of event streams is necessary to acquire timely insights and execute responsive and anticipatory actions. Instances of such applications encompass the identification and prediction of assaults in computer network nodes, human activities depicted in video content, emerging narratives and patterns on the Social Web, traffic and transportation incidents within intelligent urban areas, error conditions within intelligent energy grids, breaches of maritime regulations, cardiac arrhythmias, and the propagation of epidemics. Within any application, the utilization of Complex Event Processing (CEP) enables the interpretation of real-time data, subsequent appropriate responses, and the formulation of proactive counter-measures [**Alevizos & Artikis**, **2021**, 2].

Complex event recognition systems rely on a continuous flow of timestamped "simple, derived events" (SDEs) as their input in order to identify and acknowledge complex events (CEs). Complex events (CEs) refer to groupings of simple derived events (SDEs) that conform to specific patterns. The objective of the CER is to promptly identify significant occurrences inside a real-time context and promptly react to them. The specification of complex event patterns is performed by the user within data streams. Numerous Complex event recognition (CER) methods and programming languages have been put out thus far. All systems share a common objective, however they exhibit variations in their architectures, data structures, pattern languages, and processing algorithms.

The main categories distinguished in most papers are the following:

- **Automata-based Systems**
- **Logic-based Systems**
- **Tree-based Systems**
- **Hybrid approaches**

1.1.1 Automata-based Systems

Given that a significant number of CER engines utilize finite automata, namely deterministic (DFA) or non-deterministic (NFA) ones, it is unsurprising that automata represent a prominent methodology for managing uncertainty. Automata are widely utilized as computational models for Complex Event Recognition (CER) systems [**Alevizos et al.**, **2015**, 1]. Typically, these automata are formulated using a language similar to SQL, using supplementary operators to handle the regular aspects of the pattern. The patterns are converted into an automaton, which is typically non-deterministic. The automaton is subsequently provided with a sequence of SDEs, and its state is altered depending on whether the conditions on the outgoing transitions from the present state are fulfilled. The responsible SDE may be dismissed if deemed irrelevant or stored if deemed significant when a transition is initiated.

Numerous CER systems offer users a pattern language, which is subsequently transformed into an automaton through a compilation process. The automaton model is commonly employed for the purpose of establishing the semantics of a language and/or serving as a framework for executing pattern recognition tasks. Automata-based Complex Event Recognition (CER) systems encompass various notable examples, such as Cayuga, SASE, and SASE+. These systems employ automata for both event detection and event correlation.Additionally, TESLA is another prominent CER system that utilizes automata specifically for pattern recognition [**Artikis et al.**, **2017**, 4].

The SASE is a computational framework utilized to analyze patterns within streams of data. In this model, each event within the stream represents a specific occurrence of interest, characterized by a timestamp and additional properties. These input events are consolidated into a unified stream, arranged in chronological order based on their respective timestamps. A pattern is a systematic arrangement of events that occur in a sequential manner and fulfill all relevant temporal and other criteria.

As mentionted by Alevizos et al. (2015)[1], Kawashima et al. (2010) [22] offered a straightforward approach to address the issue of uncertainty in automata, building upon the SASE+ event processing engine. The system constructs a deterministic automaton for each user query, specifically for the purpose of defining a concept in computer engineering. It then identifies patterns that surpass a predetermined level of confidence by creating a matching tree as fresh SDEs are received. This process continues until the time window for the query elapses. In order to optimize the process, the branches of the tree that fall below the specified threshold are pruned at an early stage. It is assumed that the SDEs are independent, resulting in the calculation of probability values through multiplication. Additionally, each SDE is associated with a certain occurrence probability. The assignment of probability values to event properties or searches is not permitted. The throughput rates can attain many hundreds of events per second.However, it is important to note that these results are derived from studies involving a single query of moderate complexity. Specifically, the query involves a sequence operator with equality selection on the attributes and does not involve any shared variables [**Alevizos et al.**, **2015**, 1].

Shen et al. (2008)[46] provide an alternative method that utilizes an NFA-based technique to integrate uncertainty into a pre-existing CER system. The present study builds upon the SASE+ framework and introduces modifications to accommodate probabilistic SDEs. An SDE is characterized by a collection of options, each accompanied by its corresponding probability of occurrence. The probabilities associated with all the options in the SDE add up to a value of 1, or less than 1 if the possibility of non-occurrence is taken into account. The probability space is defined over the set of possible worlds, which are determined by the many mutually exclusive alternatives of the SDEs. The definitions of CE are represented as NFAs. However, to prevent the need for enumerating all potential scenarios, a specialized data structure known as the Active Instance Graph (AIG) is employed. The Active Instance Graph is a type of Directed Acyclic Graph that establishes connections between events and their preceding complex events. These events are those

whose potential existence could potentially result in the identification of the Complex Event (CE). By traversing the AIG in reverse order, it is possible to get the sequence(s) that meet the CE definition. Additionally, this structure enables the dynamic filtering of events in the presence of other constraints, apart from temporal sequence. Each event is ultimately linked to its lineage, which refers to a function that captures the origin or source of the event. This lineage function is utilized to calculate the likelihood of the occurrence [**Alevizatos et al.**, **2015**, 1].

Another well-known methodology in the field of Big Data processing is FlinkCEP. Flink is a computational framework designed for the purpose of real-time data processing. Additionally, FlinkCEP is a specialized library that provides support for Complex Event Processing (CEP) within the Flink framework. This approach is founded on automata, aligning with the principles of SASE. FlinkCEP is a robust industrial solution that offers a wide range of capabilities, enhanced flexibility, and inherent scalability.

Figure 1. Automata Based Systems (Semantic Scholars)

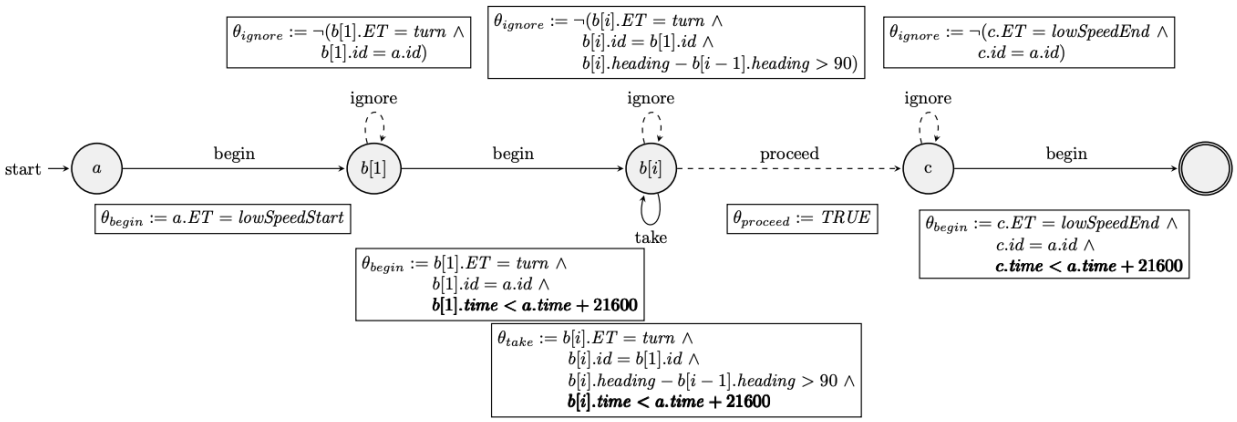


Figure 1, shows a finite state machine (FSM), a mathematical model of computation commonly used to design both computer programs and sequential logic circuits. An FSM is defined by a list of its states, its initial state, and the conditions for each transition. The states of the FSM are represented as nodes or circles, and the transitions as arrows that connect these states. The FSM in the diagram comprises the following components:

1. **States**: The nodes labeled 'a', 'b[1]', and 'c' represent the discrete states within the system. Each state corresponds to a certain configuration of the system's components or to a particular step in its process.

2. **Transitions**: The directed arrows symbolize the transitions from one state to another. These transitions can be triggered by specific events or conditions, dictating the flow of control from state to state.

3. **Events/Conditions**: These are specified within the diagram as labels on the transitions or within rectangles adjacent to them. For instance, $\theta begin := a.ET = lowSpeedStart$ denotes a condition that must be satisfied for the initial transition to state 'a' to occur. The use of 'θ' (theta) suggests a convention or a particular syntax used within the scope of the system's description.

4. **Temporal Conditions**: Certain transitions are governed by time constraints, such as $b[1].time < a.time + 21600$, indicating the real-time or relative timing aspects integral to the system's functionality

5. **Start State**: The state where the system begins its operation is indicated by an incoming arrow not connected to any other state. In this case, the start state points to state 'a'.

6. **Ignore Conditions**: Represented by dashed lines leading to diamond shapes with the label 'ignore', these transitions are conditions that the system must recognize but not act upon. This mechanism is typically employed to handle exceptions or specific non-normative behaviors.

Figure 1, serves as an abstraction to model the behavior of the system under consideration. It encapsulates the various operational states of the system, the events that cause a transition from one state to another, and the conditions under which such transitions are valid or should be ignored. The temporal aspects included in the diagram suggest that the system's transitions are also time-dependent, adding a layer of complexity to the FSM. This FSM is part of a broader system that likely includes more states and transitions, which are essential for a complete understanding of the modeled process.

1.1.2 Logic-based Systems

Contemporary organisations necessitate methodologies for the automated conversion of the substantial quantities of data they amass during their activities into actionable information. The fulfilment of this criteria can be achieved by the utilisation of event recognition systems, which are capable of identifying actions or occurrences that hold particular importance inside an organisation. These systems operate by analysing streams of "low-level" information that are typically challenging for people to effectively utilise. A plethora of event recognition systems have been proposed in the academic literature [**Artikis et al.**, **2010**, 5]. Recognition systems that utilise a logic-based representation of event structures have garnered significant interest. These systems are particularly appealing due to their formal and declarative semantics, as well as their demonstrated efficiency and scalability. Furthermore, the construction and refinement of event structures in these systems can be automated using machine learning tools [**Artikis et al.**, **2012**, 6].

Logic-based temporal formalism is a significant area of study. Patterns often take the form of rules, consisting of a head and a body that define the conditions under which a CE can be detected. The mechanism used to perform inference can vary, with some Prolog-based systems use Selective Linear Definite (SLD) resolution, while others utilise directed graphs that are formed from the rules, resembling automata. An instance of Logic-based Systems is exemplified by the Chronicle Recognition System (CRS). A Chronicle, in the context of computer science, is a computational entity that consists of a series of events that are connected by temporal constraints. It can be understood as a logical construct, comprising of a body and a head. The CRS patterns are represented in the form of a graph, where each node corresponds to an event and the edges encode the temporal constraints. This graph is sometimes referred to as a Temporal Constraint Network (TCN). This functionality allows CRS to perform pattern consistency checks and optimisations by propagating constraints throughout the graph or eliminating unnecessary constraints. The utilisation of Colored Petri Nets can potentially provide a semantic framework for the CRS language. The runtime behaviour of the CRS system bears resemblance to automata-based systems, since it involves the constant creation and termination of TCN instances based on the fulfilment of their future event needs.

Lastly, we present RTEC (Event Calculus for Run-Time Reasoning), a CER engine that utilises Event Calculus and is implemented in the Prolog programming language. The Event Calculus is a logic programming action language that facilitates reasoning about events and their effects [**Miller & Shanahan**, **1999**, 36]. RTEC is an implementation of the Event Calculus designed specifically for event streams. It incorporates a windowing system, as well as caching and indexing methods, to enhance the efficiency of reasoning processes. RTEC patterns are often defined

using rules denoted as ”initiatedAt” and ”terminatedAt,” which establish the beginning and end points, respectively, of a complex event (CE). The RTEC algorithm will identify all instances in which the withinArea function is initiated inside a specified time window. It will next determine the timepoints at which the function is terminated. Finally, it will calculate the maximum intervals by pairing the beginning and terminating timepoints. In essence, the concept of Real-Time Event Calculus (RTEC) posits that composite activities adhere to the principle of inertia, meaning that a Complex Event/Fluent (CE/fluent) persists unless explicitly terminated. It is important to highlight that RTEC does not make any assumptions regarding the temporal distance between the initial and final positions, which may be located in separate time intervals. RTEC provides the full range of expressive capabilities found in logic programming, enabling it to effectively handle a wide range of constraints, relational CEs, and background information, among other elements.

Consider the provided RTEC rule:

$$\begin{aligned} &happensAt(entersArea(VesselId, Area), T), \\ &\quad typeOf(Area, AreaType). \\ &terminatedAt(withinArea(VesselId) = \_, T) \leftarrow \\ &\quad happensAt(exitsArea(VesselId, Area), T). \end{aligned} \tag{1.1}$$

Lastly, we present RTEC (Event Calculus for Run-Time Reasoning), a CER engine that utilises Event Calculus and is implemented in the Prolog programming language. The Event Calculus is a logic programming action language that facilitates reasoning about events and their effects [**Miller & Shanahan**, **1999**, 36]. RTEC is an implementation of the Event Calculus designed specifically for event streams. It incorporates a windowing system, as well as caching and indexing methods, to enhance the efficiency of reasoning processes. RTEC patterns are often defined using rules denoted as ”**initiatedAt**” and ”**terminatedAt**”, which establish the beginning and end points, respectively, of a complex event (CE). The RTEC algorithm will identify all instances in which the withinArea function is initiated inside a specified time window. It will next determine the timepoints at which the function is terminated. Finally, it will calculate the maximum intervals by pairing the beginning and terminating timepoints. In essence, the concept of Real-Time Event Calculus (RTEC) posits that composite activities adhere to the principle of inertia, meaning that a Complex Event/Fluent (CE/fluent) persists unless explicitly terminated. It is important to highlight that RTEC does not make any assumptions regarding the temporal distance between the initial and final positions, which may be located in separate time intervals. RTEC provides the full range of expressive capabilities found in logic programming, enabling it to effectively handle a wide range of constraints, relational CEs, and background information, among other elements.

This rule exemplifies the system’s power, defining the *withinArea* complex event to identify instances where vessels operate within specific zones. Notably, RTEC’s robustness stems from its ability to handle a myriad of constraints, integrate relational complex events, and utilize background knowledge, underlining the rich expressivity of logic programming.

Logic-based systems, with their reliance on rule-based patterns and logical inferences, offer a dynamic avenue for complex event recognition. As CER continues to evolve, the flexibility and reasoning capabilities of these systems remain integral to the field.

1.1.3 Tree-based Systems

A further domain of investigation in the field of CER involves the utilisation of trees as a computational framework, with ZStream serving as the preeminent illustration. From a syntactic perspective, it is noteworthy to mention that ZStream bears a striking resemblance to SASE, exhibiting nearly identical syntax in the majority of instances. The ZStream model can be distinguished from

automata-based models by its assumption that complex events (CEs) have a durative nature. A sequence operator is considered met in the context of CEs when the end timepoint of one CE is less than the start timepoint of the subsequent CE in the series. ZStream is able to circumvent semantic ambiguities that arise in the presence of event hierarchies [**Mei & Madden**, **2009**, **Mei2009**]. Consider the patterns:

$$R1 := a; b$$

$$R2 := R1; c$$

Here, if R1 acquires the timestamp of its final event, b, this can create ambiguities when R2 is translated to an automaton. ZStream resolves this by treating the CEs in R1 as durative events, preserving the integrity of the sequence.

E-Cube is an alternative CER system that employs tree topologies, albeit with a different approach compared to ZStream. With the exclusion of iteration, E-Cube demonstrates support for the majority of the standard CER operators in terms of their expressive capabilities. The temporal model of SDEs is characterised by intervals, rendering them the sole entities that possess instantaneous properties. The work offered lacks clarity in its consumption and selection policies, making them difficult to ascertain without ambiguity. Based on the definition of the sequence operator, it can be inferred that skip-till-any-match and reuse are the most probable subsequent actions. One of the key advantages of E-Cube is in its capacity for multi-query optimisations, enabling the evaluation of numerous patterns while minimising redundant and duplicate computations in cases when these patterns exhibit structural similarities. This is achieved through the implementation of an event pattern query hierarchy, which facilitates the sharing of subpatterns and effectively eliminates repetition.

Additionally, the system is equipped with a cost-driven optimiser that is capable of generating an ideal plan by prioritising the identification of a plan that maximises the reuse of intermediate results. The E-Cube also possesses elastic properties, which holds significance. When the occurrence of a drift is detected, the system consistently gathers data regarding the stream and is capable of dynamically adjusting its execution plan in real-time [**Liu et al.**, **2010**, 37].

Figure 2 illustrates an expansive decision tree, a fundamental structure employed in computational decision-making processes. The decision tree is a graphical representation of a multi-step decision process, where each internal node represents a ”test” on an attribute (e.g., whether a coin flip comes up heads or tails), each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represent classification rules. In this decision tree:

1. **Internal Nodes**: These are test nodes that check the value of a certain attribute and branch accordingly.

2. **Leaf Nodes**: Terminal nodes that represent the outcome of the decision process. In this tree, they are likely to be the possible decisions or final classifications made by the algorithm.

3. **Branches**: Represent the decision rules or criteria that guide the flow from one node to another.

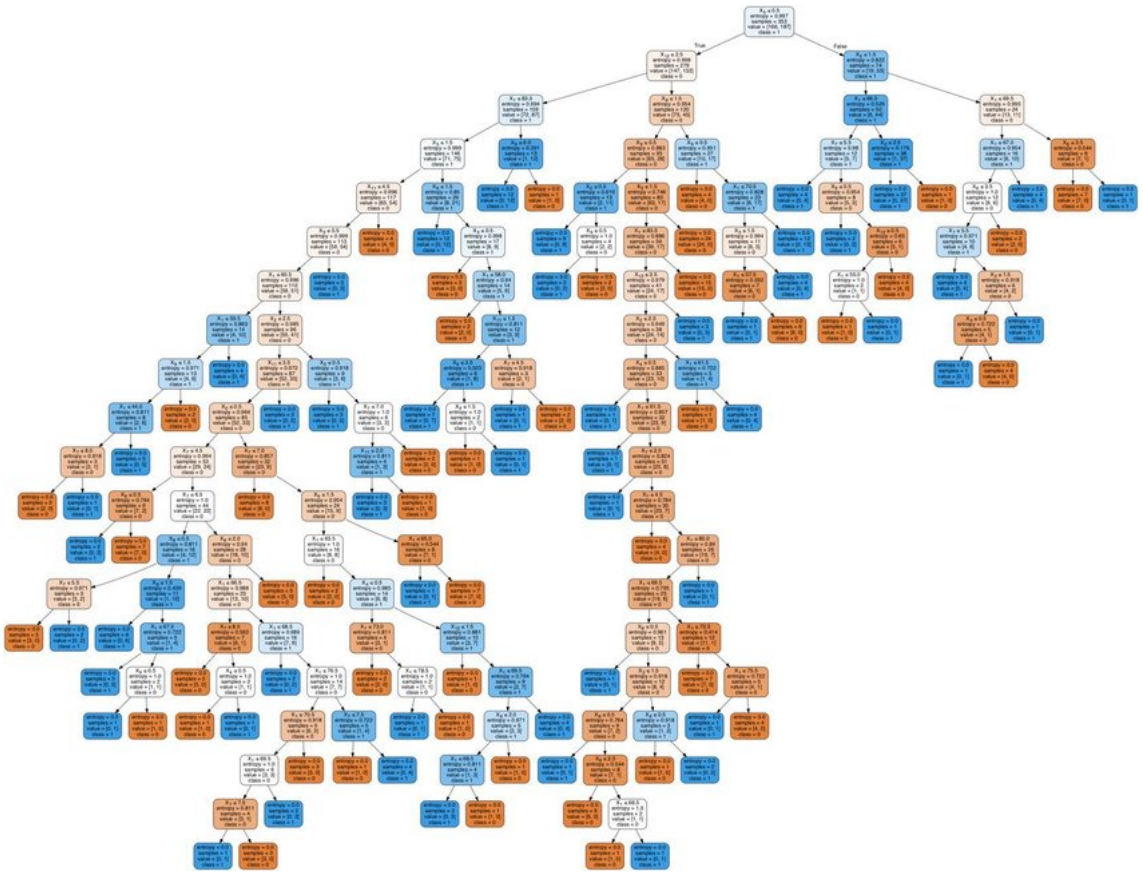
The decision tree in Figure 2 employs a binary decision-making process, as evidenced by the two branches that stem from each internal node. Each branch represents a binary decision outcome, leading to further subdivision or to a final decision at the leaf nodes. The color coding—blue and orange—could indicate different types of decision nodes or outcomes, possibly representing a categorization within the decision-making process.

The structure of the tree suggests a thorough analysis of a dataset or scenario, where every possible outcome has been meticulously mapped. The exhaustive nature of the tree is indicative

of a system that requires a comprehensive set of decisions, such as complex diagnostic systems, intricate strategy games, or sophisticated predictive models that take into account numerous variables.

This decision tree could be a crucial component of a decision support system, where the aim is to replicate human decision-making and provide automated recommendations or decisions. The level of detail within the tree highlights the algorithm's capability to handle a multitude of scenarios, ensuring robustness and accuracy in its predictive power.

