Expert Systems With Applications 215 (2023) 119376

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

DeepProbCEP: A neuro-symbolic approach for complex event processing in adversarial settings

Marc Roig Vilamala [a,*], Tianwei Xing [b], Harrison Taylor [a], Luis Garcia [c], Mani Srivastava [b], Lance Kaplan [d], Alun Preece [a], Angelika Kimmig [e], Federico Cerutti [a,f]

[a] Cardiff University, United Kingdom
[b] University of California, Los Angeles, United States
[c] University of Southern California, Information Sciences Institute, United States
[d] US DEVCOM Army Research Laboratory, United States
[e] KU Leuven, Department of Computer Science, Leuven.AI, Belgium
[f] University of Brescia, Italy

ARTICLE INFO

Dataset link: https://codeocean.com/capsule/7055867/tree/v1

Keywords:
Complex Event Processing
Neuro-symbolic architecture
Learning with sparse data
Robustness

ABSTRACT

Detecting complex events from subsymbolic data streams (such as images, audio recordings or videos) is a challenging problem, as traditional symbolic approaches cannot be used to process subsymbolic data, and neural-only approaches usually require larger amounts of training data than available. In this paper, we present DeepProbCEP, a Complex Event Processing (CEP) approach designed with four objectives: (i) allowing the use of subsymbolic data as an input, (ii) retaining flexibility and modularity in the definition of complex event rules, (iii) limiting the cost of obtaining training data and (iv) being robust against adversarial conditions. DeepProbCEP archives this by using a neuro-symbolic approach, which combines the neural and symbolic approaches to allow training with sparse data. This is made possible through the injection of human knowledge. In this paper, we demonstrate that DeepProbCEP outperforms other state-of-the-art approaches when training using sparse data. We also show that DeepProbCEP is robust in different adversarial settings. Finally, DeepProbCEP's flexibility is demonstrated by showing it can be used to process both images and audio as input.

1. Introduction

Complex Event Processing (CEP) systems process data streams and detect situations of interest, or *complex events*, which aggregate atomic *events*, or *simple events*. CEP systems detect complex events by identifying the spatio-temporal relationships between sets of simple events. CEP systems have been applied in many different areas, such as business activity monitoring (Teymourian, Rohde, & Paschke, 2012), sensor networks (Anicic, Rudolph, Fodor, & Stojanovic, 2012b) and weather reports (Anicic, Rudolph, Fodor, & Stojanovic, 2012a). Most CEP approaches allow the user to define rules which express the conditions under which a complex event occurs. Then, the CEP system uses those rules to detect when those circumstances happen in the given stream of input data. However, defining rules over raw streams of data can be challenging. For example, it is not feasible to define rules directly over raw images, audio or videos, which makes it impossible to use most CEP approaches. This paper proposes an approach capable of performing CEP on types of data for which symbolic rules cannot be (easily) manually defined. We refer to such types of data as *subsymbolic data*.

Some approaches (Xing et al., 2020) allow the user to train the system to detect complex events based on subsymbolic data. However, current implementations might benefit from extra engineering for offering greater flexibility and modularity when defining the rules for the complex events. Other approaches (Roig Vilamala, Hiley, Hicks, Preece, & Cerutti, 2019; Roldán, Boubeta-Puig, Luis Martínez, & Ortiz, 2020) have been created to allow subsymbolic inputs without limiting the flexibility of rule definitions by using pre-trained neural networks to parse this input into a symbolic form. However, if such neural networks are not available, these approaches are not possible. In such cases, the neural networks could be trained separately. However, it can be costly to obtain training data for this purpose. In this paper, we propose an

* Correspondence to: Crime and Security Research Institute, sbarc|spark, Maindy Rd, Cardiff CF24 4HQ, Wales, United Kingdom.
E-mail addresses: RoigVilamalaM@cardiff.ac.uk (M. Roig Vilamala), twxing@ucla.edu (T. Xing), harritaylor@protonmail.com (H. Taylor), lgarcia@isi.edu (L. Garcia), mbs@ucla.edu (M. Srivastava), lance.m.kaplan.civ@army.mil (L. Kaplan), PreeceAD@cardiff.ac.uk (A. Preece), angelika.kimmig@cs.kuleuven.be (A. Kimmig), federico.cerutti@unibs.it (F. Cerutti).

https://doi.org/10.1016/j.eswa.2022.119376
Received 26 December 2021; Received in revised form 21 November 2022; Accepted 26 November 2022
Available online 1 December 2022

approach that can be trained to use new types of subsymbolic data without limiting the flexibility and modularity of the rule definitions.

As defined in current AI ethical principles (Defense Innovation Board, 2019), approaches must also be robust against adversarial conditions. These indicate that AI systems should be reliable, meaning that the systems should behave as expected even in sub-optimal conditions. For this paper, we focus on situations in which the training data is noisy, meaning that some of the labels in the training data are incorrect.

From the preceding discussion, it is evident that a CEP system must fulfil four objectives:

1. Being able to operate on input streams of subsymbolic data.
2. Retaining flexibility and modularity in the definitions of complex events.
3. Limiting the cost of obtaining training data and labelling.
4. Being robust against adversarial conditions.

Currently, none of the approaches in the literature cover all four objectives. The contribution of this paper is DeepProbCEP,[1] a neuro-symbolic architecture designed to fulfil all four objectives. DeepProbCEP combines a neural network with symbolic rule definitions: the neural network processes subsymbolic data, fulfilling the first objective. Then, a probabilistic logic layer encompasses the complex event rules, which provides us with the flexibility and modularity expressed in the second objective.

The third objective expresses the aim to limit the costs of obtaining training data. As such, training data that is easier to obtain and label will be preferred. One of the ways in which DeepProbCEP archives this is by allowing end-to-end training. This means that the whole system can be trained using only labels for the complex events, which are easier to label than the simple events. For example, it would be much easier for the people labelling to recognize when the situation of interest (the complex event) happens in a given video than to label every frame or short segment of that same video (the simple events). DeepProbCEP also provides higher accuracy results with fewer training data than state-of-the-art approaches, as demonstrated by our experiments.

In order to evaluate how robust DeepProbCEP is (fourth objective), we have generated a number of synthetic datasets that simulate different types of poison attacks, where the training data contains noisy labels. DeepProbCEP's performance after training with such datasets demonstrates its robustness against these types of attacks.