Figure 2. Tree Based Systems (Towards Al)



1.1.4 Hybrid Approaches

Trees play a crucial role in the core operation of Esper, serving as a means for filtering, windowing, and aggregating data. Esper, however, poses challenges in terms of categorisation due to its utilisation of non-deterministic automata in its pattern matching functionality for recognising regular patterns [**Anicic et al.**, **2010**, 3]. This phenomenon is indicative of a broader trend. Several temporal techniques rely on Allen's interval algebra. The expressive nature of Esper patterns arises from their utilisation of a combination of trees, automata, and logic. However, the consequences regarding semantics, soundness, and completeness of these patterns have yet to be determined.

The T-Rex system is a Complex Event Recognition (CER) system that integrates logic-based rules and automata, utilising TESLA as its event specification language. The T-REX system serves as an intermediary between automata and logic, as it does not solely rely on logical principles. T-Rex is a novel CEP middleware that effectively integrates both expressive capabilities and efficiency. One aspect of this approach involves the utilisation of a language known as TESLA, which has been specifically designed to facilitate the clear and seamless description of composite events. In contrast, the proposed approach offers a proficient event detection method that relies on automata for the interpretation of TESLA rules [**Cugola & Margara**, **2012**, 11].

The patterns, which are expressed in TESLA and exhibit a syntax that can be likened to SASE, are further transformed into automata and afterwards assessed on an event stream. In contrast, it is possible to convert TESLA patterns into TRIO formulas. TRIO formulas are expressions in a first-order logical language that incorporates temporal operators and provides clear semantics based on metric temporal logic.

Figure 3, illustrates a temporal progression of a system's states or events through a series of discrete time steps, labeled from $t = 0$ to $t = 8$. Each horizontal layer represents the state of the system at a specific point in time, denoting the transitions between different states or the occurrence of events. This diagram serves as an effective visual representation of dynamic behavior in systems where the sequence and timing of events are of paramount importance.
The elements of the diagram are as follows:

1. **Initial State** (t = 0): The system begins in state 'Aut', signifying the start of the process.

2. **Events and States**: The diagram includes a series of nodes marked 'E' and 'B', which could indicate different types of events or states the system can be in. These nodes are connected by arrows suggesting permissible transitions.

3. **Time-Dependent Transitions**: At each time step, the system experiences transitions, which are represented by labeled arrows. For example, at $t = 1$, the transition is labeled 'Aut1', leading to 'E1@1'.

4. **Concurrent States**: There are instances where the system appears to be in multiple states at once, such as at $t = 5$, where 'Aut2' and 'Aut21' occur simultaneously.

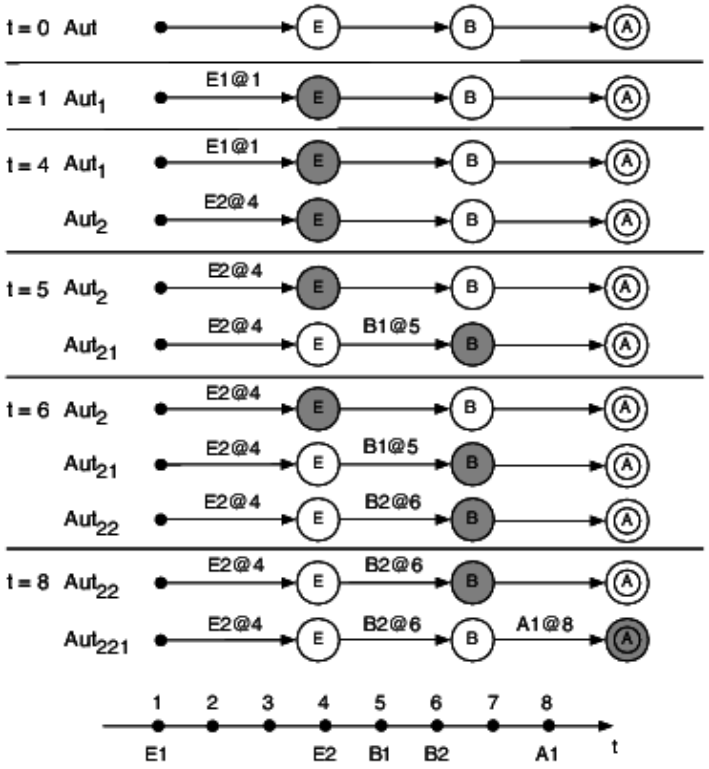
5. **State Evolution**: Over time, the system evolves through various states, as indicated by the change in labeling of the states and events, such as 'E2@4', 'B1@5', and 'A1@8'.

6. **Temporal Annotations**: The labels along the transitions, such as 'E1@1' and 'E2@4', indicate specific events occurring at particular time steps.

7. **Timeline**: The timeline at the bottom of the diagram provides a linear representation of time, with events such as 'E1', 'E2', 'B1', 'B2', and 'A1' mapped to their corresponding time steps.

This diagrammatic representation is particularly useful in understanding the complex behavior of systems in which operations must adhere to strict timing constraints. Such a visual tool is invaluable for the analysis and verification of time-critical processes, such as in real-time computing systems, automated control systems, or sequence-driven algorithmic procedures.

Figure 3. Tesla Language



1.2 Neural-Symbolic Learning and Reasoning

1.2.1 Overview

The intricate study of human behavior spans across a multitude of disciplines, integrating insights from computer science, artificial intelligence (AI), neural computation, cognitive science, philosophy, and psychology. At its core, the premise holds that behavior is not only influenced but also directed by underlying cognitive processes and mental states. Within this interdisciplinary nexus, two principal frameworks are employed to model and understand behavior: computational-logic systems and connectionist models.

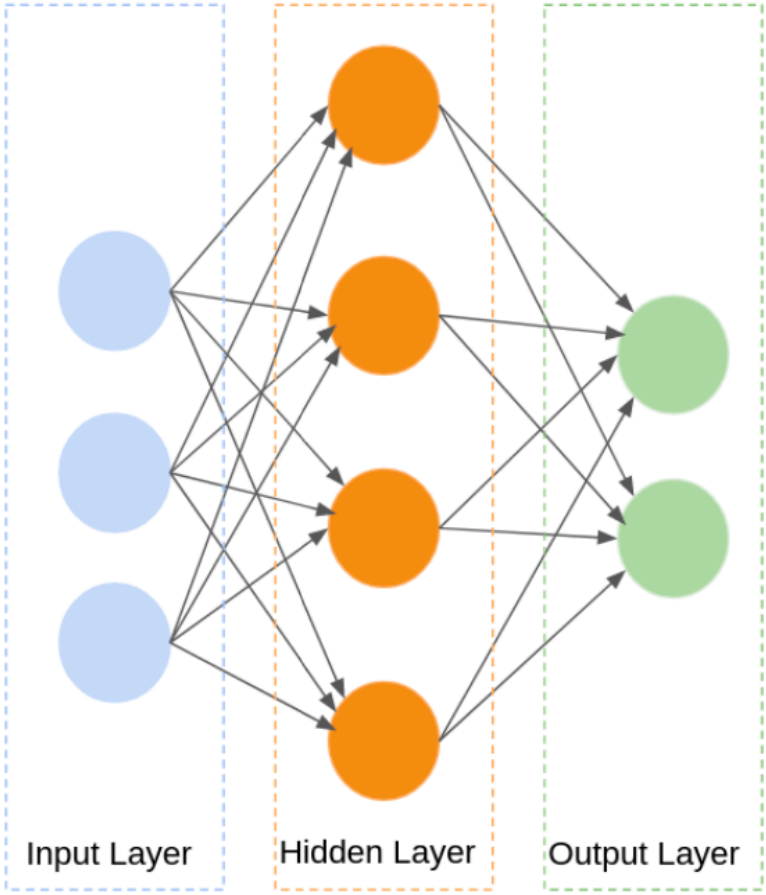
Computational-logic systems are analytical tools that facilitate high-level reasoning and complex thought processes. They are grounded in various logics, including classical, non-monotonic, modal, and temporal logic, each providing a scaffold for structured reasoning and decision-making. These systems excel at explicit rule-based processing and symbolic representation, which are essential in formal reasoning tasks.

Conversely, connectionist models, or neural networks, capture the lower-level dynamics and emergent phenomena that characterize cognitive and neural activities. These models are inspired by the architecture and functionality of the brain, emulating the distributed and parallel processing of information. They are comprised of interconnected units or 'neurons' that work collectively to process and pattern information, making them adept at handling tasks such as perception, pattern recognition, and associative memory.

Neural networks are organized into three distinct types of layers, as shown in Figure 4, each with a specific role in the data processing pipeline:

- **Input Layer**: This layer serves as the gateway to the neural network, receiving raw data inputs and passing them forward. It is the initial contact point for the external environment, setting the stage for subsequent processing.
- **Hidden Layer(s)**: One or more hidden layers may reside within a neural network. These layers are the computational workhorses, engaging in intricate data transformations through mathematical functions. The complexity and depth of a neural network are often attributed to the number and architecture of its hidden layers.
- **Output Layer**: This layer culminates the neural processing journey, delivering the network's final output. It synthesizes the learned representations from the hidden layers into a coherent prediction or classification, ready to be interpreted in the context of the problem at hand.

Figure 4. Schematic Representation of Neural Networks

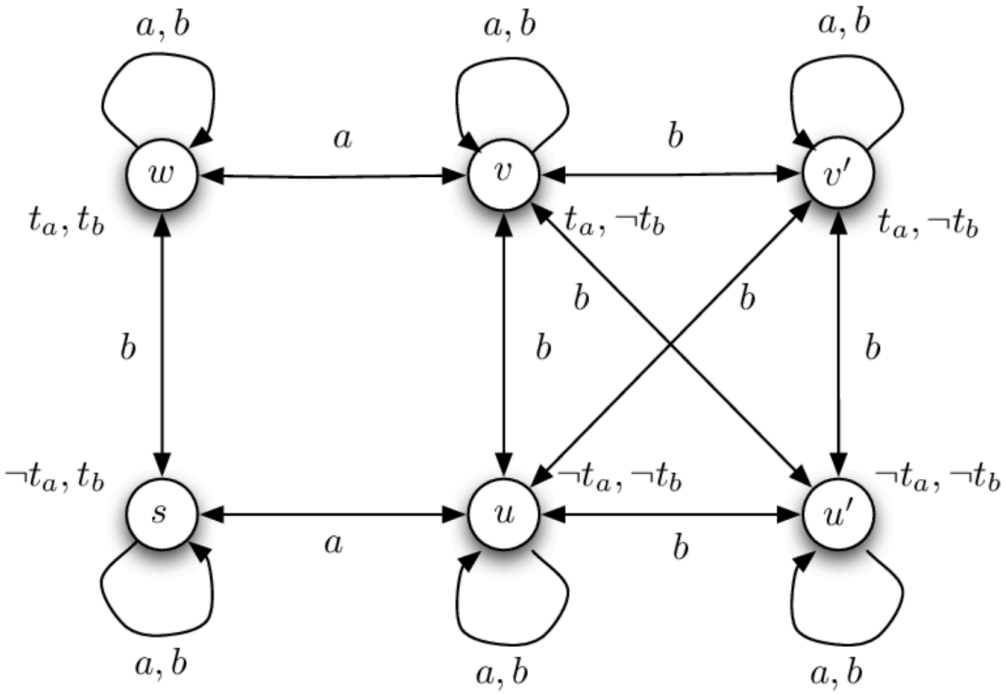


The interplay between computational-logic systems and connectionist models is pivotal in neural-symbolic learning and reasoning. By integrating the symbolic reasoning capabilities of the former with the adaptive learning abilities of the latter, a more holistic and robust approach to behavior modeling can be realized. This synergy aims to harness the strengths of both paradigms, facilitating a deeper understanding of the cognitive underpinnings of behavior and improving the efficacy of AI systems in complex, real-world tasks.

In the field of cognitive science, artificial intelligence, and psychology, there have been several computational cognitive models developed to understand processes related to thinking, learning, and language [**Pinker**, **Nowak**, **Lee**, **2008; Shastri**, **2007; Sun**, **2009**, **200**].The applications in which such systems have demonstrated potential include fault detection, computational biology, training and assessment in simulators, and software verification [**de Penning**, **d'Avila Garcez**, **Lamb**, **Meyer**, **2010**]. In addition, there have been significant advancements in the field of computer science that have led to the creation of cognitive computational systems, which integrate machine learning and automated reasoning techniques [**Garcez**, **Broda**, **Gabbay**, **2002; Garcez**, **Lamb**, **Gabbay**, **2009; Valiant**, **2000**]. An essential aspect of a connectionist computational theory of the mind is its ability to emulate the parallelism and adaptive learning processes observed in neural networks. These neural networks are widely recognised for their crucial role in ensuring the system's resilience and overall efficacy in handling common knowledge. According to [**Valiant (2008)**], a purely symbolic approach would be inadequate. In contrast, logic has been generally recognised as a fundamental tool in the representation of cognitive processes and behavioural patterns [**Kowalski**, **2011; Pereira**, **2012**]. It is commonly referred to as the "calculus of

computer science" by a significant number of scholars. Nonclassical logics often assume a significant role within this particular environment. Temporal logic has made significant contributions in both academic and industrial settings [**Pnueli**, **1977**]. Additionally, modal logics have emerged as a widely used language for the purpose of specifying and analysing knowledge and communication in multi-agent and distributed systems, among other applications [**Fagin**, **Halpern**, **Moses**, **Vardi**, **1995**]. The investigation of practical reasoning in the field of artificial intelligence has been mostly focused on non-monotonic formalisms. Intuitionistic logic has been identified as a suitable logical foundation for various fundamental domains within theoretical computer science, such as type theory and functional programming [**Van Dalen**, **2002**]. In the realm of semantic web research, description logics, which bear resemblance to Kripke models (e.g. Figure 5), have proven to be valuable [**Baader**, **Calvanese**, **McGuinness**, **Nardi**, **Patel-Schneider**, **2003**].

Figure 5. Kripke Model Example



When developing models that incorporate both learning and reasoning, it is necessary to harmonise the methodologies employed in many disciplines, particularly statistics and logic. This enables the utilisation of their respective strengths while circumventing their inherent limitations and drawbacks. The technique of neuro-symbolic systems aims to facilitate the transfer of concepts and mechanisms from logic-based computation, which is frequently nonclassical in nature, to the realm of neural computation.

1.2.2 State of the art

The predominant effort in the field of neural-symbolic learning and reasoning has been directed towards propositional logics. The initial methodologies mostly relied on the connectionist portrayal of propositional logic, a field of study that has subsequently undergone significant expansion to encompass additional finite logics [**Garcez et al.**, **2015**, 17]. Several primary proposals have been put forth in the literature to address the issue of propositional fixation in neural networks. One such proposal, as suggested by Gust, Kühnberger, and Geibel (2007), involves the utilisation of variable-free representations of predicate logic using category-theoretic Topos. Another proposal, presented by Bader, Hitzler, and Hölldobler (2008), focuses on predicate Horn logic programs with function symbols and employs an encoding of logic as vectors of real numbers mediated by the Cantor set. Additionally, Guillame-Bert, Broda, and d'Avila Garcez proposed a method in 2010 for learning first-order rules based on term encoding, also represented as vectors. These technologies have demonstrated efficacy in constrained proof-of-concept environments or small-scale instances, but efforts to attain practical utility have thus far yielded unsatisfactory results. In order to make progress, it may be necessary to consider logics with intermediate expressiveness, such as description logics. Additionally, propositionalization methods, as utilised by ILP [**França**, **Zaverucha**, **and d'Avila Garcez**, **2014**, 16], and answer-set programming [**Lifschitz**, **2002**, 28] should be taken into account. Furthermore, modal logics are known to be more expressive than propositional logic and are decidable. The viability of integrating description logics and rules in the context of neural-symbolic integration has been demonstrated by more recent studies [**Krötzsch et al.**, **2011**, 25].

The issue of variable binding and the manner in which neural networks engage in reasoning with variables continue to be fundamental aspects of this endeavour. Considerable progress has been made within the field of neural-symbolic computation in extracting logical expressions from neural networks that have undergone training [**Garcez et al.**, **2015**, 17]. This extracted knowledge is then utilised to initiate learning in subsequent challenges. In recent studies, there has been compelling evidence indicating that neural networks possess the capability to acquire and comprehend complete sequences of events, effectively resembling a form of "mental simulation" of specific, prolonged activities. Additionally, a highly sophisticated logical framework for action has been established, such as the fundamental propositional logic of programs (PDL) proposed by Harel, Kozen, and Tiuryn (2001). This framework effectively represents the truth conditions resulting from different combinations of actions. One potential avenue for further investigation is to examine the extraction process from a trained network that demonstrates the aforementioned simulated behaviour. This could involve utilising a PDL expression to succinctly capture a high-level description of the sequence of operations [**Garcez et al.**, **2015**, 17].

According to Feldman (2006) [15], the brain's composition as a network of neurons that engage in actions, rather than merely representing objects, suggests that action models should assume a pivotal position. When it comes to the representation of information in the brain, a significant obstacle is in comprehending the mechanisms by which neuronal activations, which are extensively distributed and lack symbolic attributes, lead to behaviour's that exhibit symbolism, such as language and logical reasoning. The recent advancements in functional magnetic resonance imaging (fMRI) and magnetoencephalography (MEG) analysis have facilitated the opportunity to formulate and evaluate these hypotheses. For example, the application of formal concept analysis enables the identification and description of semantic structures within the brain. Additionally, the utilisation of conceptual attribute representations [**Binder and Desai**, **2011**, 8] allows for the modeling of the mapping between semantic concepts and specific regions of the brain. One of the primary obstacles that lies ahead is comprehending the mechanisms via which semantics are formulated and influenced by contextual factors, such as the arrangement of words inside a sentence [**Garcez et al.**, **2015**, 17].

1.3 Motivation of this Thesis

The quest to merge learning with reasoning stands as a cornerstone challenge in the field of Artificial Intelligence (AI), specifically demanding a harmonious integration of symbolic inference and statistical learning methods. Historically, AI research has experienced a dichotomy: one faction focusing on statistical learning and the other on symbolic reasoning, with few intersections between the two. Recent advancements in deep learning have reignited interest in this area, particularly in understanding and enhancing representations at varying levels of abstraction.

Neural-symbolic learning and reasoning, a field of growing relevance for over two decades, addresses these representational challenges, including critical aspects like the binding problem. This domain aspires to combine the adaptability and learning efficiency of neural networks with the structured, rule-based approach of logical reasoning. The application of this hybrid approach is particularly compelling in Complex Event Recognition (CER), where identifying patterns within temporal sensor data necessitates both perceptual learning and high-level temporal reasoning.

This thesis is motivated by the objective to critically evaluate neural-symbolic techniques within the scope of CER. The focus is on the DeepProbCEP framework [**Vilamala**, **2022**, 50], an extension of DeepProbLog [**Manhaeve et al.**, **2021**, 33], renowned for its unique blend of probabilistic logic programming and neural network capabilities. DeepProbCEP stands out for its direct modeling of uncertainty and efficient learning from sparse data, making it an ideal candidate for complex event detection and analysis.

While Neuroplex, another neural-symbolic framework, offers intriguing insights, its application in CER was studied for comparative understanding rather than direct experimentation. The primary investigation centered around the utilization of DeepProbCEP, examining its performance in predictive accuracy and scalability across different CER scenarios in MNIST dataset. This research aims to delineate the capabilities and limitations of neural-symbolic approaches, particularly highlighting how DeepProbCEP contributes to advancing state-of-the-art in CER.

DeepProbCEP's integration of probabilistic reasoning with neural networks allows for sophisticated handling of uncertainty and varied data representations, key in CER tasks. This thesis presents a detailed analysis of DeepProbCEP's application in identifying complex events from simple event data, showcasing its potency in bridging the gap between low-level data processing and high-level reasoning.

The research contributes significantly to the field by showcasing the practical applications and effectiveness of DeepProbCEP in neural-symbolic learning and reasoning, setting a foundation for future explorations in this promising domain.

Chapter 2

Background

2.1 Literature Review

Machine learning and deep learning are distinct branches within the science of artificial intelligence. Machine learning enables the development of models with basic perceptual abilities like classification and prediction. On the other hand, higher-level inferences and information extraction are accomplished through reasoning using suitable representations and logic-based reasoning techniques. The integration of neural-based learning and logic-based reasoning has the potential to facilitate the development of novel systems capable of comprehending their surroundings and deriving inferences from the provided data [**Oikonomakis**, **2023**, 39].

In their 2021 paper, Manhaave et al. present DeepProbLog, a programming language that combines neural networks and probabilistic logic. This language includes neural predicates to include deep learning. The researchers demonstrate the adaptability of existing inference and learning techniques from the underlying probabilistic logic programming language ProbLog to the new language. The authors provide theoretical and practical evidence that DeepProbLog is capable of handling both symbolic and sub-symbolic representations and inference, program induction, probabilistic (logic) programming, and deep learning from instances. This study introduces a novel framework that combines general-purpose neural networks with expressive probabilistic-logical modelling and reasoning. This integration effectively utilises the full expressive capabilities and advantages of both approaches, and allows for end-to-end training using examples [**Manhaave et al.**, **2021**, 32].

Furthermore [**Oikonomakis**, **2023**, 39] conducts research on Neural-symbolic computation, a technique that integrates deep learning and reasoning using the NeurAsp software. First, the author demonstrates the capabilities of this method and explains how it works internally. He also shows how it can be integrated with traditional machine learning methods. Finally, he applies this method to recognise activities in videos involving people, using Complex Event Processing (CEP) techniques. Therefore, the author aims to emphasise the advantages of NeurAsp by conducting three experiments that compare it with established deep machine learning approaches. Machine learning enables the development of models with basic perceptual abilities, such as classification and prediction. On the other hand, higher-level inferences and information extraction are achieved by reasoning utilising suitable representations and logic-based methodologies. The integration of neural-based learning with logic-based reasoning has the potential to facilitate the development of novel systems capable of comprehending their surroundings and making inferences based on the provided input.

2.2 Complex event processing

Complex Event Processing (CEP) is a technological method that helps in drawing conclusions from various events occurring within a stream of information. This concept was highlighted in the works of [**Burgueño et al.**, **2018**, 45]. CEP enables the creation of complex events from simpler events generated by input streams. These complex events are automatically identified by the system as situations of interest.

Typically, an event is characterized by three primary aspects [**Cugola et al.**, **2012**, 11]:

1. **Form**: This refers to the specific attributes or data elements of an event. For instance, the duration of an activity could be part of an event's form, as detailed in the research by [**Vilamala**, 49] and colleagues.

2. **Significance**: This aspect highlights what an event represents or indicates. The data encompassed in an event's form often describes the nature of the activity it represents.

3. **Relativity**: This pertains to how one activity is interconnected with others. Events mirror these interconnections, exhibiting relationships similar to those of the activities they represent. The concept of relativity in an event involves the network of relationships it has with other events, and its form typically includes methods to decode these relationships.

It's important to recognize that an event is more than just a mere message or a record; while the form of an event might be in the shape of a message, it also embodies significance and relativity. Events are linked by three key types of relationships:

- **Time**: This is a sequential relationship that organizes events chronologically.

- **Cause**: This is a dependency relationship between different activities. If an event or activity happens solely because of other events or activities, there exists a causal relationship. For example, if event B is contingent on event A, then A is the cause of B. If neither event causes the other, they are considered independent.

- **Aggregation**: This occurs when event A represents an activity that is the collective outcome of a series of events $B1, B2, \ldots, Bn$. Here, A is the cumulative result of all events Bi. In other words, Bi events are components of A. Aggregation is a form of abstract relationship, often seen when event A is formulated as a consequence of the occurrence of a series of events $\{Bi\}$. Event A in this scenario is termed a complex event, as it comprises the events that led to its formation. This type of relationship can also be described as vertical causality.

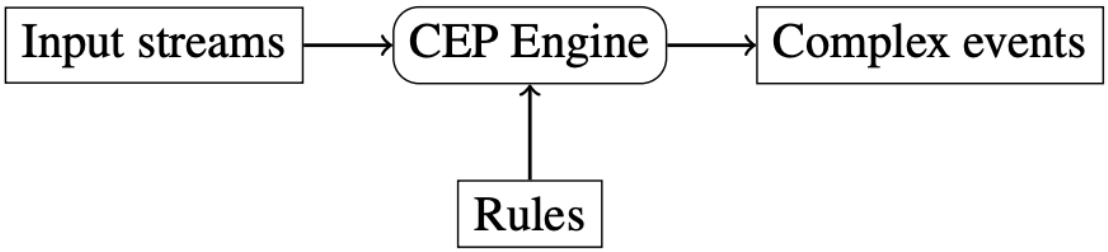
To comprehend the essence of Complex Event Processing (CEP), it's crucial to delve into its three core components, as illustrated in Figure 6:

1. **Input Stream**: The foundation of CEP lies in the continuous stream of events that serve as its input. These events represent occurrences or changes within the data source, each carrying specific attributes and timestamps that paint a vivid picture of the dynamics at play [**Luckham**, **2002**, 31].

2. **Complex Events**: The ultimate objective of CEP is to identify complex events, which are intricate assemblages of multiple simple events. These complex events transcend the boundaries of individual events, signifying meaningful occurrences or patterns within the data stream [**EtzionNiblett**, **2010**, 14].

3. **Rules**: CEP systems employ rules as the blueprints for detecting complex events. These rules are carefully crafted to capture the essence of the events we seek to identify. They are typically defined by domain experts who possess intimate knowledge of the data and its underlying patterns [**Burgueno**, **2018**, 45].

Figure 6. CEP engine



The methods for defining complex events vary, including graphical tools, SQL-like languages, logic rules, and specialized languages for defining complex events. Regardless of the approach, all CEP methodologies share common principles in defining complex events based on a combination of simpler events [**CugolaMargara**, **2012; Skarlatidis et al.**, **2015**, 47].

Luckham (2002) provides a deeper understanding of an event in CEP. An event is not just a record but has three key aspects: form (attributes or data components of the event), significance (the activity the event signifies), and relativity (the event's relationship with other events). Events are interrelated through time, cause, and aggregation. Time orders events, cause establishes a dependency relationship, and aggregation refers to the composition of a complex event from simpler events [31].

CEP primarily focuses on identifying rules for aggregation, making complex activities in a system understandable. These aggregation rules are part of an event abstraction hierarchy, which includes levels of activity descriptions and event aggregation rules [**EtzionNiblett**, **2010**, 14].

Cugola and Margara (2012) discuss information flow processing, encompassing Data Stream Management Systems (DSMS) and CEP systems. CEP views information flows as notifications to be filtered and aggregated for higher-level understanding, extending the publish-subscribe model [11].

The applications of CEP extend far beyond the confines of data analysis, reaching into domains such as network monitoring, financial data analysis, IoT management, and log analysis [**Chapnik et al.**, **2021**, **ChapnikEtAl2021 ; Yankovitch et al.**, **2022**, 53]. CEP's capabilities like real-time processing, event correlation, flexible rule definitions, and scalability make it a powerful tool for organizations to gain insights and make informed decisions [**Gomex et al.**, **2020**, 20].

2.3 Neuroplex

Deep learning algorithms have achieved significant success in various sensing applications, but they face difficulties when it comes to sophisticated reasoning tasks using multiple sensors.

This is because deep learning algorithms are typically trained on large amounts of labeled data, and complex reasoning tasks often require reasoning about multiple sensors and data streams in real time. Additionally, deep learning algorithms can be difficult to explain, which can make it difficult to trust their predictions.

Xing et al. (2020)[51] found that although deep learning algorithms have achieved significant success in various sensing applications, they face difficulties when it comes to sophisticated reasoning tasks using multiple sensors. Distinguishing transient complex activities, such as human activities like walking or running, from a single sensor is relatively easier compared to detecting more complex events that involve larger spatial and temporal dependencies across multiple sensors. For instance, utilizing a sensor network in a hospital to determine whether a nurse is adhering to a sanitary protocol while moving between patients poses significant challenges. Training a more intricate model necessitates a substantial quantity of data, which is impractical given the seldom occurrence of complex occurrences in nature. In addition, neural networks face difficulties in performing logical analysis of sequential, irregular occurrences that are widely spaced in the spatial and temporal dimensions.

In their 2020 publication, Xing et al.[51] introduced Neuroplex, a neural-symbolic framework designed to acquire the ability to engage in intricate reasoning tasks using unprocessed sensory information, while leveraging human knowledge that has been included at a higher level. Neuroplex breaks down the entire complex learning space into distinct levels of explicit perception and reasoning. This is achieved by utilizing neural networks for low-level perception tasks and reconstructed neural models for high-level, explainable reasoning.

Neuroplex enables efficient training of perception models by incorporating a neurally rebuilt reasoning model that has been trained using human knowledge. This is achieved by introducing a semantic loss using sparse, high-level annotations, resulting in effective end-to-end training. Through their research and review, it has been demonstrated that Neuroplex had the ability to learn and accurately identify intricate events, surpassing the capabilities of contemporary neural network models. Neuroplex not only decreases the need for data annotation by a factor of 100, but also accelerates the learning process for complicated event recognition by a factor of 4.

Currently, contemporary deep learning methods face challenges when it comes to processing intricate event reasoning across a dispersed array of sensors. Neuroplex is a cognitive framework that utilizes neural networks to process raw sensory data and perform intricate reasoning tasks. It achieves this by incorporating human knowledge at a higher level. Neuroplex divides the learning process into distinct layers for perception and reasoning, allowing for explicit and explainable reasoning models [**Xing et al.**, **2020**, 51].

Neuroplex enables efficient training of perception models from start to finish by incorporating a semantic loss. This is achieved by employing sparse, high-level annotations after training the reasoning model using human knowledge. The learning area is divided into two distinct components: perception and thinking. The training process involves the use of deep neural networks to acquire basic symbolic concepts for the perceptual job. Additionally, it allows for the incorporation of human knowledge at a higher level of reasoning.

Neuroplex has made significant contributions in the following areas:

- Neuroplex utilizes neural-symbolic techniques, which integrate deep learning perception with semantic logical models, to facilitate comprehensive learning for the identification of intricate events from unprocessed sensor data streams.

- Neuroplex utilizes a differential Neurally Reconstructed Logic (NRLogic) model to facilitate

learning. This model is a neural network that has been taught by a logical machine via knowledge distillation.

- Neuroplex has the capability to autonomously learn from data that has sparse, high-level, and intricate event labels. The differentiable NRLogic model allows gradients to be propagated, enabling perception modules to receive feedback from complicated event labels and be trained accordingly. The training can occur at two different stages: the initialization stage, when a perception module is untrained, and the fine-tuning stage, when a pre-trained off-the-shelf model of the perception module needs to be adjusted to a specific environment.
- Neuroplex also induces a reduction in meaning on the intermediate symbolic layer, compelling the perception module to produce a logical symbolic result in order to enhance the learning efficiency.

The Neuroplex framework underwent evaluation on three complex event datasets, where its performance was compared to that of state-of-the-art neural network models and neural-symbolic approaches. The results demonstrate that Neuroplex, when guided by injected human knowledge, is capable of successfully and efficiently learning to recognise complex events that are beyond the capabilities of other techniques. It not only enhances the speed of training by four times, but also achieves exceptional performance while using training data that is two orders of magnitude smaller.

Neuroplex suggests a hybrid neural-symbolic architecture for the purpose of identifying intricate events. The perception module consists of deep learning models designed to recognise basic events, whereas the reasoning module comprises a sophisticated logical reasoning engine that relies on human input.

Neuroplex offers both a pathway for making logical deductions based on existing human knowledge, as well as a pathway for teaching the system using this knowledge to understand complex situations. Neuroplex addresses the problem of making the reasoning module differentiable while preserving its function, and achieving effective training of the perceptual module using only high-level complex event annotations [**Xing et al.**, **2020**, 51].

Figure 7. Neuroplex (Semantic Scholars)

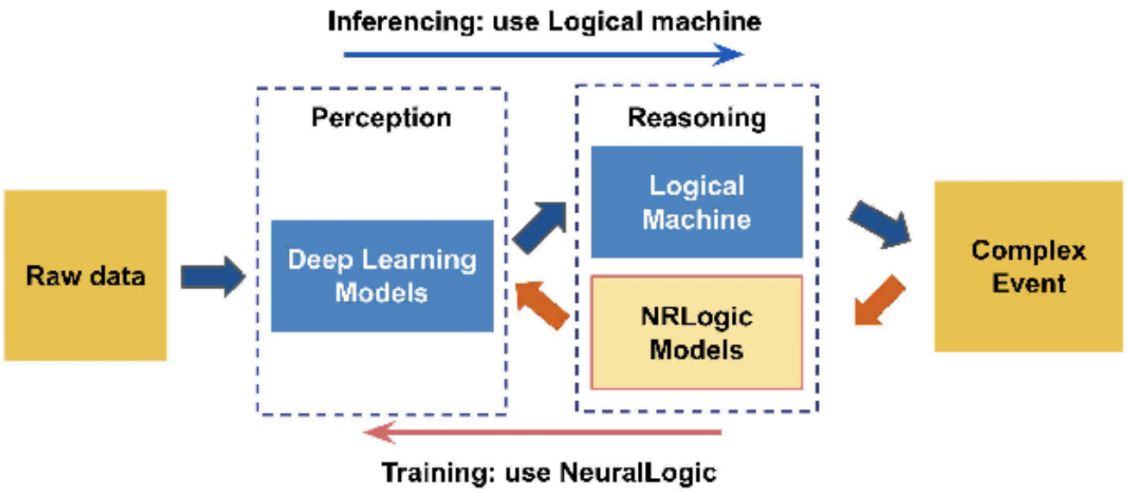


Figure 7 presents a schematic overview of Neuroplex system. It outlines the workflow from raw data acquisition to the derivation of complex events through the stages of perception and reasoning. The system utilizes both deep learning models and logical reasoning to achieve a sophisticated level of data interpretation and decision-making.

1. **Raw Data**: This is the initial input into the system, representing unprocessed information collected from various sources.

2. Deep Learning Models: Within the perception phase, deep learning models are applied to the raw data. These models are adept at extracting patterns and features, transforming the raw data into a structured format suitable for further analysis. They operate as the system's initial filter, distilling high-dimensional data into more abstract representations.

3. **Logical Machine**: In the reasoning phase, a logical machine processes the structured data obtained from the deep learning models. This component is responsible for high-level reasoning and decision-making. It applies logical rules and inference mechanisms to the data, enabling the system to make logical deductions and conclusions.

4. **NRLogic Models**: At the intersection of perception and reasoning lies the NRLogic Models, which stands for Neural-Reasoning Logic Models. These models embody the integration of neural networks and symbolic logic, allowing the system to utilize the strengths of both neural computation (for handling ambiguity and learning from data) and symbolic logic (for structured reasoning and interpretability).

5. **Complex Event:** The outcome of this integrated process is the identification or prediction of a complex event. This refers to an inferred situation or occurrence that is identified based on the combination of learned patterns from data and the application of logical reasoning.

6. **Training and Inferencing**: The diagram also denotes two critical stages in the system's lifecycle: training and inferencing. During training (labeled 'Training: use NeuralLogic'), the NRLogic Models learn from the data by adjusting the deep learning and logical parameters to improve prediction and reasoning accuracy. In the inferencing phase (labeled 'Inferencing: use Logical machine'), the system applies the trained models to new data to deduce complex events.

The process flow encapsulated in Figure 7 illustrates the synergy between machine learning and logical reasoning, signifying the essence of neural-symbolic integration. It highlights how raw data can be transformed into actionable insights by leveraging the complementary capabilities of deep learning and symbolic logic.

2.4 DeepProbLog

2.4.1 ProbLog

Problog is a probabilistic logic programming language, a powerful tool that merges two traditionally distinct fields: logic programming and probabilistic reasoning. This integration allows Problog to handle complex models that require both logical rules and uncertainty representation, making it particularly useful in areas like artificial intelligence (AI) and data science [**De Raedt**, **Kimmig**, **& Toivonen**, **2007**, 13].

Logic programming, a key component of Problog, is based on formal logic. In traditional logic programming languages like Prolog, programs consist of facts and rules. Facts represent basic assertions about the world, while rules define relationships between different facts [**Kowalski**, **1979**, 24]. For instance, if you have facts like ”Alice is a parent of Bob” and ”Bob is a parent of Charlie,” you can define a rule that Alice is a grandparent of Charlie. Logic programming excels in expressing such relationships and is used extensively in areas where complex relationships need to be represented, such as database querying and AI [**Lloyd**, **1987**, 30].

What sets Problog apart is its ability to incorporate probabilistic reasoning. In many real-world scenarios, facts are not always certain. For example, instead of definitively knowing that ”Bob is at home,” we might only know there’s a 70% chance. Probabilistic reasoning deals with such uncertainties. It allows the representation of incomplete or uncertain information and makes inferences based on probabilities. This capability is crucial in fields like machine learning, where uncertainty is a common aspect of data and predictions [**Pearl**, **1988**, 41].

In Problog, each fact can have an associated probability, indicating the likelihood that the fact is true. The syntax is straightforward: a probability is written in front of a fact or a rule. For example, 0.7::at_home(bob). states that Bob is at home with a probability of 70%. Problog computes the probabilities of various queries by combining these individual probabilities according to the rules of probability theory [**De Raedt et al.**, **2007**, 13].

2.4.2 DeepProbLog Overview

DeepProbLog is a programming language that combines probabilistic logic and deep learning through the utilisation of neural predicates. The system is capable of accommodating both symbolic and sub-symbolic representations and inference, program induction, probabilistic (logic) programming, and deep learning. This framework is the first to integrate general-purpose neural networks with expressive probabilistic logical modelling and reasoning. It leverages the entire expressiveness and strengths of both approaches, allowing for end-to-end training.

DeepProbLog enables users to train neural networks within the system using various designs in a seamless manner. A DeepProbLog program is a ProbLog program augmented with a collection of instantiated neural Ads (nADs) in the format nn. In this context, ”nn” serves as an abbreviation for ”neural network” while ”mq” is used as an identifier for a specific neural network. The neural network mq will receive the input vector and generate a probability distribution throughout the domain. NADs function in a manner akin to ads by offering a distribution of probabilities that are mutually exclusive across a set of atoms. However, in nADs, these probabilities are derived from the output of a neural network rather than being explicitly specified. The total of the probabilities throughout the domain O must be equal to 1. In the context of neural networks used for multi-class classification, it is common practice to employ a softmax layer to convert the real-valued output scores [**Manhaeve et al.**, **2021**, 33].

Once the structure of the neural network and the logic level have been established, DeepProbLog may be employed to deduce the responses to queries. DeepProbLog utilises a process called inference to convert the logic layer into an arithmetic circuit and extract the necessary probabilities from the neural network.

Subsequently, this arithmetic circuit can be employed to compute the probability of the question being true, relying on the neural network's output. To train the neural network, the system initially carries out inference as previously explained. DeepProbLog has the capability to execute gradient-based learning. The arithmetic circuit employed throughout the inference process is also utilised for conducting the gradient computations. Given that this arithmetic circuit consists of addition and multiplication operations, it may be inferred that it is differentiable. DeepProbLog is capable of calculating the gradient in relation to the probabilistic logic program. Subsequently, this gradient can be employed to train the neural network through the utilisation of backpropagation [**Manhaeve et al.**, **2021**, 33].

A limitation of DeepProbLog is its exclusive support for precise probabilistic inference. Consequently, if the program expands in size, certain stages in the inference process can become excessively expensive. The proposed integration is not directly connected to this problem, however, it can be resolved in the future by expanding DeepProbLog to incorporate approximate inference.

2.4.3 DeepProbLog Inference

Once the structure of the neural network and the logic level have been established, DeepProbLog may be employed to deduce the responses to our inquiries. DeepProbLog utilises a process called inference to convert the logic layer into an arithmetic circuit and extract the necessary probabilities from the neural network. Subsequently, this arithmetic circuit can be employed to compute the probability of the question being true, relying on the neural network's output. To train the neural network, the system initially carries out inference as previously explained. DeepProbLog is capable of conducting learning through gradient-based methods. The arithmetic circuit utilised throughout the inference process is likewise employed for the purpose of performing the gradient computations. Given that this arithmetic circuit consists of addition and multiplication operations, it may be inferred that it is differentiable. DeepProbLog is able to calculate the gradient in relation to the probabilistic logic program. Subsequently, this gradient can be employed to train the neural network through the utilisation of backpropagation [**Vilamala et al.**, **2021**, 49].

DeepProbLog inference elucidates the utilisation of a model to forecast a specific query. DeepProbLog is a modification of ProbLog inference. The initial stage of ProbLog inference is the grounding step, where the logic program is grounded in relation to the question at hand. In this stage, backward reasoning is employed to identify the pertinent ground rules that determine the truth value of the query. Additionally, it may do other logical simplifications that are unrelated to the query's probability. In the second phase, the ground logic program is transformed into a propositional logic formula that represents the truth value of the query using the truth values of probabilistic facts. However, performing WMC on this logical formula right away is not efficient [**Manhaeve et al.**, **2021**, 32].

The aggregation of information is the third stage. At this stage, the logic formula is transformed into a format that enables efficient weighted model counting. The current ProbLog system utilises Sentential Decision Diagrams, which are the most concise and suitable representation currently available. Polytime model counting can be achieved using SDDs, which are a specific type of deterministic decomposable negational normal forms (d-DNNFs). However, they do offer polynomial time conjunction, disjunction, and negation operations, and are more succinct than OBDDs. In the fourth and final stage, the SDD is transformed into an AC. The probabilities of the probabilistic facts or their negations are assigned to the leaves, while the OR nodes are substituted with addition and the AND nodes with multiplication. The WMC is subsequently calculated by an AC assessment.

The only requirement for DeepProbLog inference is to convert nADs and neural facts into normal ADs and probabilistic facts. This process consists of two distinct steps. Obtain ground nADs and ground neural facts using a symbolic representation of the likelihood during the grounding

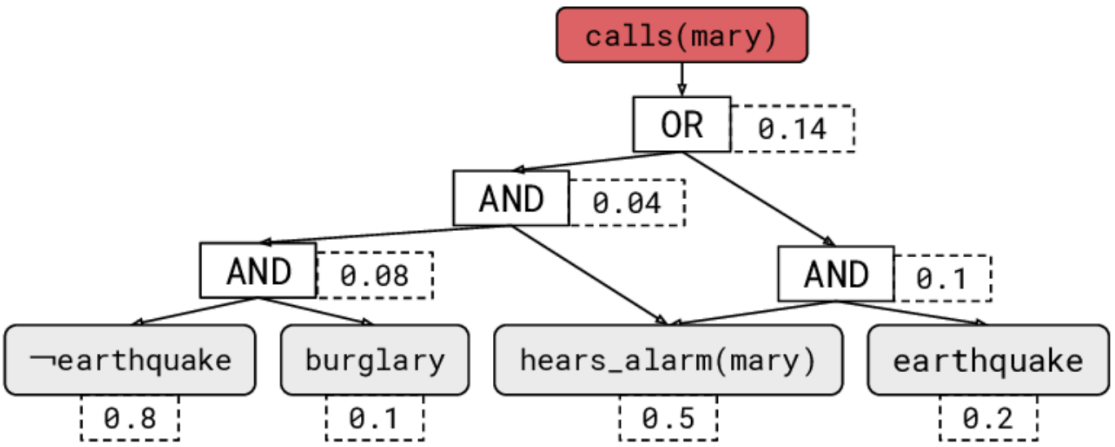
process. The specific parameters are computed at a distinct stage following the grounding process, by performing a forward pass on the relevant neural network using the grounded input (Manhaeve et al., 2021).