1.1. Motivating example

As a motivating example, suppose that a user is trying to detect concerning situations from a microphone, situated in a busy street. CEP could be used to define the types of situations that could be a concern. For instance, we could define that the sounds of children playing and street music are part of a safe environment, whereas sounds like sirens or gunshots are concerning.

Then, complex events could be defined based on combinations of the sounds described above. For instance, we could define the complex event *worrying siren* to happen when the microphone detects a siren just after the sounds of a safe environment. We could then define that a return to the safe sounds stops the *worrying siren* complex event. On the other hand, the presence of more concerning sounds (such as sirens or gunshots) would increase our confidence on saying that the complex event *worrying siren* is happening. DeepProbCEP has been used to create a demonstration for this exact scenario based on real life recordings of a street (Roig Vilamala, 2022). While we do not go into details on the implementation of this demonstration in this paper, the training performed for it works in the same manner as the experiments shown below.

[1] Code is available at https://github.com/dais-ita/DeepProbCEP.

The rest of the paper is structured as follows: Section 2 contains a critical analysis of related work. Section 3 explains how DeepProbCEP was implemented using those tools. Section 4 describes the scenarios we aim to emulate, and how we have generated synthetic datasets for that purpose. Section 5 explains how the experiments were designed, while Section 6 discusses the results obtained in those experiments. Section 7 provides a discussion of our approach and its limitations, while Section 8 discusses possible areas for future work taking into account such limitations, as well as providing the conclusion to the paper.

2. Related work

Complex Event Processing (CEP) is a technology that allows for the derivation of conclusions from the events present in a stream of information (Burgueño, Boubeta-Puig, & Vallecillo, 2018). In order to derive these conclusions, CEP allows for the definition of complex events based on the events produced by the input streams, which allow the system to identify the situations of interest—that is, the complex events—automatically. The process through which the user defines these complex events can change significantly depending on the exact approach being used, ranging from graphical tools (Roldán et al., 2020), to SQL-like languages (Bezerra, Teles, Coutinho, & da Silva e Silva, 2021; Burgueño et al., 2018), logic rules (Roig Vilamala et al., 2019; Skarlatidis, Artikis, Filippou, & Paliouras, 2015) and even languages specifically designed to define complex events (Chapnik, Kolchinsky, & Schuster, 2021; Yankovitch, Kolchinsky, & Schuster, 2022). However, despite the varied implementations and the changes in the specifics, all CEP approaches generally follow the same principles where a complex event is defined based on combinations of events.

To differentiate between approach types, we have decided to divide CEP into two groups: declarative and data-driven. The declarative approaches to CEP provide rules that can be used to define specifically when complex events occur. The types of input data that they can use tend to be relatively limited: often these rules need to directly process the input data, and that can be difficult for subsymbolic types of data, such as images, audio or video.

The data-driven approaches, instead, are designed to work with data for which it is hard to directly define rules. They however are less flexible than the declarative approaches in accommodating user-defined CEP patterns. Some remove such a possibility completely: while beneficial in some cases, it does make it harder to detect complex events with more composite definitions.

2.1. Declarative approaches to CEP

In declarative approaches to CEP, users are expected to provide aggregation rules by specifying patterns over input data that signify either the beginning and the end of complex events, or the complex events themselves (Alevizos, Skarlatidis, Artikis, & Paliouras, 2017; Giatrakos, Alevizos, Artikis, Deligiannakis, & Garofalakis, 2020). As our goal is not to provide articulated pattern recognition systems, we rely on fairly simple pattern matching capabilities.

Such aggregation rules can, in some circumstances, be learnt. For instance, in Bruns, Dunkel, and Offel (2019) genetic programming is used to learn CEP rules encoding them as syntax trees. Using this approach, the authors show that their system can learn rules of varying complexity with very good accuracy. However, such an approach falls short of addressing the type of complexity required to deal with high-bandwidth subsymbolic data streams.

Some CEP approaches use a neural network to transform high-bandwidth data into symbolic information, allowing the user to define rules on it. For example, in Roldán et al. (2020) the authors show that this allows them to reduce the number of false positives in a system when detecting IoT security attacks. They use a neural network to predict the length of the suspected packets. If the predicted length does

not match the actual length of the packet, a complex event is generated indicating that an attack might be happening. In Roig Vilamala et al. (2019), we present a system that is able to detect different violent activities from a CCTV feed. A pre-trained neural network is used to process short segments of video (16 frames, about half a second) detecting potential violent acts. Another pre-trained neural network is used to detect people from the same video feed. The outputs of these neural networks are then combined using a probabilistic logic program to detect the complex, violent events. These complex events are defined in a manner inspired by event calculus, based on the approach used in Skarlatidis et al. (2015), identifying the conditions for a complex event to initiate and terminate.

Both Roig Vilamala et al. (2019) and Roldán et al. (2020) use pre-trained neural networks to parse the simple events. In this paper, instead, we assume that no such pre-trained neural networks exist. As such, we assume that only end-to-end training is possible, meaning that only training labels for the complex events are available, and not for the simple events. While this does make the training problem harder, it is undeniably easier to obtain labels for the complex events, thus reducing the costs associated with the creation of the training set.

2.2. Data-driven approaches to CEP

The data-driven approaches are designed to work with data for which it is hard to directly define rules. Here we review two notable baselines: (i) neural-only approaches—that is, approaches that rely solely on neural networks— and (ii) an approach embedding aggregation rules in a neural architecture.

2.2.1. Neural-only approaches

One possible approach is to view the whole CEP problem as a classification problem, and—for instance—use neural networks to detect when complex events occur. For this paper, these approaches are referred to as neural-only. These approaches remove the manual definitions of complex events, and instead attempt to train the neural network to identify those definitions at the same time as it learns to classify the subsymbolic data. Due to the relevance of time in the definition of complex events, a Long Short Term Memory (LSTM) (Mishra, Jain, Siva Naga Sasank, & Hota, 2018) or a Convolutional 3D layer (C3D) (Liu, Liu, Gan, Tan, & Ma, 2018) can be used. However, due to the necessity of learning the complex event rules, these approaches need very large amounts of data to train. Furthermore, the complexity of the rules that define the complex events is limited, due to the fact that the neural networks need to learn those rules. While it is possible to train these systems to identify complex events with relatively simple definitions, particularly complex and specific definitions would likely require infeasible amounts of training data.

2.2.2. Embedding aggregation rules in a neural architecture: Neuroplex

The current state of the art in CEP with subsymbolic data is Neuroplex (Xing et al., 2020). Neuroplex is a neuro-symbolic approach that makes use of human knowledge in order to reduce the amount of training data required when compared to neural-only approaches. This is done by dividing the problem into two levels; low-level perception and high-level reasoning. The high-level reasoning is responsible for detecting complex events based on manually defined rules, while the low-level perception is responsible for abstracting the subsymbolic data into symbolic concepts that can be used by rules later. In Neuroplex, the user defines the rules for the complex events. Then, a neural network, called NRLogic, is trained to emulate a *logic layer* that recognizes those rules through a process called *knowledge distillation* (Hinton, Vinyals, & Dean, 2015; Hu, Ma, Liu, Hovy, & Xing, 2016). This neural network can be used as the logic layer for high-level reasoning, and can be combined with another neural network, which performs the task of low-level perception. During the training, the NRLogic network for high-level reasoning is frozen, meaning that the weights for this model will not be modified by further training. In this way, the system is trained in an end-to-end manner, using only labels of the complex events. This trains the low-level perception neural network to abstract the input subsymbolic data into symbolic concepts, in the form of the output classes from the perception neural network. This is done without requiring any annotations for these classes, meaning that no training labels for the simple events are required.

The use of knowledge distillation allows Neuroplex to reduce the amount of data required to train the system. While the architecture of the neural networks may be very similar to approaches such as Al-Rakhami et al. (2021), Islam, Islam, and Asraf (2020) and Shi, Bai, and Yao (2015)—that is, combining CNN and RNN layers to form a neural network—Neuroplex requires significantly less training data. This is thanks to the injection of human knowledge in the form of manually defined rules, as this simplifies the problem into two smaller parts, which can be trained individually. This makes Neuroplex the state-of-the-art for CEP on subsymbolic data, particularly when dealing with data scarcity (Xing et al., 2020).

It is important to note that this approach would not be possible if the high-level reasoning layer was non-differentiable, as the system needs to propagate the gradients through the high-level reasoning layer in order to provide feedback to the low-level perception neural network. That is why Neuroplex uses a neural network to approximate the high-level reasoning layer, because of its intrinsic differentiability. This allows Neuroplex to train in an end-to-end manner to recognize complex events.

The Neuroplex architecture keeps a dual representation for the high-level reasoning layer. It uses the DeepCEP framework (Xing et al., 2019), for the forward inference, and uses NRLogic for the backward training. In DeepCEP, a logical machine in the form of a CEP engine is generated from the rule definitions created by the user. It takes simple events abstracted by perception models and detects complex events based on definitions. During training, this logical machine can then be used as the teacher model for transferring the knowledge to the NRLogic model, which propagates gradients to the low-level perception layer.

Neuroplex also has some limitations. If the rules for the complex events are modified, a new CEP engine can be generated to perform inference. However, if users want to fine-tune the model with the new set of complex event rules, the high-level reasoning neural network NRLogic would need to be trained again. This could cause additional overhead in practice.

Secondly, while Neuroplex can generate synthetic data to train the neural network to emulate the rules, it is not possible for the user to know that the neural network will behave exactly as the rules define in all situations. This is due to the nature of the neural network, which may give an unexpected answer if the given situation has not been seen in the training data. The only way to guarantee that the neural network will always behave as expected is to evaluate every possible situation, which becomes unfeasible as the complexity of the problem increases. As a result, there is a risk that Neuroplex will not be robust against some adversarial conditions. Although this could be avoided by performing inference with the logic machine as the reasoning layer, it is still an issue during training for the perception model, as the use of the neural network is required to perform the backpropagation. Having a sub-optimal logic model to propagate gradients could lead to poor learning performance in the perception model. In the experiment section, we will show examples where Neuroplex fails to get robust results without careful fine-tuning.

In this paper, we propose the architecture DeepProbCEP, where the reasoning layer uses differentiable logic programs that are explainable and robust to adversarial conditions.

2.3. Tools used

This section introduces the various tools used in building our approach. We first introduce ProbLog, a probabilistic logic programming language which is used in DeepProbLog, the tool that allows us to perform end-to-end training in DeepProbCEP as a neuro-symbolic approach.

2.3.1. ProbLog

ProbLog (De Raedt, Kimmig, & Toivonen, 2007) is a probabilistic logic programming language. ProbLog allows users to encode complex interactions between different components. A ProbLog program consists of a set of probabilistic facts F and a set of rules R. Facts have the form $p :: f$ where f is an atom that represents a concept that can be true or false. p is a value between 0 and 1 that represents the likelihood of the fact being true. Rules have the form $h :- b_1, \dots, b_n$ where h is an atom and b_i are literals. A literal is an atom or the negation of an atom.

One convenient syntactic extension is an annotated disjunction (AD), which is an expression of the form $p_1 :: h_1; \dots; p_n :: h_n :- b_1, \dots, b_m$. where the p_i are probabilities so that $\sum p_i = 1$, and h_i and b_j are atoms. ADs allow us to express choices between different categorical variables. This means that, whenever all the conditions b_i hold, exactly one of the atoms h_j will be true, with probability p_j. All other h_i will be false (unless other parts of the program make them true).

2.3.2. DeepProbLog

DeepProbLog (Manhaeve, Dumancic, Kimmig, Demeester, & De Raedt, 2018; Manhaeve, Dumanić, Kimmig, Demeester, & De Raedt, 2021) is a neural probabilistic logic programming language that allows the user to create neuro-symbolic architectures. DeepProbLog allows the user to train the neural networks in these architectures as part of the system in an end-to-end manner.

A DeepProbLog program is a ProbLog program that is extended with a set of neural ADs (nADs). These nADs work similarly to ADs in the sense that they provide a mutually-exclusive distribution of probabilities over a set of clauses. In nADs, however, these probabilities are generated from the output of a neural network, instead of being manually defined. For instance, in order to identify the digits in MNIST images to allow us to perform operations with them, we would specify the following nAD:

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

This would then allow us to perform a query of the form `digit(``,Y)`. This query would be transformed into an AD of the form

$$p_0 :: \texttt{digit}(\blacksquare, 0); p_1 :: \texttt{digit}(\blacksquare, 1); \dots; p_9 :: \texttt{digit}(\blacksquare, 9)$$

where the probabilities p_i are based on the output of the neural network. This neural network can take any shape, as long as its output satisfies the AD requisite of $\sum p_i = 1$. A softmax activation is usually used in the last layer to guarantee this.