2.4.4 Learning in DeepProbLog

An artificial neural network represents a complex mathematical function with high customizability, allowing it to be trained to exhibit specific behaviors by adjusting its parameters. During the training phase, the network learns to discern and distill essential features from the input data relevant to the task at hand. This ability negates the need for manual feature engineering, a staple of non-neural machine learning techniques, and shifts the focus to architectural engineering. The field now boasts a plethora of neural network components that can be assembled in various configurations to construct a model tailored for particular applications [**Manhaeve**, **2021**, 33].

Deep neural networks are typically constructed and trained in an end-to-end fashion, where the only knowledge imparted during training is of the initial input and the final objective. The entire model, including all its components, undergoes simultaneous training. For instance, in the domain of handwritten digit recognition, an input instance would be an image of a handwritten digit, and the target is the actual numeric value it represents [**Manhaeve**, **2021**, 33].

Figure 8. Arithmetic Circuit for Probabilistic Query in DeepProbLog



(d) The AC for query `calls(mary)`.

Figure 8 illustrates an Arithmetic Circuit (AC) utilized for a probabilistic query in DeepProbLog. The AC is employed not only for probabilistic inference but also for gradient computation. In the depicted AC, the query calls(mary) is evaluated through a series of logical AND and OR gates, which are inherently differentiable due to their additive and multiplicative nature. This circuit calculates the probability of Mary receiving a call, given certain conditions such as an earthquake or a burglary, and the likelihood of her hearing an alarm. Each leaf node represents a probabilistic fact (e.g., ¬earthquake with a probability of 0.8), and the internal nodes combine these probabilities through logical conjunctions (AND) and disjunctions (OR) to deduce the final probability of the query.

DeepProbLog leverages gradient-based learning, diverging from the expectation-maximization approach previously used by Gutmann et al. (2008) [21] for parameter learning in ProbLog. This shift facilitates the integration of probabilistic logic networks with traditional neural network training pipelines, enhancing the model's learning capabilities [**Manhaeve**, **2021**, 33].

The primary distinction between inference in traditional ProbLog and DeepProbLog is the evaluation of neural predicates and numerical atoms (nADs). In DeepProbLog, inference also involves the integration of neural network outputs, which serve as probabilistic facts. Similarly, the key difference in gradient-based learning between the two systems is the concurrent optimization of both neural and probabilistic parameters within DeepProbLog.

In the context of DeepProbLog, the probabilistic parameters π in the logic program are adjustable via the gradient semiring. This novel approach allows for the computation of the gradients alongside the probabilistic inferences. The gradients thus obtained are crucial for updating the model parameters through gradient-based learning algorithms.

Figure 9. Learning Pipeline in DeepProbLog

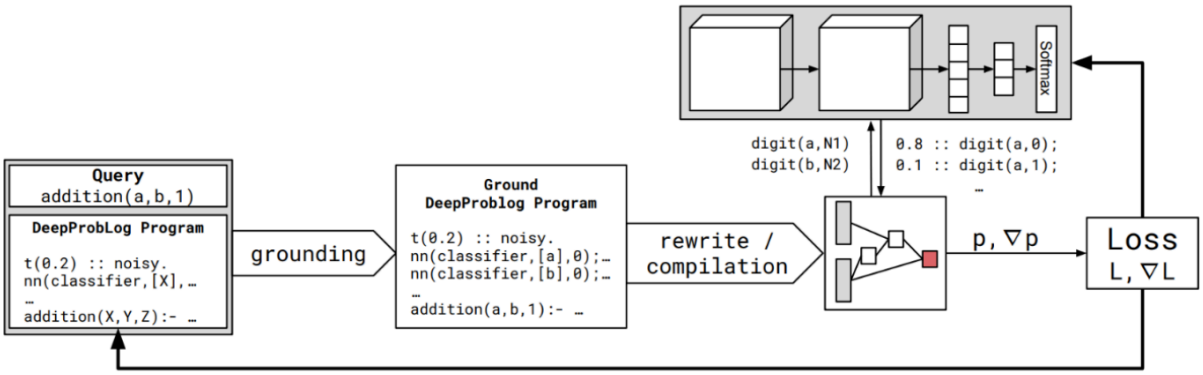


Figure 9 delineates the learning pipeline in DeepProbLog, which encompasses several stages:

1. **Query and DeepProbLog Program**: The process begins with a query to the system, for example, addition(a,b,1), and a DeepProbLog program that contains a mix of probabilistic facts (e.g., $t(0.2) :: noisy.$) and neural predicates (e.g., $nn(classifier, [X], ...)$.

2. **Grounding**: The DeepProbLog program is then grounded. Grounding entails transforming the program into a format where the variables are replaced with constants. This step results in a ground program where neural predicates are instantiated with actual data inputs.

3. **Rewriting/Compilation**: Subsequently, the ground program is rewritten or compiled into an arithmetic circuit (AC). The AC is a computational graph that represents the probabilistic logic program in a structure amenable to both probabilistic inference and gradient computation.

4. **Neural Network Integration**: The neural network components, such as classifiers, are integrated with the AC. They process inputs and provide probabilities as outputs, which are incorporated into the AC as terminal nodes.

5. **Loss Calculation**: The output of the AC, combined with the neural network predictions, leads to the calculation of a loss function (denoted as L), which measures the difference between the predicted and actual values. The gradient of the loss (denoted as 'L') is then computed, providing the necessary information for parameter updates.

6. **Gradient-Based Learning**: Utilizing the computed gradients, the model parameters are updated using gradient descent or other optimization techniques. This step completes the

learning cycle, allowing DeepProbLog to refine both its neural network weights and probabilistic facts for improved performance on subsequent queries.

This learning pipeline exemplifies a unified and differentiable framework where conventional differentiation methods are applied to compute gradients, thereby facilitating the optimization of the combined neural-symbolic model [**Manhaeve**, **2021**, 33].

2.4.5 DeepProbCEP: A Neuro-Symbolic Approach for Complex Event Processing

In the current data-driven era, Complex Event Processing (CEP) has become essential for extracting meaningful insights from high-volume, high-velocity data streams. However, traditional CEP systems often grapple with the complexity and uncertainty inherent in real-world data streams, making them less effective for applications requiring high accuracy, robustness, and adaptability.

Conventional CEP systems typically utilize rule-based or statistical methods to identify and analyze events. While effective in certain scenarios, these methods encounter limitations when dealing with dynamic, real-world data:

1. **Handling of Rare or Infrequent Events**: Traditional CEP systems may struggle to detect rare or infrequent events due to their rarity amidst large volumes of data, leading to significant oversight [**Etzion et al.**, **2010**, 14].

2. **Sensitivity to Noise and Incomplete Data**: Real-world data streams are often riddled with noise and incomplete information, posing a challenge for event detection and potentially leading to false positives or missed events in traditional CEP systems [**Cugola et al.**, **2012**, 11].

3. **Adaptability to Changing Patterns**: Evolving event patterns due to user behavior, system changes, or external factors can impede the performance of conventional CEP systems, which may not be sufficiently adaptable [**Luckham et al.**, **2002**, 31].

To surmount these challenges, DeepProbCEP [**Vilamala**, **2022**, 50] represents a groundbreaking neuro-symbolic approach, integrating the strengths of deep learning with probabilistic logic programming. This combination significantly enhances the capabilities of CEP systems.

DeepProbCEP employs deep neural networks to effectively extract features and discern patterns from raw data streams. By transforming raw data into a structured representation, it lays the groundwork for the subsequent reasoning layer. The prowess of deep learning in recognizing intricate patterns and correlations furnishes DeepProbCEP with a solid base for event detection. This innovative approach addresses the inherent limitations of traditional CEP methods, enabling more accurate and flexible event processing even in complex and uncertain environments [**Yankovitch et al.**, **2022**, 53].

Probabilistic Logic Programming for Event Reasoning

Probabilistic logic programming, a fusion of logic programming and probability theory, is instrumental in DeepProbCEP for reasoning with uncertainty. This approach facilitates the definition of complex event patterns and probabilistic reasoning about their occurrences based on the features and patterns extracted from data streams [**DeRaedt et al.**, **2015**, 12]. By incorporating probabilistic logic, DeepProbCEP quantifies and manages the inherent uncertainty in data, significantly enhancing its robustness against noisy or incomplete information [**Riguzzi et al.**, **2013**, 44].

The architecture of DeepProbCEP is a harmonious integration of deep learning and probabilistic logic programming. It creates a unified representation, merging symbolic reasoning with subsymbolic data processing. This dual approach enables DeepProbCEP to interpret both high-level

abstractions and detailed numerical data, making it exceptionally capable of handling complex event patterns in diverse real-world scenarios [**Manhaeve et al.**, **2021**, 32].

A key advantage of DeepProbCEP over traditional CEP systems is its capacity to learn and evolve from data. The system continually refines its understanding of event patterns and reasoning rules based on incoming data, allowing it to adapt to changing environments and enhance its performance over time. Such adaptability makes DeepProbCEP particularly effective in dynamic and evolving settings, where fixed rule-based systems might falter.

Furthermore, DeepProbCEP is adept at operating in adversarial contexts, where the integrity of event streams might be at risk from manipulation or disruption. Leveraging its probabilistic logic underpinnings, the system is equipped to recognize and respond to uncertainties and anomalies effectively. This capability is vital for maintaining security and integrity in critical applications such as finance, cybersecurity, and infrastructure monitoring, where adversaries may attempt to exploit vulnerabilities in event processing systems [**Etzion et al.**, **2010**, 14].

Applications of DeepProbCEP

DeepProbCEP's versatility is evident in its broad range of real-world applications:

- **Fraud Detection**: In financial systems, DeepProbCEP can identify aberrant patterns that signify fraudulent transactions. By analyzing transaction data, it detects anomalies that deviate from normal behavior, playing a crucial role in safeguarding financial integrity [**Bolton et al.**, **2002**, 9].

- **Network Security**: For cybersecurity, DeepProbCEP monitors network traffic to identify signs of intrusions or cyberattacks. Its capability to process and analyze vast streams of data in real-time makes it an invaluable tool for defending against sophisticated cyber threats [**Sommer et al.**, **2010**, 48].

- **Predictive Maintenance**: In industrial settings, DeepProbCEP analyzes sensor data to anticipate equipment failures. This predictive approach enables proactive maintenance strategies, reducing downtime and maintenance costs significantly [**Mobley et al.**, **2002**, 38].

- **Log Analysis**: DeepProbCEP analyzes log data from complex systems to identify malfunctions, security breaches, or performance issues. Its sophisticated pattern recognition capabilities facilitate swift intervention and corrective actions [**Xu et al.**, **2009**, 52].

Additionally, DeepProbCEP is highly effective in human activity detection from video streams. Its neuro-symbolic framework, which combines feature extraction with logic-based reasoning, is particularly suited for tasks involving uncertainty and complex pattern recognition [**Poppe**, **Roland**, **2010**, 43].

For instance, in monitoring crowd behavior or identifying unusual activities, DeepProbCEP operates as follows:

1. It extracts features from video streams, such as positions, movements, and appearances of individuals.

2. Utilizes probabilistic logic programming to define complex event patterns correlating to different human activities, such as walking or running [**Liu et al.**, **2018**, 29].

3. Reasons about these features to classify events that match the defined patterns, enabling the detection of activities like walking, running, jumping, or falling.

DeepProbCEP's handling of uncertainty is a key strength, particularly in scenarios where video data may be noisy or incomplete. Factors such as shadows, occlusions, or low resolution may obscure critical visual information. The probabilistic framework of DeepProbCEP allows it to manage such uncertainties effectively, ensuring accurate detections despite data imperfections [**Piciarelli et al.**, **2006**, 42].

Chapter 3

Extending DeepProbLog Complex Event Processing

Building upon the foundational principles of DeepProbLog and its initial applications in complex event processing as discussed in Chapter 2, this chapter delves deeper into the theoretical advancements and novel applications of DeepProbLog, exploring its extended capabilities and innovative integrations in more complex scenarios.

Initially, we delve deeper into the probabilistic logic underpinnings of DeepProbLog, examining complex relationships and the nuanced handling of uncertainty, which are pivotal for sophisticated event processing. This is followed by an exploration of how DeepProbLog seamlessly integrates neural networks with probabilistic logic, shedding light on the harmonious interplay between symbolic reasoning and subsymbolic learning. We then pivot to practical applications, presenting case studies where DeepProbLog's advanced capabilities have been leveraged, thereby illustrating its practical effectiveness and versatility. Finally, the chapter discusses the technical challenges encountered in advancing DeepProbLog's capabilities, along with the innovative solutions developed to overcome these obstacles, showcasing the evolving nature of this technology.

3.1 Deep Dive into Probabilistic Logic

3.1.1 Introduction to Probabilistic Logic

Probabilistic logic, a cornerstone in reasoning under uncertainty, merges the clarity of logic with the dynamic nature of probabilistic reasoning, akin to Bayesian networks. This approach is particularly effective in scenarios where data is incomplete or ambiguous. In DeepProbLog, this concept is taken further by integrating it with deep neural networks, enhancing its capability to discern complex relationships in data. This fusion results in a powerful tool, combining logical expressiveness and the nuanced understanding of probabilities, crucial for accurately navigating real-world complexities. Mathematical expressions in this domain, like conditional probabilities $(P(A|B))$ and Bayes' Theorem $(P(A|B) = P(B|A) * P(A)/P(B))$, quantify these uncertainties, allowing DeepProbLog to effectively leverage both logical and probabilistic information.

3.1.2 Probabilistic Logic vs. Classical Logic

Probabilistic logic and classical logic are two different logics that have different strengths and weaknesses.

Classical logic is a formal system that deals with statements that are either true or false. It is based on the idea that there are two truth values, true and false, and that every statement must have one of these two truth values. Classical logic is used to reason about deductive arguments, which are arguments that are valid if and only if the conclusion follows from the premises.

Probabilistic logic is a formal system that deals with statements that have a probability of being true. It is based on the idea that there are a range of truth values, from 0 (completely false) to 1 (completely true), and that every statement has a probability of being true that lies somewhere between these two values. Probabilistic logic is used to reason about uncertain situations, where we do not have complete information about the truth of a statement.
Here is a table summarizing the key differences between probabilistic logic and classical logic:

Feature	Probabilistic logic	Classical logic
Truth values	Continuous range from 0 (false) to 1 (true)	Two: true and false
Reasoning	Inductive reasoning	Deductive reasoning
Applications	Uncertain situations	Deductive arguments

Table 1. Probabilistic Logic vs. Classical Logic

3.1.3 Representation of Uncertainty in DeepProbLog

In DeepProbLog, the representation of uncertainty is a critical component, integrating probabilistic reasoning with logic programming. It assigns probabilities to logic predicates, enabling the handling of scenarios where data may be incomplete, ambiguous, or noisy. This probabilistic approach allows DeepProbLog to model real-world complexity more accurately, embracing uncertainty as an inherent aspect of data and knowledge. By utilizing probabilities, DeepProbLog can make inferences and predictions even when faced with partial information, thus bridging the gap between deterministic logic and the probabilistic nature of real-world data

3.1.4 Probabilistic Inference Mechanisms

Probabilistic inference mechanisms in DeepProbLog involve mathematical techniques to deduce the likelihood of certain outcomes based on known probabilities. Using Bayes' Theorem, $P(A|B) = \frac{P(B|A) \times P(A)}{P(B)}$, it revises the probability of a hypothesis A in light of new evidence B. Furthermore, DeepProbLog employs algorithms for probabilistic inference like belief propagation, where $P(A) = \sum_B P(A|B) \times P(B)$, to combine various probabilistic inputs. This approach allows DeepProbLog to handle complex scenarios where direct observation is not possible, enabling it to infer hidden or uncertain information from known data.

3.1.5 Extending Traditional Logic Programming

DeepProbLog's extension of traditional logic programming marks a significant evolution in handling real-world complexities. Traditional logic programming, based on a binary framework of true or false, often falls short in modeling the uncertain, probabilistic nature of real-life scenarios. DeepProbLog addresses this by incorporating probabilistic reasoning, allowing for a more flexible,

nuanced representation of information. This approach not only retains the structured, rule-based reasoning of classical logic but also integrates the ability to manage uncertain, incomplete, or ambiguous data. By doing so, DeepProbLog expands the horizons of logic programming, making it apt for applications where uncertainty is inherent, such as in decision-making processes, AI reasoning systems, and complex data analysis. This integration also paves the way for new research directions and applications, where the combination of deterministic logic and probabilistic inference can lead to more robust, adaptable, and intelligent systems. The result is a more dynamic, versatile logic programming paradigm, better suited to the complexities and uncertainties of the modern data-driven world.

3.1.6 Probabilistic Logic and Machine Learning

The integration of probabilistic logic with machine learning in DeepProbLog presents a significant advancement in AI. This combination allows for the incorporation of uncertainty and probabilistic reasoning into machine learning models, enabling them to handle real-world data more effectively. Probabilistic logic provides a framework for representing and reasoning about uncertain information, which is often encountered in complex datasets. When coupled with machine learning, particularly deep learning, it enhances the model's ability to learn from data that is not only large and complex but also uncertain or incomplete. This synergy leads to more robust, adaptable, and intelligent systems that are better suited for tasks like pattern recognition, decision making, and predictive analytics in diverse domains, ranging from natural language processing to robotics. The fusion of probabilistic logic and machine learning has opened up new avenues for innovation in artificial intelligence. DeepProbLog, a probabilistic logic programming language, represents a notable example of this integration. It harnesses the expressiveness of probabilistic logic and the representational power of deep learning to model complex systems with uncertainty.

3.2 Integration of Neural Networks

This section explores the innovative integration of neural networks with probabilistic logic programming in DeepProbCEP, showcasing a leading-edge fusion of symbolic and subsymbolic AI. This synergy between different layers of AI is at the core of DeepProbCEP's approach to complex event processing (CEP), combining the strengths of both neural networks and logic-based reasoning.

DeepProbCEP approaches the CEP task by dividing it into two complementary levels:

1. **Perception Level**: Operating at a low level, this layer employs neural networks to classify raw, non-symbolic data into identifiable simple events. The neural network, pre-configured with specific classes, extracts symbolic information from the input data, transforming the raw sensory input into a structured format [**Bengio et al.**, **2013**, 7].

2. **Reasoning Level**: At a higher level, DeepProbCEP uses probabilistic logic programming to recognize complex events. It processes user-defined rules against the symbolic data derived from the perception layer, determining the occurrence of complex events [**DeRaedt et al.**, **2015**, 12].

The perception layer acts as the 'eyes' of DeepProbCEP, efficiently categorizing non-symbolic data into meaningful segments. This symbolic data is then utilized by the reasoning layer, akin to the 'brain' of the system, which applies probabilistic logic programming for deeper analysis and event identification. This layered approach ensures a seamless transition from data perception to high-level reasoning.

Distinct from conventional methods that rely on pre-trained neural networks, DeepProbCEP adopts an end-to-end training approach. This seamless integration within the probabilistic logic programming framework enables the system to learn directly from data, adapting to the nuances of specific tasks [**Manhaeve et al.**, **2021**, 33].

Advantages of DeepProbCEP's neuro-symbolic integration include:

1. **Knowledge Injection**: The inclusion of human expertise through rules enhances the system's interpretive and analytical capabilities, bridging the gap between AI and human reasoning [**Marcus et al.**, **2019**, 34].

2. **Limited Data Training**: DeepProbCEP's efficiency in learning from limited data resources makes it particularly suitable for scenarios where data availability is a challenge [**Lake et al.**, **2017**, 26].

3. **End-to-End Learning**: This approach enables the system to evolve and adapt its learning process to specific task requirements, ensuring a more tailored and effective solution [**Le Cun et al.**, **2015**, 27].

4. **Symbolic Information Processing**: The use of probabilistic logic programming allows for sophisticated processing of the symbolic information generated by the perception layer, enhancing the system's reasoning capabilities [**Getoor et al.**, **2007**, 18].

DeepProbCEP exemplifies the potential of merging neural networks with symbolic reasoning, showcasing a formidable approach to tackle complex event processing tasks. Its capacity to handle limited data, integrate human knowledge, and process symbolic information positions it as a significant tool for diverse CEP applications.

3.2.1 Reasoning Layer

DeepProbCEP is built upon DeepProbLog, a probabilistic logic programming framework developed by Manhaeve et al. [**2021**, 49]. This integration allows us to define complex events using ProbLog rules. With ProbLog code, users can specify the conditions that must be met for each type of complex event to be considered active. This can be accomplished by creating a clause for each complex event type, where the clause's truth value determines whether or not the event has occurred. Consequently, users must define rules that specify when these clauses should be considered true, triggering the event's activation.