After defining the structure of the neural network and the logic level, it is possible to use DeepProbLog to infer the answers to our queries. In order to perform this inference, DeepProbLog transforms the logic layer into an arithmetic circuit, obtaining the required probabilities from the neural network. Using this arithmetic circuit, the probability for the given query can be calculated. Each arithmetic circuit is specific to a given query, meaning that this transformation needs to be performed each time.

In order to train the neural network, the system first performs inference as described above. Then, DeepProbLog performs gradient-based learning of the neural network through the logic layer. This is possible because the arithmetic circuit is differentiable, which allows for the use of backpropagation. For a more detailed explanation of the technical aspects of DeepProblog's inference and learning, see Manhaeve et al. (2021).

3. DeepProbCEP: a neuro-symbolic approach for CEP

In this paper, we propose DeepProbCEP, a neuro-symbolic approach to CEP designed to fulfil the objectives defined in Section 1. In particular, DeepProbCEP is designed to be able to train even with small amounts of data. This is achieved by allowing us to inject human knowledge into the system, which facilitates the training process. DeepProbCEP divides the task into two distinct levels; (i) a perception level, where subsymbolic data is classified into the correct type of simple event using a neural architecture and (ii) a reasoning level, where complex events can be detected based on the rules that define them.

In DeepProbCEP, the perception level is implemented by a neural network, which classifies the subsymbolic data into a pre-defined set of classes. Thanks to this classification, the system is able to extract the symbolic information from the input data, which allows the symbolic high-level part of the system to interact with it. This high-level reasoning consists of a probabilistic logic programming layer, where the user-defined rules are contained. This reasoning layer is responsible for detecting the situations under which the complex events occur, as defined by the user. The following sections explain how both levels have been implemented.

As explained above in Section 2.2.2, Neuroplex (Xing et al., 2020) also divides the problem into perception and reasoning levels. However, in Neuroplex, a neural network is used to emulate the rules for the high-level reasoning. In contrast, DeepProbCEP uses an explicit probabilistic logic layer for the reasoning level, which gives DeepProbCEP several advantages. It allows the user to modify the rules for the complex events in a much easier way, as they only need to update them in the logic layer. Secondly, it also eliminates the risk that the reasoning level will behave unexpectedly. While Neuroplex can generate synthetic data from the manually defined rules to train the perception level neural network to emulate those rules, there is no guarantee that this neural network will behave exactly as expected in all situations. This is particularly the case in complex situations, for which it might be difficult to evaluate that the neural network does in fact work as expected. This problem is not present in DeepProbCEP, as the logic rules are used directly as defined by the user.

3.1. Reasoning level

DeepProbCEP builds on top of DeepProbLog (Manhaeve et al., 2021) (see Section 2.3.2 above) that allows for end-to-end training. It allows the user to refine rules for complex events.

To compare the performance of DeepProbCEP against the state-of-the-art, we define complex events as a pattern of simple events that happens within a given time window. This is because those other approaches are capable only of detecting complex events defined in this way. For this paper, only these type of complex event definitions are considered, leaving further interpretations to be considered by future work. However, ProbLog, the logic programming language used in DeepProbCEP to define the rules, has been previously shown to be able to detect complex events using more sophisticated rules in Roig Vilamala et al. (2019) and Skarlatidis et al. (2015). Both of these approaches take inspiration from event calculus (Kowalski & Sergot, 1986) in order to define the conditions for the complex event to start and terminate, which allows them to perform all the operations used by other CEP approaches. As the logic language used is the same, DeepProbCEP also allows users to define complex events in this manner. While we will not go into details in this paper, the demonstration described in Section 1.1 above makes use of this approach. Furthermore, as ProbLog is Turing complete, users should be able to define complex events in the manner they find most appropriate.

In order to facilitate the definition of patterns for the complex events, we have implemented a framework that is able to detect sequences of simple events. The code for this framework has been added

```
1  happensAt(overheating, T) :-
2      sequence([highTemp, highTemp], 5, T).
```

Listing 1: Example of a complex event definition, specifying the case of an overheating machine.

as Appendix A, and its functionalities are explained in the following section.

3.1.1. Sequence framework

The sequence framework provides the clause *sequence(L, W, T)*, which allows the user to define a list *L* of the simple events. The clause will be true if the list *L* happens, in the provided order, within a given window *W* of time, also provided by the user. Finally, the value of *T* is the timestamp at which the sequence is evaluated. This is usually left as a variable in rule definitions, which allows the user to specify which timestamp they want to evaluate at run time.

The clause *sequence* can be used to define complex events where a set of simple events must happen in order. For instance, in order to illustrate how the framework can be used, we have defined some rules to detect if a machine is overheating. For this example, the complex event the user is trying to detect will be *overheating*. Let us assume that there is an input stream that periodically provides us with events indicating the temperature of the machine. These input events cover different temperature ranges (*lowTemp*, *mediumTemp*, *highTemp*, etc.). At the reasoning level, we are not concerned on whether these classes are the result of the classification performed by the low-level perception, or if the data from the input stream was already symbolic. This is because the logic will be the same whether the values come directly from the input stream or if they have been classified by the low-level neural network. Listing 1 shows a way in which the rules for *overheating* could be defined. First, we define the complex event *overheating* (line 1) to be formed by two instances of the simple event *highTemp* (line 2). This can be thought of as a regular expression, where the list provided in line 2 is equivalent to `h.*h`, where `h` stands for *highTemp*. Note that a `.*` has been added between the instances of *highTemp*. By default, *sequence* allows other simple events to happen between the simple events in the provided list. *sequence* also adds a condition where the matched input must be shorter than the given window *W* (set to 5 in this example). As such, if two instances of high temperature arrive from the stream within a window of 5 timestamps, a complex event will be generated, with the other simple events arriving in between being irrelevant.