To effectively define these rules, users need to discern the occurrence of simple events. The method for assessing the state of simple events depends on their origin. If simple events originate from symbolic inputs, they can be directly incorporated into the reasoning layer, bypassing the perception layer. This can be achieved by adding clauses reflecting their values to the ProbLog code. Conversely, if simple events stem from non-symbolic data, they must undergo perception layer processing beforehand. This can be facilitated by employing a neural AD, enabling communication between the perception and reasoning layers. For instance, this can be accomplished by adding the following line to the ProbLog code:

$$nn(mnist\_net, [X], Y, [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]) :: digit(X, Y).$$

This line defines a clause named $digit(X, Y)$, which provides the argument X to the neural network mnist_net. This neural network has 10 distinct outputs, which are mapped to digits between 0 and 9 using a list. Once this line is added, the clause $digit(X, Y)$ can be leveraged to consult the perception layer. For this example, the value of X must be an MNIST image. For each value of X, and for each possible digit in Y, the clause is true with a certain probability. This probability is determined by the output of the neural network in the perception layer using X as an input.

Upon defining simple events in the logic layer, users are tasked with crafting rules that effectively discern the emergence of complex events based on these simple events. Diversified frameworks can be employed for this purpose. However, for the present discussion, we restrict our focus to complex events structured as patterns of simple events, where the events must unfold in a specific sequence within a specified time window. This approach aligns with the prevailing literature, which primarily employs this type of event definition. To effectively capture these patterns, DeepProbCEP leverages functionalities from the sequence framework [**Vilamala**, **2022**, 50].

3.2.2 Perception Layer

The perception layer of DeepProbCEP stands as the gateway between the non-symbolic data stream and the high-level reasoning engine [**Vilamala**, **2022**, 50]. It's tasked with transforming raw data into symbolic representations that can be processed by the reasoning module. This conversion is achieved through a neural network trained specifically to classify non-symbolic inputs into a pre-defined set of categories. The neural network's architecture can be tailored using PyTorch [**Paszke et al.**, **2017**, 40], enabling users to customize it to suit the nature of their non-symbolic data. This adaptability empowers DeepProbCEP to handle a wide range of data types, including text, images, and numerical values.

However, due to its integration with DeepProbLog, certain limitations apply to the implementation of the neural network. Firstly, DeepProbLog currently doesn't support regression-based outputs, necessitating the use of classification-oriented neural networks. Secondly, the predicted class probabilities generated by the neural network are converted into an AC (Approximate Certainty) value, where the score associated with each class signifies its probability. This ensures that the probabilities for all classes add up to a maximum of 1, typically accomplished by incorporating a SoftMax layer at the conclusion of the neural network.

Once both the reasoning and perception layers are defined, DeepProbLog [**Manhaeve et al., 2021**, 33] is utilized to train the perception layer neural network in an end-to-end manner, optimizing its performance for classifying simple events. After successful training, the perception layer neural network can be extracted, providing a trained classifier capable of analyzing non-symbolic data. This can serve as a valuable byproduct, particularly in situations where direct access to training data for such a classifier is restricted.

3.3 Technical Challenges and Solutions

One of DeepProbCEP's core challenges lies in reconciling two fundamentally different AI paradigms: symbolic probabilistic logic and subsymbolic neural networks. While neural networks excel at handling raw, continuous data and providing fluid probability distributions, probabilistic logic thrives on discrete, categorical representations. Bridging this gap is crucial for DeepProbCEP's operation, requiring seamless interaction between these distinct approaches.

The solution revolves around the ingenious SoftMax layer, meticulously designed to translate the neural network's continuous output into a format compatible with the logic programming layer. This layer transforms the network's output into a probability distribution across pre-defined classes, carefully mapping each class probability to a corresponding logical term in the probabilistic logic layer. This strategic integration, a cornerstone of DeepProbCEP's architecture, achieves several crucial objectives:

1. **Lossless Translation**: By converting continuous data into a format understandable by the logic programming layer, the nuanced, probabilistic insights gleaned from the neural network are not lost but effectively utilized in symbolic reasoning processes.

2. **Enhanced Accuracy and Efficiency**: The neural network's ability to extract subtle data patterns and nuances, now effectively incorporated into the logic programming framework, empowers DeepProbCEP to process complex events with greater accuracy and efficiency. This is particularly evident in scenarios involving intricate data patterns and subtle probabilistic shifts, where the system's enhanced reasoning capability leads to more accurate event detection and classification.

Beyond merely solving the technical challenge, this integration unlocks significant advantages for DeepProbCEP:

- **Improved Generalizability**: DeepProbCEP can learn from both symbolic and non-symbolic data, leading to improved generalizability across diverse data types and tasks.

- **Fine-grained Reasoning**: The combined power of neural networks and symbolic logic enables DeepProbCEP to perform fine-grained reasoning, capturing subtle relationships and nuances within complex data patterns.

- **Data-efficient Learning**: By leveraging the strengths of both paradigms, DeepProbCEP can achieve impressive results even with limited training data, a valuable asset in many real-world applications.

While alternative approaches to bridging the symbolic-subsymbolic gap exist, the SoftMax-based solution offers a particularly elegant and efficient solution for DeepProbCEP's unique requirements. This seamless integration not only paves the way for accurate and efficient complex event processing but also opens doors for exciting future research avenues in hybrid AI, where the combined strengths of neural networks and symbolic reasoning can unlock even greater potential.

Chapter 4

Experimental Results

4.1 Experimental Methodology

Chapter 4 marks a pivotal transition from the theoretical foundations laid in the preceding chapters to a rigorous empirical exploration, delving into the practical realm of complex event processing (CEP). This chapter is dedicated to a comprehensive empirical evaluation of DeepProbCEP and a simple LSTM model, aimed at bridging the theoretical concepts of CEP with their real-world applicability. Through a series of meticulously crafted experiments, we endeavor to offer an in-depth analysis of these models, scrutinizing their performance across a spectrum of conditions and scenarios.

In an effort to provide a holistic understanding of CEP technologies, our experimental framework has been further enriched by the inclusion of an advanced LSTM-over-CNN model. This hybrid model artfully integrates Convolutional Neural Networks (CNNs) for their exceptional image feature extraction capabilities with the temporal processing prowess of Long Short-Term Memory (LSTM) networks. Our objective transcends mere performance comparison; it seeks to unravel the intricacies of each model – the standalone LSTM, the LSTM-over-CNN hybrid, and the sophisticated DeepProbCEP – and their efficacy in addressing the multifaceted challenges inherent in complex event processing tasks. This comprehensive approach promises not only to highlight the unique strengths and limitations of each model but also to illuminate the path forward in the evolution of CEP methodologies.

Overview of Experimental Approach

In this chapter, our exploration delves into the empirical realm, predominantly focusing on the advanced capabilities of DeepProbCEP within the landscape of complex event processing (CEP). The experimental approach is designed with the intention of comprehensively evaluating DeepProbCEP, a sophisticated model that merges the analytical strengths of neural networks with the nuanced reasoning of probabilistic logic. These experiments are pivotal in demonstrating DeepProbCEP's adeptness in interpreting and managing the complexities inherent in dynamic, real-world data scenarios, marking a significant advancement in the field of CEP.

Central to our experimental approach is the in-depth analysis of DeepProbCEP's functionalities and performance. To provide a robust framework for this analysis, we introduce a baseline comparison with a Long Short-Term Memory (LSTM) network, renowned for its efficacy in sequential data processing. This comparison serves as a crucial benchmark, shedding light on the evolutionary progress and the enhanced capabilities that DeepProbCEP brings to the domain of sequence processing.

Further enriching this comparative landscape, we incorporate an LSTM-over-CNN model into our suite of experiments. This hybrid model, which combines the feature extraction prowess of

Convolutional Neural Networks (CNNs) with the sequential data handling of LSTMs, serves as an intermediary touchstone. It allows us to dissect and appreciate the complexity that DeepProbCEP manages with its unique blend of neural network computation and probabilistic logic. The inclusion of the LSTM-over-CNN model is instrumental in delineating the scope and scale of DeepProbCEP's innovation, particularly in its ability to unravel intricate event patterns that pose challenges to more conventional neural network models.

Selection of Models

In our experimental journey, three distinct models are subjected to a series of rigorous tests using the MNIST dataset. Renowned in the machine learning community, this dataset serves as a standard benchmark, particularly apt for simulating a stream of digit sequences. This provides an ideal testing ground for evaluating each model's ability to detect complex sequences and DeepProbCEP resilience to varying levels of noise. For DeepProbCEP, our experimental setup entails training the model on these digit streams, with an emphasis on identifying complex events. These events are defined as consecutive occurrences of the same digit within a dynamic window size. The architecture of DeepProbCEP, as elaborated in earlier sections, encompasses a data pre-processing layer, a perception layer named DigitNN, and a probabilistic logic programming layer. This sophisticated structure is designed to unravel the complexities of sequential data, highlighting DeepProbCEP's unique capabilities in handling intricate event patterns under various conditions.

In parallel, the LSTM model provides a contrastive baseline. Utilizing a standard LSTM network architecture, devoid of the probabilistic logic programming component, this model is fine-tuned with an optimized number of hidden layers and units specifically for processing the MNIST digit streams. The core objective here is to evaluate the LSTM's performance in sequence detection tasks, offering a benchmark to compare against the more advanced DeepProbCEP framework. This comparison aims to shed light on the additional benefits brought about by the integration of probabilistic logic within DeepProbCEP. Adding a further dimension to our comparative analysis is the LSTM-over-CNN model. This hybrid model ingeniously blends the feature extraction capabilities of Convolutional Neural Networks (CNNs) with the temporal sequence processing strength of LSTM networks. The LSTM-over-CNN model is especially tailored to assess the enhancement in sequence detection brought about by CNN-derived features. This model aims to bridge the gap between traditional LSTM processing and the complex event processing of DeepProbCEP, offering a nuanced perspective on the efficacy of combining convolutional and recurrent neural network architectures in the realm of CEP. Adding a further dimension to our comparative analysis is the LSTM-over-CNN model. This hybrid model ingeniously blends the feature extraction capabilities of Convolutional Neural Networks (CNNs) with the temporal sequence processing strength of LSTM networks. The LSTM-over-CNN model is especially tailored to assess the enhancement in sequence detection brought about by CNN-derived features. This model aims to bridge the gap between traditional LSTM processing and the complex event processing of DeepProbCEP, offering a nuanced perspective on the efficacy of combining convolutional and recurrent neural network architectures in the realm of CEP.

4.1.1 Objective and Expected Outcomes

The primary aim of these experiments is to empirically validate the theoretical constructs of DeepProbCEP and compare its performance with traditional LSTM and the hybrid LSTM-over-CNN models. This endeavor focuses on assessing their effectiveness in complex event processing tasks, particularly in real-world-like scenarios simulated using the MNIST dataset. The objective is to unravel how DeepProbCEP's integration of neural networks with probabilistic logic enhances event processing capabilities compared to the other models. We anticipate the study to provide critical insights into the applicability and strengths of each model in complex event processing. A key

outcome will be showcasing DeepProbCEP's potential in handling challenging scenarios, thereby advancing the field of CEP. The comparative analysis is also expected to highlight the unique advantages and situational suitability of each model, guiding their practical applications. Ultimately, the findings aim to contribute to the broader understanding of combining different neural network architectures for sophisticated data analysis and decision-making in dynamic environments.

4.1.2 Stream of MNIST Digits and Complex Event Definitions for DeepProbCEP

DeepProbCEP represents a significant leap in the field of Complex Event Processing (CEP) by marrying the power of neural networks with the precision of probabilistic logic programming. This section elucidates the mechanisms by which DeepProbCEP capitalizes on this union to enhance complex event detection and reasoning.

The experiments, as presented by Marc Roig Vilamala, showcase DeepProbCEP's prowess in processing continuous streams of MNIST digits. Complex events are defined as consecutive occurrences of the same digit within a variable window size, highlighting the flexibility of DeepProbCEP to adapt to different temporal constraints [50].

Figure 10. DeepProbCEP Architecture for MNIST Digit Stream Processing

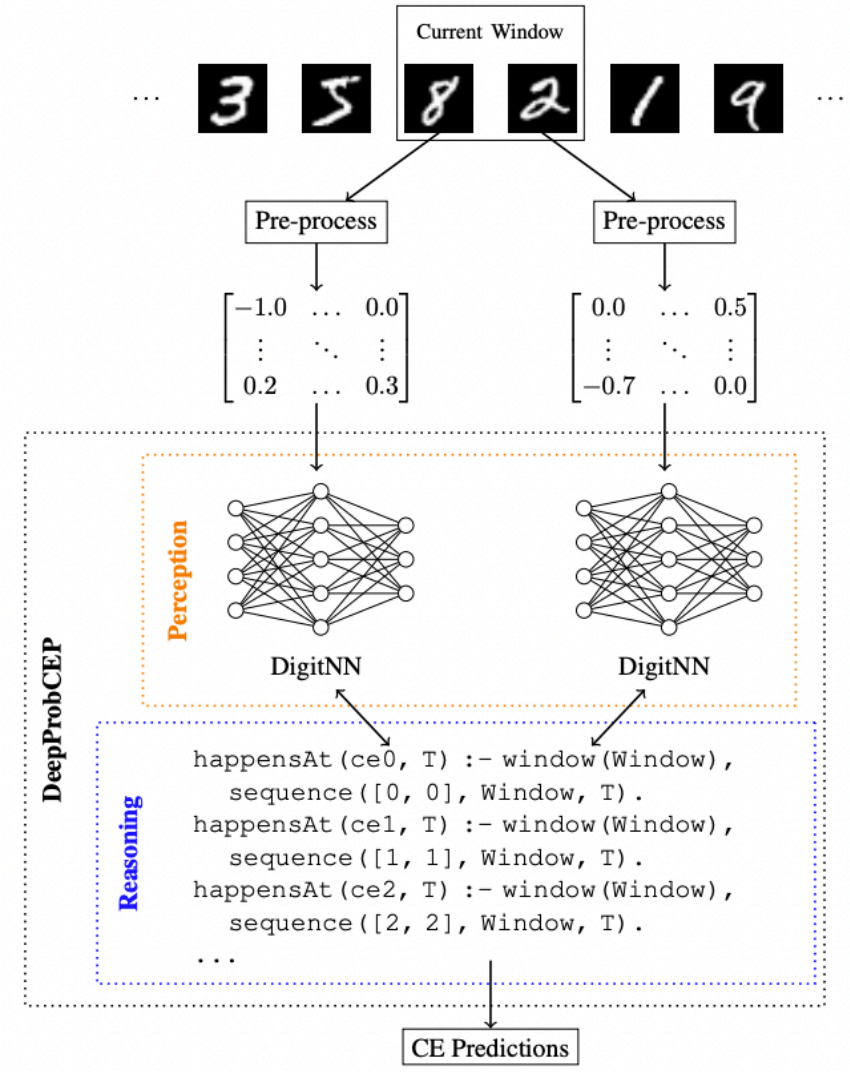


Figure 10 illustrates the DeepProbCEP architecture for MNIST digit stream processing. The

architecture is divided into two main layers:

1. **Data Pre-processing**: The raw digit images from the MNIST dataset are first normalized to ensure consistency across the data inputs. This normalization involves adjusting pixel values to have a mean of 0.5 and a standard deviation of 0.5.

2. **Perception Layer**: The pre-processed images enter the perception layer, where a neural network, referred to as DigitNN, classifies each image. DigitNN is a specialized neural network trained to recognize and classify the ten digits (0-9) based on their visual features extracted from the MNIST images.

3. **Probabilistic Logic Programming Layer**: The output from DigitNN, which consists of probability distributions for digit classifications, feeds into the probabilistic logic programming layer. Here, the DeepProbCEP framework uses rules to determine if complex events—defined as sequential occurrences of the same digit within a moving window—have taken place.

4. **Complex Event Predictions**: The system evaluates the logic programming rules against the sequence of classified digits to predict complex events. For instance, the rule 'happensAt(ce0, T)' checks for consecutive occurrences of the digit '0' within a defined window of time.

Figure 10 showcases the DeepProbCEP's ability to process and reason about temporal sequences of data—a fundamental aspect of CEP. By utilizing neural networks for digit classification and probabilistic logic for temporal pattern detection, DeepProbCEP demonstrates a powerful approach for identifying patterns that are not readily discernible through traditional methods. This capability is essential for real-time decision-making processes in various applications, such as fraud detection, network security, and predictive maintenance.

DeepProbCEP undergoes a rigorous training regimen, aiming to maximize complex event detection within a hundred epochs, incorporating early stopping to mitigate overfitting. Utilizing the Adam optimizer and a learning rate of 0.001, DeepProbCEP fine-tunes DigitNN's weights for optimal performance [**Kingma et al.**, **2014**, 23].

Through a series of experiments, DeepProbCEP has demonstrated its superiority over traditional CEP systems like ProbCEP and ALaSh. Its impressive performance is attributed to its hybrid architecture, which leverages neural networks for pattern extraction and probabilistic logic for rule-based analysis [**Bouchard et al.**, **2015**, 10].

The dual capacity of DeepProbCEP to process both symbolic and non-symbolic data, along with its rule-based reasoning, presents a robust approach for complex event patterns across various domains. The empirical evidence from the experiments underscores the transformative potential of hybrid AI systems in revolutionizing CEP for practical, real-world scenarios.

In this study, we embark on a comprehensive evaluation of three distinct experiments, all derived from the DeepProbCEP repository https://github.com/marcRoigVilamala/DeepProbCEP. These experiments are meticulously designed to test the boundaries and capabilities of the DeepProbCEP framework in processing complex event sequences and operating within noisy environments.

The first experiment focuses on the detection of complex sequences. This task tests each model's ability to discern intricate patterns within continuous data streams, a crucial capability for practical applications in diverse real-world scenarios.

The second experiment delves into the realm of noisy sequences, challenging the models' robustness and accuracy under conditions rife with data uncertainty and interference. This experiment is pivotal in understanding how well the DeepProbCEP framework can maintain performance integrity in less-than-ideal data environments.

To provide a comprehensive analysis, we also conducted a third experiment involving both a simple LSTM model and an LSTM-over-CNN model. This additional comparison serves as a baseline to evaluate the effectiveness of DeepProbCEP against more conventional neural network approaches. This experiment aims to contextualize the performance of DeepProbCEP in relation to established models, thereby highlighting its unique strengths and areas for improvement.

Central to our study is the Prolog-based temporal reasoning mechanism, integral to the DeepProbCEP system. This mechanism is designed for sophisticated event processing in dynamic environments. It is capable of tracking and evaluating the state of various conditions, or 'fluents', across different time points, a functionality that is essential for interpreting and understanding the evolution of sequences over time.

The Prolog code employed in our experiments uses a set of intricately designed rules. These rules form the backbone of our event processing system, encapsulating the complex interplay between events and their temporal properties. In the forthcoming discussion, we will dissect the mechanisms that enable our system to track the initiation, continuation, and alteration of event states. This exploration will offer an in-depth understanding of the inner workings of our complex event processing methodology, demonstrating how it adeptly navigates the temporal intricacies of real-world data sequences.

For detailed insights into the experimental setup, code, and data, interested readers and researchers can access our GitHub repository at https://github.com/VarsouPenny/DeepProbCEP-master. This repository provides a comprehensive resource for replicating and building upon our experimental findings.

Below, we provide an overview of these rules along with their functionalities found in event_defs.pl file for **ce0**, all rules can be found in Appendix.

Listing 4.1. event_defs.pl

```
initiatedAt(sequence0 = true, T) :-
    happensAt(X, T),
    digit(X, 0),
    allTimeStamps(Timestamps),
    previousTimeStamp(T, Timestamps, Tprev),
    Tprev >= 0,
    happensAt(Xprev, Tprev),
    digit(Xprev, 0).

initiatedAt(sequence0 = true, T) :-
    happensAt(X, T),
    digit(X, 0),
    allTimeStamps(Timestamps),
    previousTimeStamp(T, Timestamps, Tprev),
    previousTimeStamp(Tprev, Timestamps, Tprevprev),
    Tprevprev >= 0,
    happensAt(Xprev, Tprevprev),
    digit(Xprev, 0).

initiatedAt(sequence0 = false, T) :-
    happensAt(X, T),
    digit(X, 1),
    allTimeStamps(Timestamps),
    previousTimeStamp(T, Timestamps, Tprev),
    Tprev >= 0,
```

```
    happensAt(Xprev, Tprev),
    digit(Xprev, 1).
```

The first rule asserts that a sequence (**sequence0**) is initiated (i.e., starts) at time **T** if a certain condition is met. Specifically, it checks if an event **X** happens at time T and is identified as the digit '0'. Additionally, it looks for a previous timestamp (**Tprev**) where the same digit '0' occurred. If both these conditions are satisfied, the rule concludes that **sequence0** is initiated at time **T**.

The second rule states that a sequence (**sequence0**) is initiated (i.e., considered to start or be true) at a specific time **T** if certain conditions are met:

- An event **X** occurs at time **T** and is recognized as the digit '0'.
- The timestamps of all events are stored in **Timestamps**.
- The timestamps immediately preceding **T** (t**Tprev**) and the one before **Tprev** (**Tprevprev**) are identified.
- **Tprevprev** must be a valid time (not negative).
- An event **Xprev** occurred at **Tprevprev**, and it is also recognized as the digit '0'.