Using this rule would allow us to perform queries of the form *happensAt(overheating, T)*, where *T* can be substituted for any timestamp. For instance, performing the query *happensAt(overheating, 10)* would return a value between 0 and 1. This value would represent the likelihood of the complex event *overheating* happening at timestamp 10. This likelihood would be calculated based on the rules. If the system knew for a fact that an instance of *highTemp* has happened between timestamps 6 and 9, and another timestamp has happened at timestamp 10, the pattern would match and a complex event would be generated with 100% confidence. If the system is slightly less confident on the fact that the simple events have happened, the confidence on the complex event will reflect this. This can happen for a multitude of reasons, but for our purposes it will generally be because the simple events come from the classification of a neural network, which generally outputs a probability distribution across all the classes, without being 100% certain about any of them.

While, as explained above, *sequence* allows for other simple events to happen between the ones defined in the list *L*, the framework also offers a functionality to specify exceptions. For instance, imagine that the user wanted to specify that the temperature should not be fluctuating from high, to low and back to high again in short periods of time

```
1  happensAt(malfunctioning, T) :-
2      sequence([highTemp, lowTemp, highTemp], 5, T).
3  happensAt(malfunctioning, T) :-
4      sequence([lowTemp, highTemp, lowTemp], 5, T).
5
6  happensAt(overheating, T) :-
7      sequence([highTemp, not(lowTemp), highTemp], 5, T).
```

Listing 2: Further examples of complex event definitions, specifying the case of a malfunctioning sensor and re-defining the overheating case.

(or vice versa). As such, they want to specify that such fluctuations are likely to be caused by a malfunctioning sensor, and as such should not trigger an overheating complex event. Listing 2 defines such a situation. Lines 1 to 4 define the malfunctioning complex event. Note that lines 1 and 2 define one possible sequence of events, whereas lines 3 and 4 define the other. If either of them happen, a malfunctioning complex event will be generated. Lines 6 and 7 specify the new definition for the overheating complex event, where instances of *lowTemp* are not allowed to happen between the instances of *highTemp*. Transformed to a regular expression, this would be the equivalent of `h[^l]*h`, with `h` standing in for *highTemp* and `l` for *lowTemp*. Note that instances of other simple events—such as *mediumTemp*— would still be allowed unless explicitly excluded. This could be done by specifying multiple negated events between the instances of given events. For instance, the list *[highTemp, not(lowTemp), not(mediumTemp), highTemp]* would be the equivalent of the regular expression `h[^lm]*h`, with `m` standing for *mediumTemp*. It is worth noting that the condition that all the simple events forming the pattern must happen within the given window *W* is still present in these cases.

3.2. Perception level

The other part of the system is the perception level, which is responsible for transforming the subsymbolic data into information that can be used to define rules in the reasoning level. This transformation is performed by a neural network, which will classify the input subsymbolic data into a pre-defined set of classes. The structure of this neural network can be defined using PyTorch (Paszke et al., 2017). This allows the user to make use of the most appropriate neural network architecture for classifying the input data being used. As we demonstrate in our experiments, DeepProbCEP can be used to detect complex events from streams of data of two types (images and audio) just by changing the structure of the neural network and adapting the rules to the new types of simple events. Other neural network structures could be used to make use of many other types of subsymbolic data, which would allow DeepProbCEP to make use of any type of data that can be classified by a neural network.

DeepProbCEP is also able to make use of types of data that are already symbolic, meaning types of data that can be directly processed using rules. This can be done by simply providing the information directly to the reasoning level, without transforming it in the perception level.

Once the reasoning and perception levels are defined, DeepProbLog (Manhaeve et al., 2018, 2021) is used to integrate them. This allows DeepProbCEP to make the logic layer differentiable, and thus makes it possible to train the perception level neural network in an end-to-end manner.

3.3. Setup used in experimentation

In order to compare DeepProbCEP with other approaches in our experiments, we consider a stream of MNIST digits as input. For the aggregation rules we define that a complex event consists of two

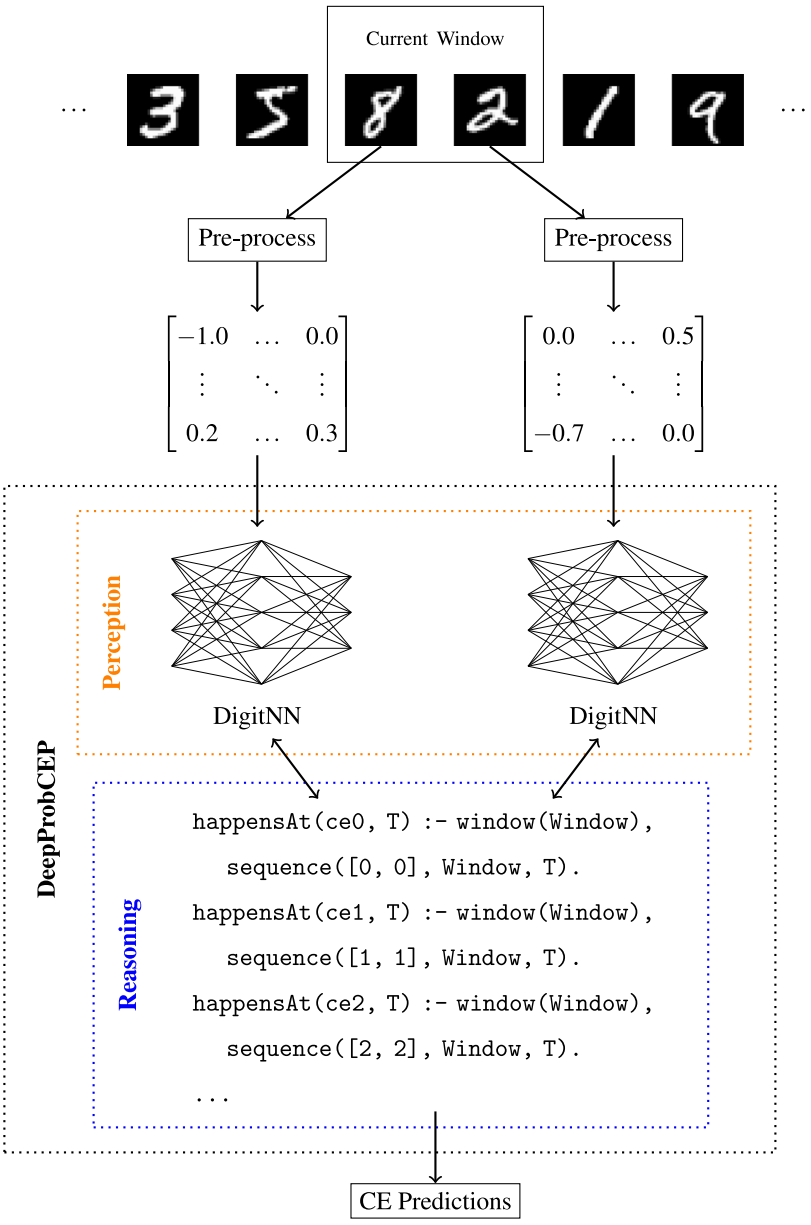


Fig. 1. Overall architecture of DeepProbCEP for the MNIST setting. From top to bottom, the stream of numbers is pre-processed and fed into the perception layer (DigitNN). The reasoning layer uses the outputs from the perception layer to predict when complex events are occurring.

instances of the same digit within the given window. One of these instances must occur at the last position in the window size and the other can occur in any other position. Therefore, if a window of simple events ends with a 0 and there is another 0 somewhere within the window, that will create $ce0$. If the last simple event is a 1 and there is another 1 within the window, that will create $ce1$, and so on. Fig. 1 shows the setup used for DeepProbCEP in all of the MNIST experiments.

As shown in Fig. 1, the first step is to pre-process the input data. For the case of MNIST digits, we simply normalize them with a mean of 0.5 and a standard deviation of 0.5. However, more complex pre-processings can be performed if the type of input data requires it. Then, the matrix resulting of the pre-processing is fed into the DeepProbCEP system. More specifically, they are fed through the perception neural network, DigitNN in the diagram. This neural network identifies the different types of simple events from the input stream. In the case of DigitNN, it detects which digit appears in the image, returning a likelihood value for each of the 10 possible digits (0 to 9). However, as explained above, any neural network structure can be used for this perception level. In this case, DigitNN is the same neural network that was used in the paper presenting DeepProbLog (Manhaeve et al., 2021) for classifying MNIST digits. The architecture of the neural network, which is based on the PyTorch tutorial, consists of 2 sets of convolutional layers with MaxPools, followed by linear layers, as shown in Table 1.

Finally, the prediction from the neural network is then used in the logic layer, which allows it to predict whether or not a certain complex event is happening at a certain point in time. This is based on the rules defined by the user, and any frameworks being used. A snippet of the code used in experimentation can also be seen in the right side of Fig. 1. This shows how complex events $ce0$, $ce1$ and $ce2$ are defined, according to the description for each of the complex events given above. Here, we use the definitions given above where $ce0$ consists of two instances of a 0.

When using DeepProbCEP for experimentation, we set a maximum number of epochs of 100. However, in order to avoid overfitting we also make use of early stopping with a patience of 10 epochs. This means

Table 1
Architecture for DigitNN.

Layer type	Output channels	Kernel size	Stride
Conv2D	6	(5, 5)	(1, 1)
MaxPool2D	6	(2, 2)	(2, 2)
ReLU	6	–	–
Conv2D	16	(5, 5)	(1, 1)
MaxPool2D	16	(2, 2)	(2, 2)
ReLU	16	–	–
Reshape	256	–	–
Linear (ReLU activation)	120	–	–
Linear (ReLU activation)	84	–	–
Linear (Softmax activation)	10	–	–

that if the performance of the system on the validation dataset does not improve for 10 epochs, the training finishes early. The weights that performed the best in the validation dataset are then used for testing.

4. Scenarios and dataset generation

This section explains how we generated synthetic datasets to represent different real world scenarios in a controlled manner, allowing us to evaluate how each of the approaches perform under different circumstances. Table 2 shows a summary of all the types of datasets used for this paper. Each of those dataset types aims to evaluate how modifying a specific parameter affects the performance of the different approaches, allowing us to evaluate how each parameter individually affects the performance of the approach. These datasets are also designed to allow us to control the severity of each of the parameters, ranging from ideal circumstances to extremely difficult, thus allowing us to evaluate up to which degree the approaches are likely to perform well. The following sections provide more detail on what scenarios are being emulated by each dataset type, as well as the procedure used to generate those datasets.

One of the main parameters is the size of the sliding window within which the system looks for complex events. All the compared approaches make use of a sliding window to detect the complex events. In order to generate a complex event, all of the simple events that will be aggregated need to be within this sliding window at the same time. As such, the size of the sliding window limits the maximum span of time for a complex event. The value for the size of the sliding window needs to be determined before the datasets are generated, as the training labels need to match that window size. It might seem best to use a window size as big as possible to detect as many complex events as possible. However, increasing the window size increases the number of simple events that are being considered, which makes the problem of finding a pre-defined pattern harder. This is especially true if we want the system to be fast. Furthermore, sometimes users might want to limit the window size as, if two simple events are too distant from each other temporally, there might not be a relationship between them.

Another parameter is the size of the training dataset: in real scenarios, complex events tend to be rare. Therefore, seeing how much changing the size of a dataset affects the performance of the system will allow us to predict how well that system will perform when training with sparse data. In order to evaluate this, we have created the *data-scarcity datasets*, which allow us to compare how different approaches perform after training with different amounts of training data.

Finally, we want to see how robust DeepProbCEP is against incorrectly labelled training data, as, in some real world scenarios, the situations represented by different complex events may quite similar or ambiguous, making them difficult to differentiate, even for humans. For this purpose, we have created different types of poisoning adversarial attacks by generating new versions of the dataset where a part of the training points has been affected by one of the poisoning attacks we are considering. The *random noise datasets* are designed to evaluate the effect of random noise in the labelling. On the other hand, the *targeted poisoning datasets* evaluate the effect of poisoning the training data with the intent of making the system misclassify either one (*single label*) or all (*multiple labels*) classes as a substitute one.

For all datasets, the same definitions for the complex events are being used, as defined above ($ce0$ to $ce9$).

4.1. Base dataset

This section describes how the *base datasets* were generated, which represent scenarios with different window sizes. While these datasets have not directly been used for training, they are used as the base for the rest of the datasets in this paper, which were generated through the modifications described in following sections. The process used to generate the base dataset allows us to change the window size by changing the value of $Window$. $Window$ is a positive integer that indicates the number of timestamps between the first and last simple events that form a complex event. For this paper, we have generated datasets with window sizes of 2, 3, 4, 5, 10 and 15.