In essence, this rule detects whether two consecutive '0' digits occur within a defined time frame. The last rule determines that **sequence0** is not initiated (i.e., is false) at time **T** if:

- An event **X** occurs at time **T**, identified as the digit '1'.
- The timestamps of all events are listed in **Timestamps**.
- The timestamp immediately preceding **T** (**Tprev**) is identified.
- **Tprev** must be a valid time (not negative).
- An event **Xprev** occurred at **Tprev**, and it is recognized as the digit '1'.

This rule checks for the presence of two consecutive '1' digits within a defined time frame and concludes that **sequence0** is false under these conditions.

In summary, these rules are used to detect specific sequences of digits ('0' or '1') in a stream of events. They are part of a complex event processing system where the occurrence and order of specific events (digits in this case) are critical in determining the state of a given sequence.

In Problog files we got also prob_ec_cached.pl, that helps determines the state of various fluents (conditions or properties) at different timestampts, all rules can be found in Appendix.

Listing 4.2. prob_ec_cached.pl

```
holdsAt_(aux = true, 0).

holdsAt(F = V, T) :-
  \+ sdFluent(F),
  T @> 0,
  allTimeStamps(Timestamps),
  previousTimeStamp(T, Timestamps, Tprev),
  holdsAt_(F = V, Tprev),
  \+ broken(F = V, Tprev, T).

holdsAt(F = V, T) :-
  \+ sdFluent(F),
```

```
    T @> 0,
    allTimeStamps(Timestamps),
    previousTimeStamp(T, Timestamps, Tprev),
    initiatedAt(F = V, Tprev).

broken(F = V1, T1, T2):-
    allTimeStamps(Timestamps),
    previousTimeStamp(T2, Timestamps, T3),
    initiatedAt(F = V2, T3),
    V1 \= V2.

broken(F = V, T1, T2) :-
    allTimeStamps(Timestamps),
    previousTimeStamp(T2, Timestamps, T3),
    T3 > T1,
    broken(F = V, T1, T3).

previousTimeStamp(T, Timestamps, Tprev):- Tprev is T - 1.
nextTimeStamp(T, Timestamps, Tnext):- Tnext is T + 1.
```

The first **holdsAt** rule initializes a fluent aux to true at time 0. It's a base case for recursion or iterative checks in subsequent rules.

The second **holdsAt** rule checks if a fluent **F** holds a value **V** at time **T**. It applies to fluents that are not self-derived (**sdFluent**). The rule asserts that F = V holds at T if it held at the previous timestamp (**Tprev**) and has not been "broken" (changed or invalidated) between **Tprev** and **T**.

The third **holdsAt** rule determines that a fluent **F** holds a value **V** at time **T** if it was initiated (began to hold) at **Tprev**.

The first **broken** rule defines when a fluent **F's** value **V1** is considered "broken" (no longer holds) between times **T1** and **T2**. It occurs if **F** was initiated with a different value **V2** at a time **T3** before **T2**.

The second **broken** recursive rule further checks for broken fluents between **T1** and **T2** by looking at intermediate timestamps.

Last but not least the **timeStamp** helpers predicates calculate the previous and next timestamps relative to a given time **T**. n summary, this code describes a set of rules for tracking the state of various fluents over time, determining whether they hold true at different points based on their initiation and any changes (broken conditions) that occur in between. This logic is crucial for temporal reasoning in dynamic systems where conditions can change over time.

4.1.3 Data

Data used in this study created by [**Vilamala**, **2022**, 50], based on utilizes a series of structured symbolic events, denoted as **initiatedAt** and **happensAt**, to represent the temporal progression and occurrence of actions in a sequential manner. The training dataset comprises a rich collection of **initiatedAt** predicates, signifying the onset or cessation of various sequences pivotal to the experiments. For instance, $initiatedAt(sequence4 = true, 1)$ indicates the commencement of the fourth sequence at the first timestamp, embedding a temporal structure to the data.

A sample from the training data includes:

Listing 4.3. init_train_data.txt

```
initiatedAt(sequence4 = true, 1).
initiatedAt(sequence2 = true, 14).
```

```
initiatedAt(sequence0 = true, 47).
initiatedAt(sequence2 = false, 14).
initiatedAt(sequence2 = true, 23).
initiatedAt(sequence0 = false, 27).
...
initiatedAt(sequence0 = true, 59991).
```

For testing, similar initiatedAt predicates are employed, ensuring the evaluation is consistent with the training phase. The test dataset echoes the training structure but is used to validate the model's predictions on unseen data.

The Problog training file captures event occurrences with **happensAt** predicates, each tied to specific timestamps, which are crucial for the temporal reasoning tasks the model learns to perform. For example, $happensAt(33273, 0)$ records an event with ID 33273 at the zeroth timestamp. The Problog test file maintains this format, facilitating the direct application of the trained model to assess its generalization capabilities.

A sample from the Problog training data includes:

Listing 4.4. in_train_data.txt

```
happensAt(33273, 0).
happensAt(8310, 1).
happensAt(36168, 2).
happensAt(41016, 3).
happensAt(41817, 4).
...
happensAt(34263, 59999).
allTimeStamps([0, 1, 2,..., 59997, 59998, 59999]).
```

The script used in the experiments created by [**Vilamala**, **2022**, 50], integrating the various components. It employs the training and testing data, defines the neural predicates through the network, and iteratively updates the model using gradient descent, as orchestrated by the optimizer. The script not only executes the training procedure but also systematically evaluates the model's performance, leveraging the structured data to yield insights into the model's predictive accuracy across different noise levels in the data.

4.1.4 LSTM and LSTM-over-CNN Experiment Data

In parallel with the DeepProbCEP experiments, the LSTM and LSTM-over-CNN models were subjected to a similar dataset, ensuring consistency in the comparative analysis. This subsection details the data preparation, preprocessing, and configuration specific to the LSTM and LSTM-over-CNN experiments.

Dataset Description

Both the LSTM and LSTM-over-CNN models were evaluated using the MNIST digit stream, identical to the one used for DeepProbCEP. This dataset, comprising 70,000 images of handwritten digits (0-9), is split into a training set of 60,000 images and a test set of 10,000 images. This uniformity across experiments ensures a direct and fair comparison of results across all models.

Data Preprocessing

For both experiments, the MNIST images underwent preprocessing to meet the input requirements of each respective model. In the LSTM experiment, each 28x28 pixel grayscale image was trans-

formed into a 784-dimensional vector. This conversion was essential for the LSTM's input layer, which processes one-dimensional sequential data.

Similarly, in the LSTM-over-CNN experiment, the preprocessing was tailored to suit the CNN's input specifications. The images were maintained in their original two-dimensional format, suitable for feature extraction by the CNN layers before being fed into the LSTM for sequence processing.

Normalization was applied in both cases, scaling pixel values from 0 to 255 down to a range of 0 to 1. This step is crucial for both models as it aids in the learning process by providing a standardized input format.

Sequence Generation for LSTM

The LSTM model requires sequential data for training, thus sequences of digits were generated from the MNIST dataset. Each sequence, comprising a series of digits, was designed to challenge the LSTM in learning both short-term and long-term dependencies within the streams.

In contrast, the LSTM-over-CNN model processed each image in the sequence through the CNN layers first, extracting salient features. These extracted features were then arranged in sequences to be fed into the LSTM component, combining spatial feature recognition with temporal sequence processing.

Data Splitting

For both models, the dataset was strategically divided into training, validation, and testing sets. This division allowed for the comprehensive training of the models, fine-tuning of hyperparameters during the validation phase, and an unbiased evaluation of each model's performance on the test set. This approach ensured that both models were assessed on their ability to generalize and perform on unseen data, providing a robust evaluation of their capabilities.

4.2 Results

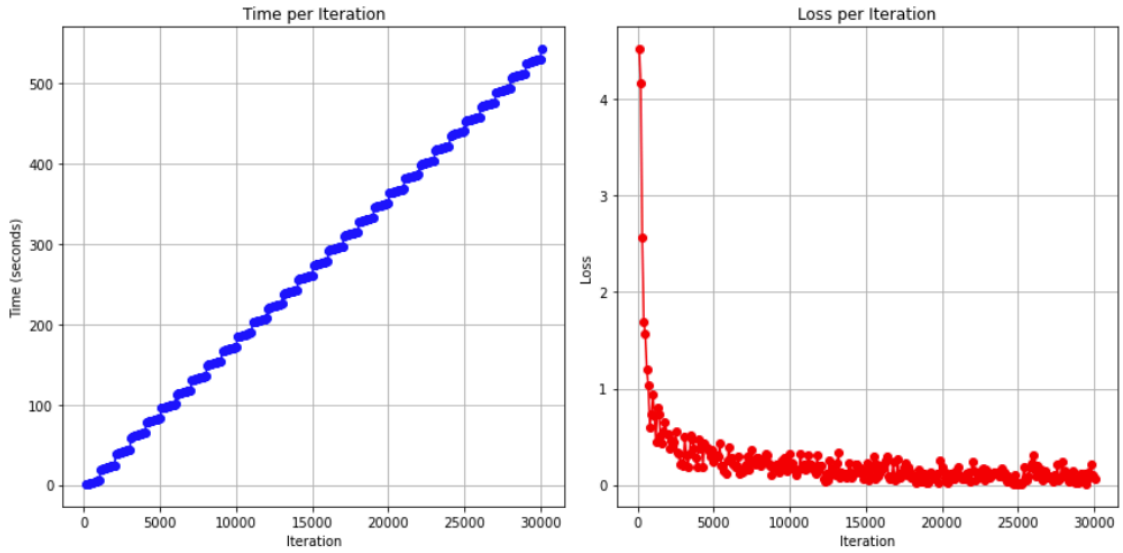
4.2.1 Complex Sequence Detection Results

In the pursuit of evaluating the performance of the complex sequence detection algorithm, we conducted a series of iterative experiments. Figure 11 presents a dual-axis graph comparing the computational time per iteration and the loss per iteration during the training of our complex sequence detection model. The left graph, titled 'Time per Iteration', illustrates a linear increase in computational time as the number of iterations grows. This steady increase suggests a consistent computational load for each iteration, which could be expected if each iteration processes a fixed amount of data or performs a constant number of operations.

The right graph, labeled 'Loss per Iteration', shows a significant decrease in loss in the initial iterations, indicating that the model quickly learned to reduce errors in its predictions. After this initial rapid improvement, the loss levels off, demonstrating that the model has reached a state of convergence where further learning is minimal with respect to loss reduction.

The juxtaposition of these two metrics highlights the efficiency of the learning process: while the time per iteration remains predictable and controlled, the model's loss decreases notably and stabilizes, reflecting a successful training phase where the model's predictive accuracy has likely reached its peak given the current configuration and dataset.

Figure 11. Complex Sequence Detection Performance Plots



Evaluation of Model F1 Scores

Figure 12 illustrates the variation in the average 'F1 initiatedAt' score of our complex sequence detection model across 30,000 iterations. The F1 score, a balanced measure of the model's precision and recall, is a critical indicator of the model's accuracy in sequence prediction tasks.

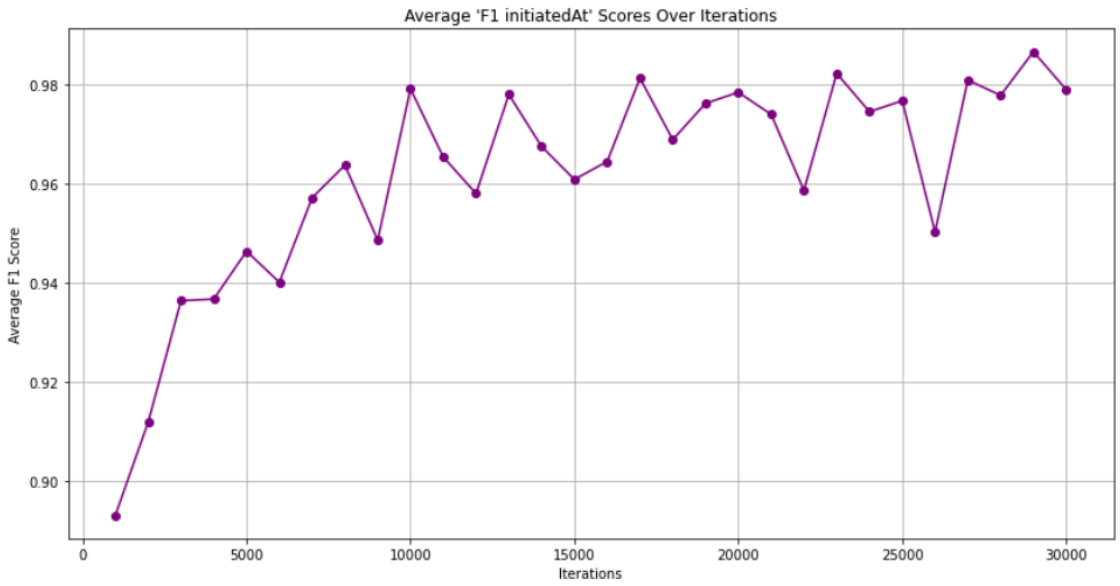
As depicted in the graph, the model's F1 score experiences fluctuations throughout the iterative training process. Initially, there is a steep increase in the F1 score, which suggests that the model rapidly improves its predictive accuracy. Following this ascent, the F1 score displays a series of peaks and troughs, yet it generally maintains an upward trend, indicating progressive learning and adaptation by the model.

Notably, the model's F1 score consistently remains above 0.90, which signifies a high level of performance. The oscillatory pattern might be attributed to the varying complexity of sequences

the model encounters in different iterations or could reflect the model's continuous refinement in response to the intricacies of the data.

In summary, the graph demonstrates the model's ability to sustain a high F1 score across numerous iterations, underscoring its robustness and effectiveness in detecting complex sequences within the dataset.

Figure 12. Complex Sequence Detection F1



Analysis of Model Accuracy Over Iterations

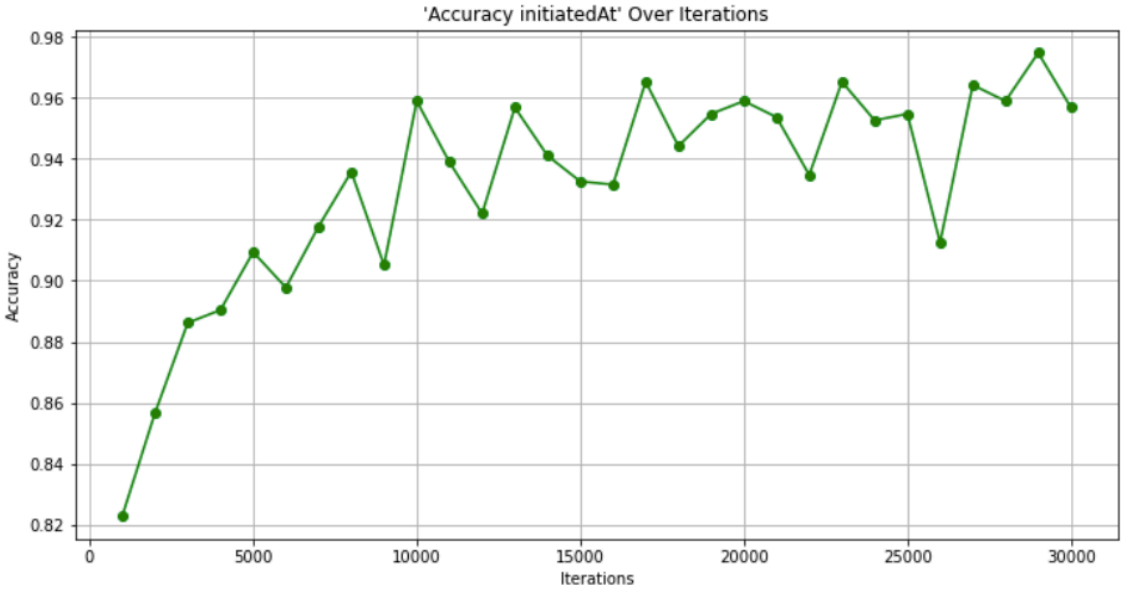
Figure 13 provides an overview of the 'Accuracy initiatedAt' metric as observed over 30,000 iterations during the training phase of our complex sequence detection model. The metric 'Accuracy initiatedAt' quantifies the proportion of sequences correctly identified at the outset of each iteration.

The plot showcases a positive trend in accuracy, starting from below 0.85 and reaching heights just below 0.98, reflecting a substantial improvement in the model's predictive capabilities as it processes more data. This upward trend is punctuated by several dips and rises, which may indicate the model's encounter with particularly challenging data at certain points, or could signify moments of re-adjustment and optimization within the learning algorithm.

Despite these fluctuations, the overall trajectory remains upward, indicating that the model is successfully learning and improving its accuracy over time. The highest peaks on the graph suggest moments where the model is performing at its best, while the troughs represent opportunities for further refinement and improvement.

This graph clearly demonstrates the model's learning progression and its ability to adapt to the complexity of the sequences it is designed to detect, with a general trend towards increased accuracy as training progresses.

Figure 13. Complex Sequence Detection Accuracy



Conclusion

The results presented in the figures illustrate the performance dynamics of the complex sequence detection model over the course of the training iterations. The time per iteration graph shows a consistent linear increase, suggesting that the computational load per iteration is stable and predictable throughout the training process. This is a positive indication of the model's scalability and efficiency.

In terms of model accuracy, the 'Accuracy initiatedAt' metric indicates a general upward trend with some fluctuations, which denotes the model's adaptive learning capability and its potential to refine its predictions when exposed to complex sequences. The F1 score, a harmonic mean of precision and recall, also exhibits an upward trend with some variability, underscoring the model's robustness and effectiveness in balancing the precision-recall trade-off.

The initial rapid decrease in loss and subsequent leveling off further confirm the model's quick adaptation to the training data, followed by a phase of stability where the model has likely achieved a near-optimal state given the architecture and data.

In conclusion, the combination of these metrics—time per iteration, loss per iteration, accuracy, and F1 score—provides a comprehensive picture of the model's learning behavior. The results demonstrate that the model not only learns effectively, achieving high levels of accuracy and F1 scores, but also does so with a computational cost that grows linearly with iterations, which is a desirable trait in practical applications. The slight fluctuations observed across the metrics are typical in machine learning and represent areas for potential model refinement and exploration of parameter tuning or data preprocessing techniques to achieve even higher performance.

4.2.2 Noisy Sequence Detection Results

In this section, we explore the results of our second experiment, which focuses on the detection of sequences within noisy environments—aptly termed 'Noisy Sequence Detection'. The presence of noise in data sequences poses a significant challenge for predictive models, as it can obscure the underlying patterns that are crucial for accurate sequence detection and prediction.

The goal of this experiment was to assess the resilience and adaptability of our model when faced with varying levels of noise in the input data. Noise can come in many forms, such as random

errors, irrelevant information, or distortions that are not representative of the true sequence patterns. A robust model must filter through this noise to reliably identify the sequences of interest.

We introduced controlled noise into our datasets and observed the model's performance in terms of accuracy, precision, recall, and F1 score—metrics that are particularly telling of the model's capacity to discern signal from noise. Through these measures, we can evaluate the model's effectiveness in maintaining high performance despite the degradation of data quality.

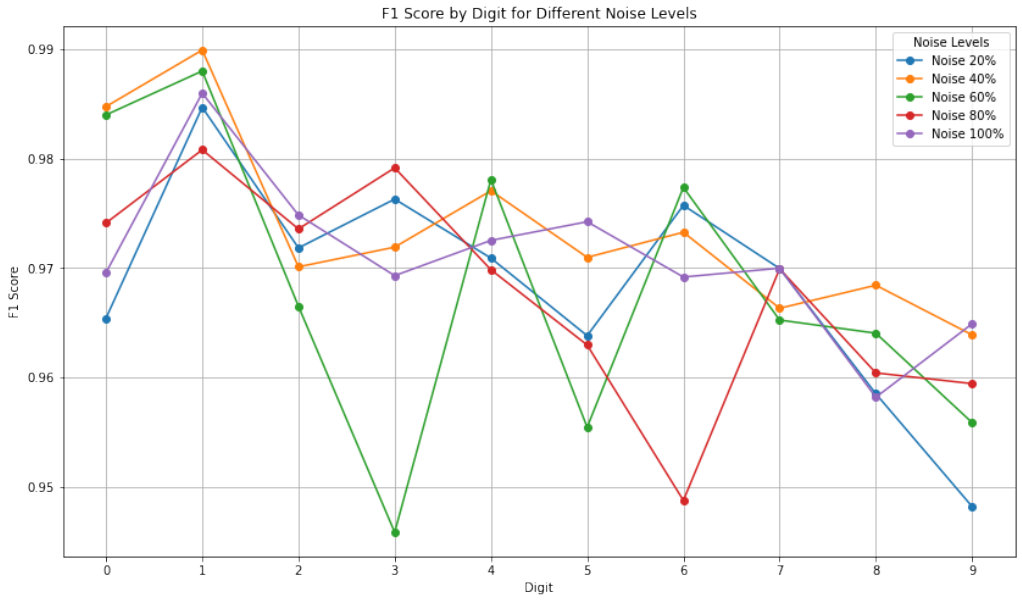
The following subsections detail the experimental setup, describe the nature and level of noise introduced, and present the results in a series of visualizations and tables for two different scenarios, with different level of noise in each (0%, 20%, 40%, 60%, 80%, 100%).

First Scenario - Noise Introduction through Randomness

The first scenario in our experimental framework delves into the introduction of noise through randomness in the labeled MNIST dataset. This setup is designed to mimic common real-world data corruption, where errors occur sporadically and without a discernible pattern. This randomness in noise introduction serves as a fundamental test for the robustness and adaptability of machine learning algorithms in unpredictable environments. To comprehensively evaluate the impact of noise, we conduct tests with varying levels of noisy data: 0% (no noise), 20%, 40%, 60%, 80%, and 100%. This range allows us to observe the performance of the models under gradually increasing noise conditions, providing a nuanced understanding of their noise tolerance.

The primary goal of this approach is twofold: to replicate the types of errors that occur in real-world data collection, such as human labeling errors or system faults, and to assess the capability of learning models to discern correct patterns in the presence of partially incorrect or misleading data. Models that perform effectively under these conditions demonstrate a crucial capacity for handling real-world datasets where perfect labeling is a rarity as shown in Figure 14.

Figure 14. Noisy Sequence Detection F1 for Digit Classification, Scenario 1



The analysis of the following six confusion matrices in Figure 15 from our digit classification experiments provides insightful revelations about the performance of the model under varying conditions. Each matrix represents a unique instance of the model's predictions, illuminating how the model fares in distinguishing between the ten digits (0-9). Across all matrices, a significant concentration of values along the diagonal indicates a high rate of correct predictions. This suggests

that the model, for the most part, is effectively recognizing and classifying the different digits. The presence of off-diagonal values, although minimal in comparison to the diagonal values, points to areas where the model can be improved. However, the overall high accuracy rates across the matrices underscore the robustness of the model in handling digit classification tasks.

First Scenario - Noise Introduction with Memory

In the first scenario of our investigation, we introduce a layer of complexity to the noise model by incorporating a memory component. This scenario simulates a more realistic setting where the noise is not completely random but rather dependent on the historical sequence of events, mirroring real-world situations where the quality of data may deteriorate based on previous occurrences or context. Unlike the first scenario where noise was randomly distributed, here, the noise introduction is conditional. The model is challenged to identify the **initiatedAt** condition despite the noise being influenced by past events. This means that the presence of noise in the current prediction is not independent but is instead correlated with the sequence of labels leading up to that point. Such a condition tests the model's ability to leverage its understanding of temporal dynamics, requiring it not only to learn from the immediate data but also to recall the sequence history to make accurate predictions.

The results from this scenario are particularly illuminating as they demonstrate the model's resilience when faced with noise that has a temporal structure. The confusion matrices from this scenario, as displayed in the thesis, show that while the overall accuracy may drop compared to a noise-free environment, the model still captures the essence of the temporal relationships. This is evidenced by its ability to maintain relatively high F1 scores for certain 'initiatedAt' conditions, even as noise levels increase.

We observed that even in the presence of memory-influenced noise, the model displayed resilience, as demonstrated by the confusion matrices and F1 scores for the 'digit' and 'initiatedAt' conditions see Figure 18 and 19. For instance, with no noise present, the model achieved a high accuracy of 0.9642 and F1 scores consistently above 0.95. Interestingly, the 'initiatedAt' condition retained high F1 scores, such as 0.971 for noise level 0 and impressively reached 1.0 even with noise present, indicating a strong ability to track sequence initiation despite interference

Figure 15. Noisy Sequence Detection F1 for CE, Scenario 1

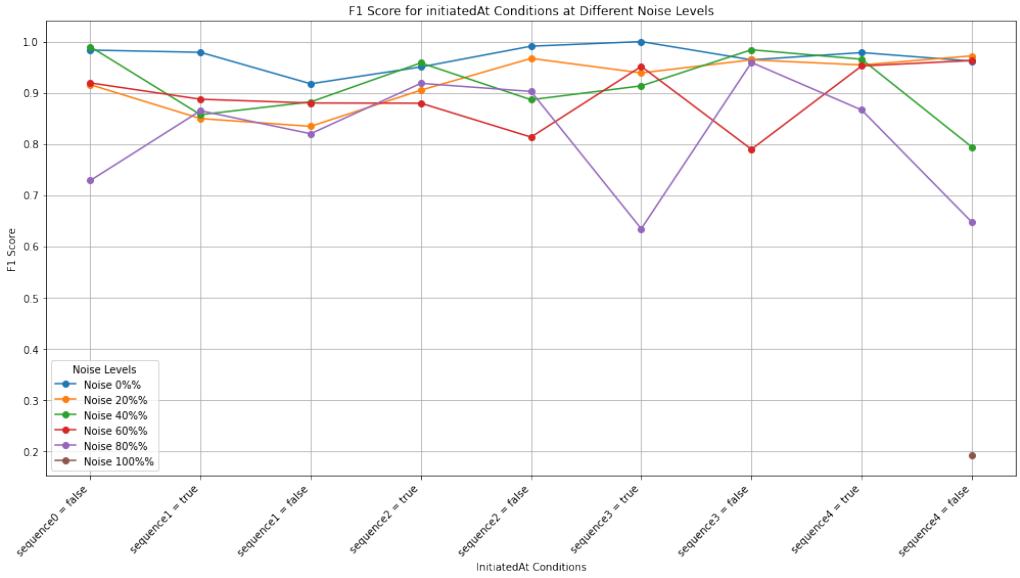
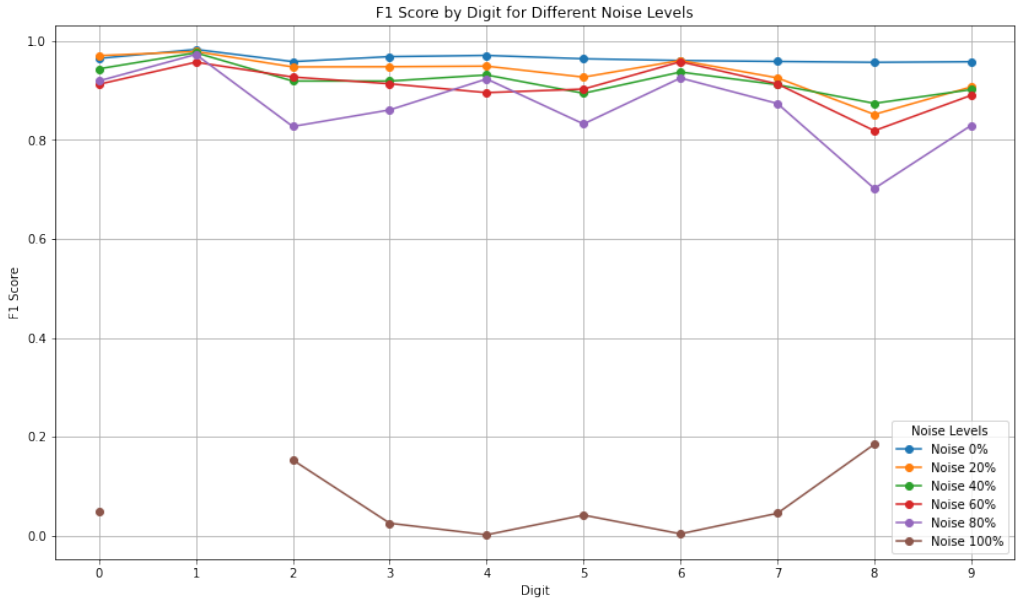


Figure 16. Noisy Sequence Detection F1 for Digit Classification, Scenario 1



As we can see in Figure 20, at lower noise levels (20% and 40%), the model shows a degree of robustness, with a majority of the predictions still being accurate. However, as the noise level rises to 80% and above, this robustness diminishes, and the model's predictions become less reliable.

The performance degradation under noisy conditions has direct implications for real-world applications. It suggests that the model, while potentially useful in controlled environments, may require additional safeguards or compensatory measures to ensure reliability in scenarios where data quality cannot be guaranteed.

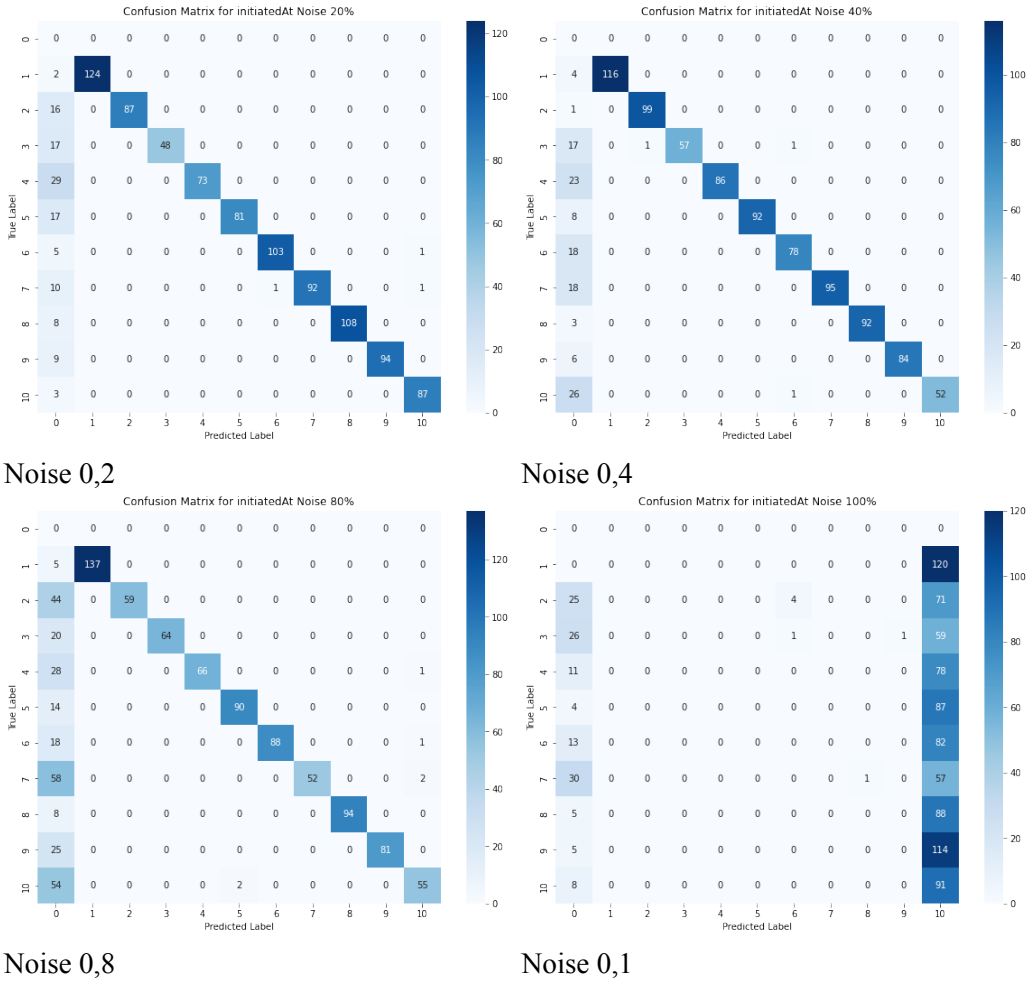


Figure 17. Noisy Sequence Detection Confusions Matrix for CE, Scenario 1

Second Scenario - Structured Noise with Assignment

The second scenario introduces a structured noise model through a predetermined assignment mechanism. This experiment simulates a scenario where noise is introduced not at random but follows a certain pattern or rule that assigns specific noise values to data points based on predefined conditions.

In this set of experiments, we explore the effects of noise that is structured in a way that mimics potential real-world situations where errors or alterations are systematic. By implementing a predefined assignment of noise, we investigate the resilience of our model to recognize and adapt to patterns within the noisy data.

The results from the third scenario, which introduced structured noise with assignment, highlight the model's robustness and ability to discern patterns despite the presence of noise. As observed in the F1 score plots for both digit recognition and initiatedAt conditions at different noise levels (Figure 20, 21), there is a clear trend that the model's performance degrades as noise levels increase. The model demonstrates high precision and recall at lower noise levels, with a notable dip in performance as noise approaches 100%, particularly for certain digits and conditions, signifying a threshold beyond which the model's predictive capabilities are significantly impeded.

Figure 18. Noisy Sequence Detection F1 for initiatedAt, Scenario 3

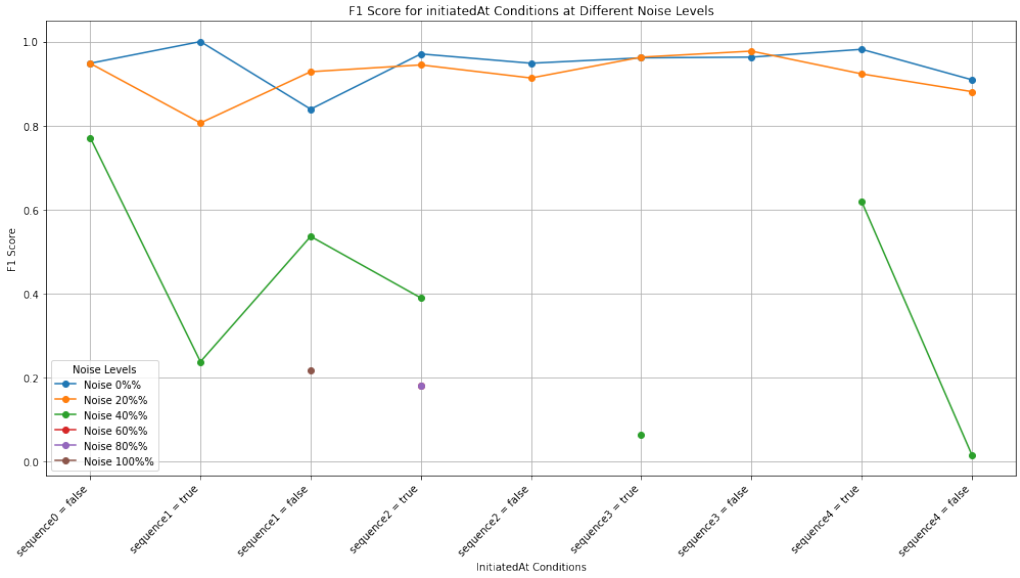
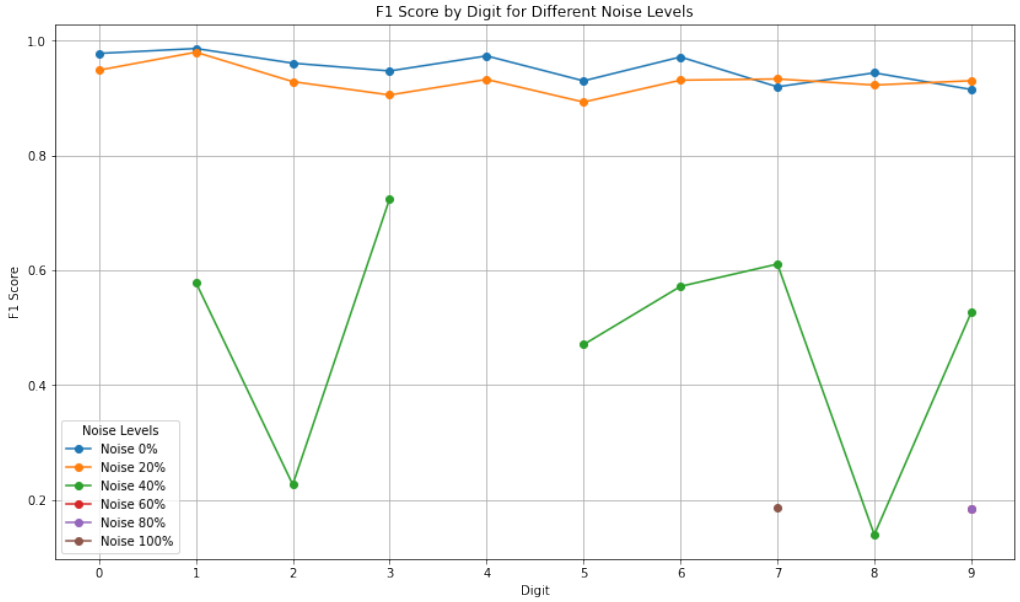


Figure 19. Noisy Sequence Detection F1 Score for Digit Classification, Scenario 3



Structured noise presents a unique challenge as it not only obscures the true signal by introducing random errors but also systematically alters the data based on predefined assignments, which can mislead the model's learning process. This experiment sheds light on the model's limitations in the face of structured noise and underscores the need for more sophisticated algorithms capable of handling such complexities. The model's resilience to noise up to a certain extent indicates the potential for deploying such systems in real-world applications where data may not be pristine.

The empirical findings suggest that while the model can handle noise to a degree, its performance is contingent on the noise level and the nature of the noise introduced. The steep decline in F1 scores at high noise levels suggests that the model heavily relies on the integrity of the data. This underlines the importance of data quality and the necessity to develop noise-robust models.

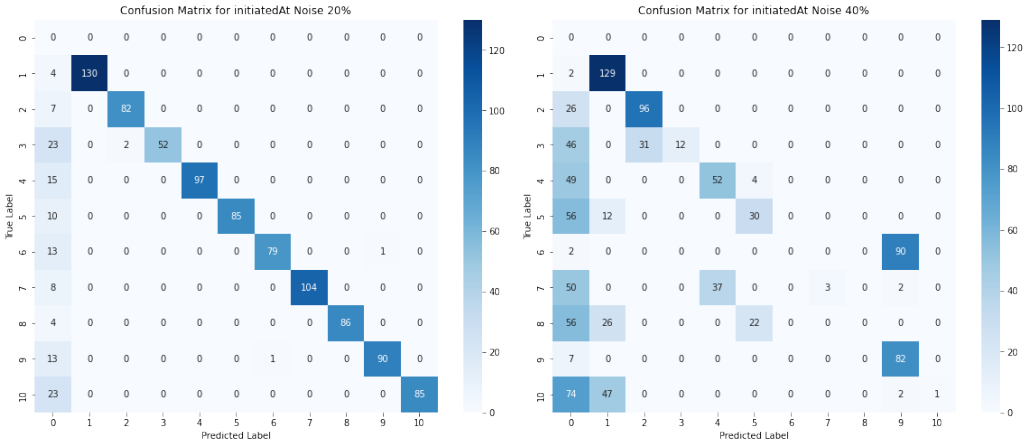


Figure 20. Noisy Sequence Detection Confusions Matrix for CE, Scenario 3

A closer examination to Figure 23 reveals that the model's ability to correctly classify **initiatedAt** events is differentially impacted by the structured noise. Certain events show a resilient classification rate up to moderate noise levels, beyond which the performance drops markedly, as seen in the confusion matrix for 40% noise. This trend is exacerbated at higher noise levels, such as 80% and 100%, where the confusion matrices exhibit significant misclassifications across several complex events.

These findings underscore the model's sensitivity to the systematic distortions introduced by structured noise, particularly when the noise follows a pattern that mimics potential errors in real-world data. While the model retains a degree of accuracy in the face of low to moderate noise, the sharp decline in performance at higher noise levels highlights the limitations of current algorithms

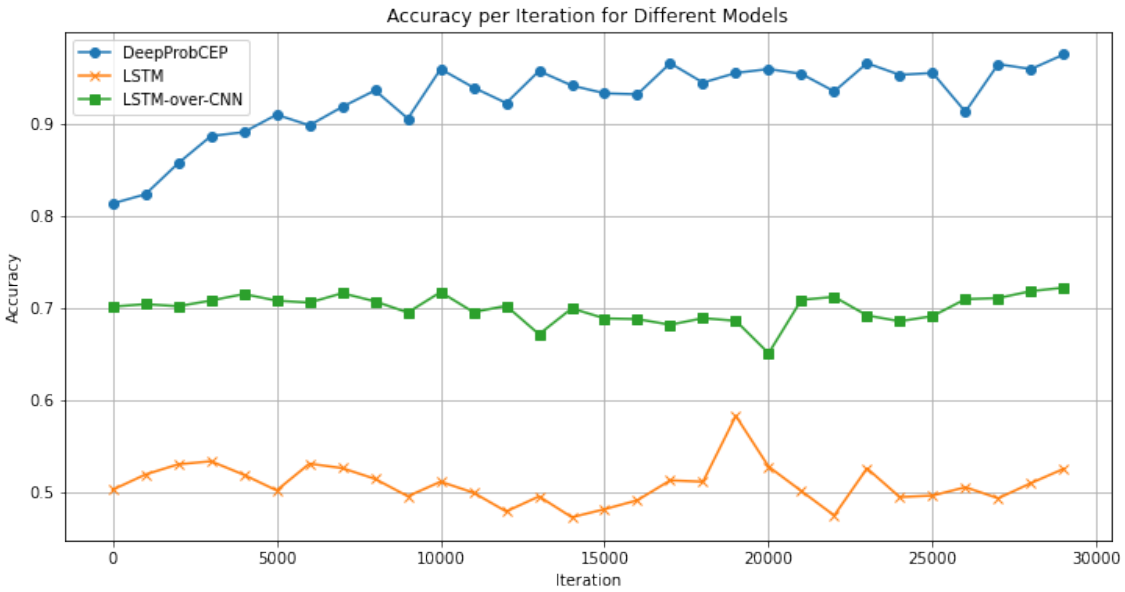
when dealing with structured noise. The confusion matrices also reveal that while some events maintain a high true positive rate, others are more prone to misclassification, suggesting that the model's predictive capabilities are not uniformly affected by noise. This indicates the necessity for targeted improvements in the model's architecture to enhance its robustness against structured noise. Overall, these experiments illustrate the crucial role of data integrity for machine learning models and pave the way for future research aimed at developing algorithms that can withstand the intricacies of structured noise.