In order to generate the base dataset, MNIST images are used, which we have split into three datasets; training, validation and testing. Each dataset has 50,000, 10,000 and 10,000 images, respectively. We then use these MNIST images datasets to generate the base datasets for training, validation and testing. However, these datasets are not directly used for training. Instead, they are used as a base to generate the other datasets, as explained in the following sections. The steps to generate the base datasets are shown in Fig. 2, which illustrates the following steps:

1. We take all the images from the original dataset and randomly shuffle them into a sequence S of simple events, where each image represents one simple event. Therefore, the length of S is the number of images in the original dataset. Simple events can be accessed by their index like so $S[I]$, and for each of them we can also access their image (which represents a digit from 0 to 9) and their class (the ground truth of which digit is represented) using $S[I].image$ and $S[I].digit$, respectively.
2. We create a list C that will indicate for each timestamp whether a complex event happens. We initialize this list with *null*, which hereinafter represents that no complex event happens at the specified timestamp.
3. For each timestamp T where $0 < T < len(S)$:
   (a) If the pattern for one of the complex events occurs, mark $C[T]$ as the corresponding complex event. More specifically, if there exists P such that $T - Window < P \leq T$ and $S[P].digit = S[T].digit$, mark $C[T]$ as ceN, where N is the value of $S[T].digit$. This means that if the last digit in the window is a 0 and there is another 0 within the window size, we mark the timestamp as $ce0$. If the last digit is a 1 and there is another 1, we mark it as $ce1$, and so on.
   (b) Otherwise, leave $C[T]$ marked as the null class.
4. Finally, if this is the training dataset, generate the training sequence of simple events TS, which will only contain the images, but not the ground truth of which class they represent, as these should not be available when performing end-to-end training. Therefore, $TS[I] = S[I].image$ for $0 < I < len(S)$.

4.2. Data-scarcity datasets

In order to evaluate how the different approaches may perform in scenarios where training data is sparse, compared to scenarios with large amounts of training data, we have created the *data-scarcity datasets*. These datasets range in size from 100 to 400,000 training

Table 2
Summary of the dataset types used in this paper, with a description of the scenarios they represent.

Dataset type		Scenario	Training dataset Size
Base datasets		Used to compare changing window sizes. Not directly used for training, but used to generate other datasets.	49,999
Data-scarcity datasets		Used to compare training with different amounts of training data, ranging from sparse to dense data.	10 different sizes 100, 500, 1000, 2500, 5000, 10,000, 25,000, 50,000, 100,000, 400,000
Random noise datasets		Used to evaluate how robust approaches are to noisy training datasets.	2500
Targeted poisoning datasets	Single label	Used to evaluate robustness against poisoning attacks targeting *one class*	2500
	Multiple labels	Used to evaluate robustness against poisoning attacks targeting *all classes*	2500

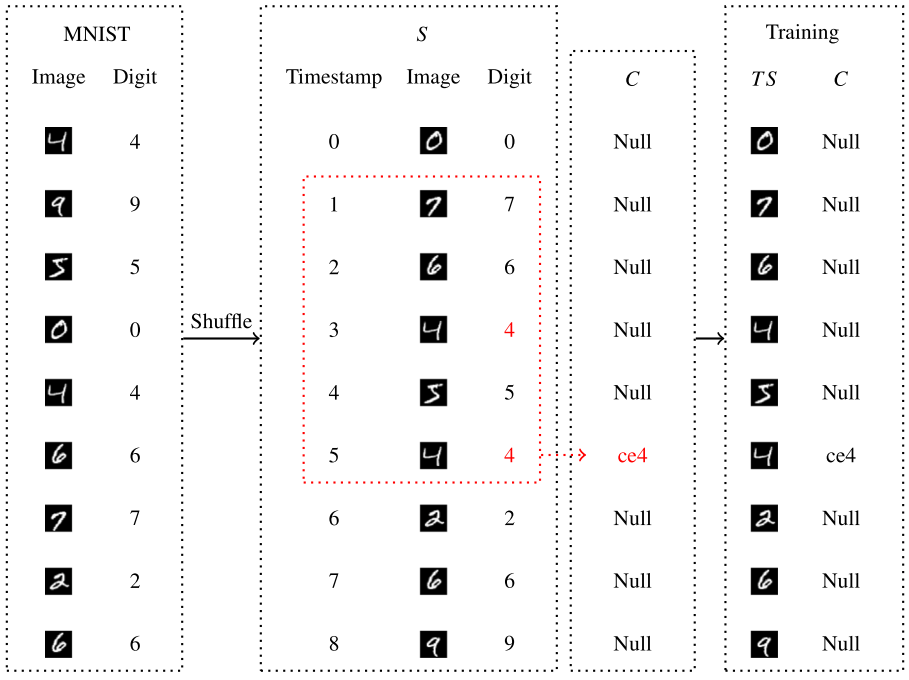


Fig. 2. Diagram representing how the datasets used in this paper are generated. A window of 5 has been used and a complex event *ce4* has been detected at timestamp 5.

points.[2] For this setting, we consider the datasets with 2500 training points and less to be sparse, whereas the datasets with more than 50,000 training points are quite dense. These data-scarcity datasets have been generated by performing either undersampling or oversampling on the base datasets.

The smaller datasets are generated by performing a balanced undersampling for the different classes. In this process, we randomly select only a few of the training examples from the base dataset in order to get to the total amount of training examples required. In contrast, for the bigger datasets we perform oversampling on the complex event classes. The only class that is not oversampled is the null class, which happens when no complex event occurs. This is because there are many more cases of the null class in the base dataset, as it is much more likely for no complex event to happen than any other individual complex event, given our complex event definition. It is important to note, however, that all datasets used for training have been balanced. This was done in order to avoid overfitting for a specific class. The different dataset sizes we have used, as well as their class distribution, can be seen in Table 3.

[2] A total of 10 different training dataset sizes have been used: 100, 500, 1000, 2500, 5000, 10,000, 25,000, 50,000, 100,000, 400,000.