Future work should focus on enhancing the model's resistance to noise, perhaps through more advanced noise filtering techniques or by incorporating mechanisms within the learning algorithm that can distinguish between true and noisy data. The overall goal remains to build models that can maintain high accuracy and reliability even when the data is imperfect.

4.2.3 Integrating Complex Sequence Detection with LSTM and LSTM-over-CNN Models

In the realm of complex event processing (CEP), the accurate and efficient detection of intricate sequences in data streams is paramount. This subsection delves into an experimental approach that integrates advanced machine learning techniques - specifically LSTM networks and LSTM-over-CNN models - to enhance sequence detection capabilities. The primary objective of this experiment is to rigorously compare the effectiveness of these integrated models against the DeepProbCEP framework, with a keen focus on two critical performance metrics: **Accuracy** and **F1 Score** presented at Figure 24 and Figure 25.

Figure 21. Complex Sequence Detection Accuracy for all models



As we can observe in Figure 24 accuracy per iteration graph distinctly illustrates the comparative performance of three different models: DeepProbCEP, LSTM, and LSTM-over-CNN. From the visual data, it is apparent that:

- **DeepProbCEP** stands out as the superior model, with accuracy scores consistently above 0.9 throughout the iteration range. This robust performance implies that the model is effectively capturing the underlying patterns within the complex sequence data, leading to reliable predictions.

Figure 22. Complex Sequence Detection F1 Score for all models

- **LSTM-over-CNN** shows intermediate performance with some variability in its accuracy. However, it maintains a level of accuracy that, while not matching DeepProbCEP, suggests a decent understanding of the sequence structure. The convolutional layers may be aiding in feature extraction which, when combined with LSTM's sequence processing, delivers moderate results.

- **LSTM**, on the other hand, demonstrates the lowest accuracy across all iterations. The trend line for LSTM indicates a struggle to adapt to the sequential nature of the data adequately. While there are slight improvements and declines over time, the model's accuracy remains significantly lower than that of DeepProbCEP and LSTM-over-CNN, never crossing the 0.7 threshold. This underperformance could be attributed to a number of factors, including the potential lack of capacity to capture long-term dependencies or the absence of complex feature extraction mechanisms that are present in DeepProbCEP and LSTM-over-CNN architectures.

Also, F1 score is a crucial metric in classification tasks as it balances precision and recall, providing a holistic view of a model's performance, especially when the class distribution is imbalanced. The F1 score plot in Figure 25 for different models over iterations reveals significant disparities in performance among DeepProbCEP, LSTM, and LSTM-over-CNN models.

- **DeepProbCEP** consistently achieves the highest F1 scores, often nearing the ideal score of 1. This suggests that DeepProbCEP not only makes accurate predictions but also maintains a balanced rate of precision and recall, crucial for effective complex sequence detection. The high F1 scores across iterations underline the robustness of DeepProbCEP in handling both relevant and irrelevant elements within the sequence data effectively.

- **LSTM-over-CNN** model demonstrates moderate F1 scores, generally staying within the 0.7 to 0.8 range. This indicates a reasonable trade-off between precision and recall but also room for improvement. It seems that the LSTM-over-CNN model, while not as effective as DeepProbCEP, still manages to capture the sequential dependencies to some extent, possibly benefiting from the feature extraction capabilities of CNN layers before LSTM processes the sequences.

- **LSTM** shows the lowest F1 scores among the three models, with scores fluctuating and at times dropping below 0.6. This performance indicates a significant challenge in achieving a balance between precision and recall. The LSTM model's inability to consistently produce high F1 scores suggests that it might be failing to capture either the full complexity of the data sequences or struggling with specific aspects of the sequence that affect precision or recall.

The comprehensive evaluation of model performance using Accuracy and F1 scores in complex sequence detection tasks has led to several important conclusions. The DeepProbCEP model has consistently outperformed both LSTM and LSTM-over-CNN models across all iterations, as evidenced by its superior accuracy and F1 scores. This suggests that the architecture of DeepProbCEP, likely integrating advanced probabilistic reasoning with sequence modeling, is highly effective for the nuanced task of complex sequence detection.

Chapter 5

Conclusion & Future Work

5.1 Recapitulation

This thesis presented a comprehensive investigation into the integration of neural networks and symbolic logic for complex event recognition (CER), leveraging the DeepProbCEP framework [**Vilamala**, **2022**, 50] to address the complexities of uncertainty and sparse data. Through a series of experiments with varying noise levels, it was convincingly demonstrated that fusing logical reasoning with deep learning significantly enhances model robustness and interpretability. These findings underscore the synergistic potential of symbolic reasoning and statistical methods in AI, paving the way for more versatile and adaptable CER systems.

Key Insights and Contributions:

- Robustness to Uncertainty: The integration of logical reasoning enabled the model to effectively handle noise and uncertainty in input data, leading to improved recognition accuracy and reduced sensitivity to noise levels.
- Interpretability Enhancement: The incorporation of symbolic reasoning provided a deeper understanding of the underlying patterns and relationships within the data, enhancing the interpretability of the model's decisions.
- Synergy of Neural Networks and Logic: The hybrid approach demonstrated the complementary strengths of neural networks and symbolic logic, effectively leveraging pattern recognition capabilities of neural networks with the rule-based reasoning of logic programming.
- Generalization to Diverse Scenarios: The framework's ability to handle uncertainty and sparsity across various noise levels and data types suggests its potential for generalization to diverse real-world CER applications.

Future Directions:

- Scaling for Large-Scale Data: Scaling the proposed methods to handle large, complex datasets encountered in real-world applications is crucial for practical implementation.
- Expressiveness vs. Efficiency Balance: Striking a balance between the expressive power of symbolic reasoning and computational efficiency is essential for efficient and scalable CER systems.

Neuroplex Implementation Challenges

Despite significant efforts, we were unable to successfully implement the Neuroplex algorithm due to technical limitations. However, the insights gained from the DeepProbCEP experiments provide a foundation for future exploration of Neuroplex integration.

Conclusion

The integration of neural networks and symbolic logic has emerged as a promising direction for CER, offering enhanced robustness, interpretability, and generalization capabilities. The presented work demonstrates the effectiveness of this approach and paves the way for further development of advanced CER systems. Future research efforts should focus on scaling these methods for large-scale datasets, optimizing the balance between expressiveness and efficiency, and exploring unsupervised learning techniques for improved data sparsity management.

Appendix A

Problog Rules

A.0.1 evet_defs.pl

```
nn(mnist_net,[X],Y,[0,1,2,3,4,5,6,7,8,9]) :: digit(X,Y).

    initiatedAt(sequence0 = true, T) :-
        happensAt(X, T),
        digit(X, 0),
        allTimeStamps(Timestamps),
        previousTimeStamp(T, Timestamps, Tprev),
        Tprev >= 0,
        happensAt(Xprev, Tprev),
        digit(Xprev, 0).
    initiatedAt(sequence0 = true, T) :-
        happensAt(X, T),
        digit(X, 0),
        allTimeStamps(Timestamps),
        previousTimeStamp(T, Timestamps, Tprev),
        previousTimeStamp(Tprev, Timestamps, Tprevprev),
        Tprevprev >= 0,
        happensAt(Xprev, Tprevprev),
        digit(Xprev, 0).
    initiatedAt(sequence0 = false, T) :-
        happensAt(X, T),
        digit(X, 1),
        allTimeStamps(Timestamps),
        previousTimeStamp(T, Timestamps, Tprev),
        Tprev >= 0,
        happensAt(Xprev, Tprev),
        digit(Xprev, 1).
    initiatedAt(sequence1 = true, T) :-
        happensAt(X, T),
        digit(X, 2),
        allTimeStamps(Timestamps),
        previousTimeStamp(T, Timestamps, Tprev),
        Tprev >= 0,
        happensAt(Xprev, Tprev),
        digit(Xprev, 2).
```

```
initiatedAt(sequence1 = false, T) :-
    happensAt(X, T),
    digit(X, 3),
    allTimeStamps(Timestamps),
    previousTimeStamp(T, Timestamps, Tprev),
    Tprev >= 0,
    happensAt(Xprev, Tprev),
    digit(Xprev, 3).
initiatedAt(sequence2 = true, T) :-
    happensAt(X, T),
    digit(X, 4),
    allTimeStamps(Timestamps),
    previousTimeStamp(T, Timestamps, Tprev),
    Tprev >= 0,
    happensAt(Xprev, Tprev),
    digit(Xprev, 4).
initiatedAt(sequence2 = false, T) :-
    happensAt(X, T),
    digit(X, 5),
    allTimeStamps(Timestamps),
    previousTimeStamp(T, Timestamps, Tprev),
    Tprev >= 0,
    happensAt(Xprev, Tprev),
    digit(Xprev, 5).
initiatedAt(sequence3 = true, T) :-
    happensAt(X, T),
    digit(X, 6),
    allTimeStamps(Timestamps),
    previousTimeStamp(T, Timestamps, Tprev),
    Tprev >= 0,
    happensAt(Xprev, Tprev),
    digit(Xprev, 6).
initiatedAt(sequence3 = false, T) :-
    happensAt(X, T),
    digit(X, 7),
    allTimeStamps(Timestamps),
    previousTimeStamp(T, Timestamps, Tprev),
    Tprev >= 0,
    happensAt(Xprev, Tprev),
    digit(Xprev, 7).
initiatedAt(sequence4 = true, T) :-
    happensAt(X, T),
    digit(X, 8),
    allTimeStamps(Timestamps),
    previousTimeStamp(T, Timestamps, Tprev),
    Tprev >= 0,
    happensAt(Xprev, Tprev),
    digit(Xprev, 8).
initiatedAt(sequence4 = false, T) :-
    happensAt(X, T),
```

```
        digit(X, 9),
        allTimeStamps(Timestamps),
        previousTimeStamp(T, Timestamps, Tprev),
        Tprev >= 0,
        happensAt(Xprev, Tprev),
        digit(Xprev, 9).

    sdFluent( aux ).
```

A.0.2 prob_ec_cached.pl

```
    holdsAt_(aux = true, 0).

    holdsAt(F = V, T) :-
      \+ sdFluent(F),
      T @> 0,
      allTimeStamps(Timestamps),
      previousTimeStamp(T, Timestamps, Tprev),
      holdsAt_(F = V, Tprev),
      \+ broken(F = V, Tprev, T).
    holdsAt(F = V, T) :-
      \+ sdFluent(F),
      T @> 0,
      allTimeStamps(Timestamps),
      previousTimeStamp(T, Timestamps, Tprev),
      initiatedAt(F = V, Tprev).

    %holdsAt(F = V, T):-
    %  \+ sdFluent(F),
    %  T @> 0,
    %  initiatedAt(F = V, Tprev),
    %  Tprev < T,
    %  \+ broken(F = V, Tprev, T). % crisp version contains a cut here

    broken(F = V1, T1, T2):-
      allTimeStamps(Timestamps),
      previousTimeStamp(T2, Timestamps, T3),
      initiatedAt(F = V2, T3),
      V1 \= V2.
    broken(F = V, T1, T2) :-
      allTimeStamps(Timestamps),
      previousTimeStamp(T2, Timestamps, T3),
      T3 > T1,
      broken(F = V, T1, T3).

    previousTimeStamp(T, Timestamps, Tprev):- Tprev is T - 1.
    nextTimeStamp(T, Timestamps, Tnext):- Tnext is T + 1.
```

Bibliographic References

[1] E. Alevizos et al. “Complex Event Recognition Under Uncertainty.” In: *Event Processing, Forecasting and Decision-Making Journal* (2015), pp. 97–103.

[2] E. Alevizos and A. Artikis. *Complex Event Recognition and Forecasting*. 2021. URL: https://cer.iit.demokritos.gr/events/cerf21/.

[3] D. Anicic et al. “A Rule-Based Language for Complex Event Processing and Reasoning.” In: *Web Reasoning and Rule Systems Conference Proceedings*. Springer Berlin Heidelberg, 2010, pp. 42–57.

[4] Alexander Artikis, Alessandro Margara, Matias Ugarte, Stijn Vansummeren, and Matthias Weidlich. “Tutorial: Complex Event Recognition Languages.” In: *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems*. ACM, 2017, pp. 7–10.

[5] Alexander Artikis, Georgios Paliouras, François Portet, and Anastasios Skarlatidis. “Logic-based Representation, Reasoning and Machine Learning for Event Recognition.” In: *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems*. ACM, July 2010, pp. 282–293.

[6] Alexander Artikis, Anastasios Skarlatidis, François Portet, and Georgios Paliouras. “Logic-Based Event Recognition.” In: *The Knowledge Engineering Review* 27.4 (2012), pp. 469–506.

[7] Yoshua Bengio, Aaron Courville, and Pascal Vincent. “Representation Learning: A Review and New Perspectives.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.8 (2013), pp. 1798–1828.

[8] Jonathan R. Binder and Rutvik H. Desai. “The Neurobiology of Semantic Memory.” In: *Trends in Cognitive Sciences* 15.11 (2011), pp. 527–536.

[9] Richard J. Bolton and David J. Hand. “Statistical Fraud Detection: A Review.” In: *Statistical Science* 17.3 (2002), pp. 235–255.

[10] Guillaume Bouchard et al. “From NIPS to MLJ: Reviewing Papers for Machine Learning Conferences and Journals.” In: *Machine Learning Journal* (2015). Add additional details such as volume, number, pages, DOI if available.

[11] Gianni Cugola and Alessandro Margara. “Complex Event Processing with T-REX.” In: *Journal of Systems and Software* 85.8 (2012), pp. 1709–1728.

[12] Luc De Raedt and Angelika Kimmig. “Probabilistic Logic Programming: A Survey.” In: *ACM Computing Surveys* 47.4 (2015), Article 31.

[13] 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 (IJCAI’07)*. 2007.

[14] Opher Etzion and Peter Niblett. *Event Processing in Action*. Manning Publications, 2010.

[15] Jerome A. Feldman. *From Molecule to Metaphor: A Neural Theory of Language*. Cambridge, Mass.: A Bradford Book, 2006.

[16] Marcio V. França, Gerson Zaverucha, and Artur S. d'Avila Garcez. "Fast Relational Learning Using Bottom Clause Propositionalization with Artificial Neural Networks." In: *Machine Learning* 94 (2014), pp. 81–104.

[17] Artur D'Avila Garcez, Thomas R. Besold, Luc De Raedt, Peter Földiak, Pascal Hitzler, Thomas Icard, et al. "Neural-Symbolic Learning and Reasoning: Contributions and Challenges." In: *2015 AAAI Spring Symposium Series*. Mar. 2015.

[18] Lise Getoor and Ben Taskar, eds. *Introduction to Statistical Relational Learning*. MIT Press, 2007.

[19] Nikos Giatrakos, Elias Alevizos, Alexander Artikis, Antonios Deligiannakis, and Minos Garofalakis. "Complex Event Recognition in the Big Data Era: A Survey." In: *The VLDB Journal* 29 (2020), pp. 313–352.

[20] José Roldán Gómez, Juan Boubeta-Puig, and Guadalupe Ortiz. "Integrating complex event processing and machine learning: An intelligent architecture for detecting IoT security attacks." In: *Expert Systems with Applications* 149 (July 2020). Corpus ID: 213728291, p. 113251. DOI: 10.1016/j.eswa.2020.113251. URL: https://doi.org/10.1016/j.eswa.2020.113251.

[21] Bernd Gutmann, Ingo Thon, and Luc De Raedt. "Learning the Parameters of Probabilistic Logic Programs from Interpretations." In: *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*. Springer. 2008, pp. –.

[22] Hiroki Kawashima, Hiroyuki Kitagawa, and Xiang Li. "Complex Event Processing over Uncertain Data Streams." In: *2010 International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*. IEEE, 2010, pp. 521–526.

[23] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization." In: *International Conference on Learning Representations*. Add additional details such as pages, DOI if available. 2014.

[24] Robert A. Kowalski. *Logic for Problem Solving*. North-Holland, 1979.

[25] Markus Krötzsch, Frederick Maier, Adila Krisnadhi, and Pascal Hitzler. "A Better Uncle for OWL: Nominal Schemas for Integrating Rules and Ontologies." In: *Proceedings of the 20th International Conference on World Wide Web*. ACM, Mar. 2011, pp. 645–654.

[26] Brenden M. Lake, Tomer D. Ullman, Joshua B. Tenenbaum, and Samuel J. Gershman. "Building Machines That Learn and Think Like People." In: *Behavioral and Brain Sciences* 40 (2017), e253.

[27] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep Learning." In: *Nature* 521 (2015), pp. 436–444.

[28] Vladimir Lifschitz. "Answer Set Programming and Plan Generation." In: *Artificial Intelligence* 138.1-2 (2002), pp. 39–54.

[29] J. Liu et al. "A Survey of Human Action Recognition Based on Visual Data." In: *Journal of Visual Communication and Image Representation* 55 (2018), pp. 340–353.

[30] John W. Lloyd. *Foundations of Logic Programming*. 2nd ed. Springer-Verlag, 1987.

[31] David C. Luckham. *The Power of Events*. Addison-Wesley, 2002.

[32] Robin Manhaeve, Sebastijan Dumančić, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. "Neural Probabilistic Logic Programming." In: *Advances in Neural Information Processing Systems*. Vol. 31. 2021, pp. 3753–3763.

[33] Robin Manhaeve, Sebastijan Dumančić, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. "Neural Probabilistic Logic Programming in DeepProbLog." In: *Artificial Intelligence* 298 (2021), p. 103504.

[34] Gary Marcus and Ernest Davis. *Rebooting AI: Building Artificial Intelligence We Can Trust*. Pantheon Books, 2019.

[36] Robert Miller and Murray Shanahan. *The Event Calculus in Classical Logic - Alternative Axiomatisations*. Tech. rep. Linköping University Electronic Press, 1999.

[37] Liu Ming, Elke A. Rundensteiner, Kajal T. Greenfield, Chetan Gupta, Song Wang, Ismail Ari, and Abhay Mehta. "E-Cube: Multi-Dimensional Event Sequence Processing Using Concept and Pattern Hierarchies." In: *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*. IEEE, Mar. 2010, pp. 1097–1100.

[38] R. Keith Mobley. *An Introduction to Predictive Maintenance*. Butterworth-Heinemann, 2002.

[39] Alexandros Oikonomakis. "Neuro-symbolic Answer Set Programming for Human Activity Recognition in Videos." PhD thesis. University of Piraeus, 2023.

[40] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. "Automatic Differentiation in PyTorch." In: *NIPS Workshop*. 2017.

[41] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, 1988.

[42] Claudio Piciarelli and Gian Luca Foresti. "Online Trajectory Clustering for Anomalous Events Detection." In: *Pattern Recognition Letters* 27.15 (2006), pp. 1835–1842.

[43] Ronald Poppe. "A Survey on Vision-Based Human Action Recognition." In: *Image and Vision Computing* 28.6 (2010), pp. 976–990.

[44] Fabrizio Riguzzi and Terrance Swift. "The PITA System: Tabling and Answer Subsumption for Reasoning under Uncertainty." In: *Theory and Practice of Logic Programming* 13.4-5 (2013).

[45] H. Salehi and R. Burgueño. "Emerging artificial intelligence methods in structural engineering." In: *Engineering Structures* 171 (Sept. 2018), pp. 170–189. DOI: 10.1016/J.ENGSTRUCT.2018.05.084.

[46] Zhenyu Shen, Hiroshi Kawashima, and Hiroyuki Kitagawa. "Probabilistic Event Stream Processing with Lineage." In: *Proceedings of Data Engineering Workshop*. 2008, pp. 61–64.

[47] Anastasios Skarlatidis et al. "Probabilistic Complex Event Recognition: A Survey." In: *ACM Computing Surveys* 48.4 (2015), Article 53.

[48] Robin Sommer and Vern Paxson. "Outside the Closed World: On Using Machine Learning for Network Intrusion Detection." In: *IEEE Symposium on Security and Privacy*. IEEE. 2010.

[49] M. R. Vilamala, T. Xing, H. Taylor, L. Garcia, M. Srivastava, L. Kaplan, and F. Cerutti. *Using DeepProbLog to perform complex event processing on an audio stream*. 2021. eprint: arXiv:2110.08090.

[50] Marc Roig Vilamala. "Combining Neural Networks with Symbolic Approaches to perform Complex Event Processing on Non-Symbolic Data." PhD thesis. Cardiff University, July 2022.

[51] T. Xing, L. Garcia, M. R. Vilamala, F. Cerutti, L. Kaplan, A. Preece, and M. Srivastava. "Neuroplex: learning to detect complex events in sensor networks through knowledge injection." In: *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*. Nov. 2020, pp. 489–502.

[52] Wei Xu and et al. "Detecting Large-Scale System Problems by Mining Console Logs." In: *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*. ACM, 2009.

[53] Maor Yankovitch, Ilya Kolchinsky, and Assaf Schuster. "HYPERSONIC: A Hybrid Parallelization Approach for Scalable Complex Event Processing." In: *Proceedings of the 2022 International Conference on Management of Data*. SIGMOD '22. New York, NY, USA: ACM, June 2022, pp. 1093–1107. DOI: 10.1145/3514221.3517829. URL: https://doi.org/10.1145/3514221.3517829.