4.3. Random noise datasets

Given the complexity of the definition of some of the complex events in realistic scenarios, it can sometimes be hard to correctly label when a certain complex event is happening. This can lead to errors on the training dataset, which might affect the accuracy of the system after training. In Section 1, we defined that one of our objectives was to be robust against adversarial conditions. As such, we want to evaluate how much of an effect training using a noisy dataset has on the performance of DeepProbCEP. While synthetic datasets should not contain unexpected noise, they do allow us to introduce noise in a controlled manner, making it possible to evaluate how well an approach may perform when trained under different levels of noise. This can provide us with information that can be useful when using the same approaches on real datasets, which might contain unknown levels of noise.

A poisoning attack is used, where a percentage of the training labels are randomly changed for another label, also chosen randomly, to simulate this noise. For instance, assume that, based on the complex event definitions, a certain timestamp is marked as $ce0$. However, if this case is affected by the noise it will be labelled as another complex event, which is selected randomly every time. We will call the datasets generated using this process *random noise datasets*.

Table 3
Distribution of instances on the data-scarcity datasets. The distribution is the same for all window sizes. Note that all classes have almost the same number of instances, with a maximum difference of 1.

Total	Null	ce0	ce1	ce2	ce3	ce4	ce5	ce6	ce7	ce8	ce9
100	10	9	9	9	9	9	9	9	9	9	9
500	46	45	45	46	45	45	45	46	46	46	45
1000	91	90	91	91	91	91	91	91	91	91	91
2500	228	227	227	227	227	227	227	227	228	228	227
5000	455	454	454	455	454	454	455	455	455	455	454
10 000	910	909	909	909	909	909	909	909	909	909	909
25 000	2273	2272	2272	2273	2273	2272	2273	2273	2273	2273	2273
50 000	4546	4545	4545	4546	4545	4545	4545	4546	4546	4546	4545
100 000	9091	9090	9091	9091	9091	9091	9091	9091	9091	9091	9091
400 000	36 364	36 363	36 363	36 364	36 364	36 363	36 364	36 364	36 364	36 364	36 363

In order to generate the random noise datasets, we use the same steps described to generate the base dataset, explained above in Section 4.1. However, when labelling a certain timestamp as a complex event (Step 3a), there is probability of changing that label to a random complex event. This probability is determined by the noise percentage parameter. We have generated datasets ranging from 0.0 to 1.0 noise, with a step of 0.2. For this, 0.0 means that no data has been poisoned, while 1.0 means all the data has been poisoned. Finally, the datasets are balanced and reduced to a size of 2500 training points.

It is important to note that this attack is only performed on the training dataset. This means that the testing dataset will maintain the ground truth, which will allow us to see how the system would perform in a real life scenario.

This kind of noise might appear both due to a malicious agent, and to the difficulty of labelling the sophisticated scenarios where a system like DeepProbCEP would be useful: for instance, different annotators might be having different consideration on what constitutes the complex event.

4.4. Targeted poisoning attacks

We also want to evaluate how a targeted poisoning attack might affect the performance of our approach. More specifically, we are interested in poisoning attacks that target specific labels with the intention of making the system incorrectly classify them as another label. The following sections describe the creation of two datasets, each under a different type of targeted attack. These scenarios simulate what could happen if an annotator is maliciously trying to provide incorrect training data, but they could also happen non-maliciously if an annotator is confused as to which samples belong to which classes. This could particularly be the case if the conditions for two different complex events are difficult to differentiate.

4.4.1. Targeted attack against a single label

In the first targeted attack, we are only targeting one complex event class. More specifically, we are choosing the targeted complex event and its substitute. This type of attack should cause the system to incorrectly classify instances of the targeted complex event as the chosen substitute. Again, in order to generate the training datasets with this attack we follow the steps to create the base dataset described in Section 4.1 above. However, when labelling the complex events (Step 3a), if we would label the complex event as the target, there is a probability of changing that label to the substitute class. As is the case for the random noise attack, we use probabilities ranging from 0.0 to 1.0 with a step of 0.2. In this case, with a probability of 1.0 all the complex events for the targeted class will be labelled as the substitute class. Finally, these datasets are also balanced and reduced to a size of 2500.

4.4.2. Targeted attack against multiple labels

For this second attack, all complex events are targeted. In this case, we choose a substitute for each complex event. This attack should cause the system to incorrectly classify all complex events as another type of complex event, based on the substitution we are applying. The process for generating the datasets under this type of attack is very similar to the previous attack. More specifically, in Step 3a there is a probability of changing the label for the complex event for its substitute. Again, probabilities between 0.0 and 1.0 are used. In this case, a probability of 1.0 would mean that all the training points have the substitute class label, effectively meaning that we are training for the incorrect class. These datasets are also balanced and reduced to a size of 2500.

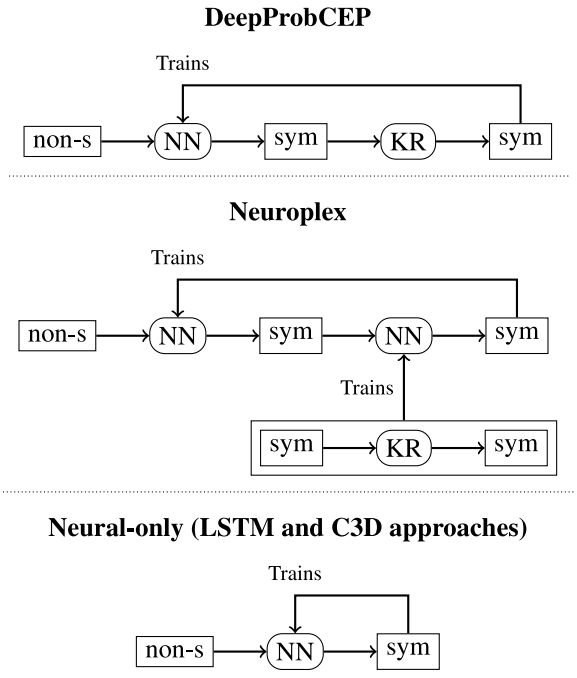


Fig. 3. Flowchart of the approaches considered for this paper. Rectangular boxes represent data, using *sym* for symbolic data and *non-s* for non-symbolic data. Oval boxes represent algorithms, using *NN* for neural network components and *KR* for knowledge reasoning (symbolic) components.

5. Experiment design

In order to evaluate DeepProbCEP and compare it with existing approaches, a number of experiments have been performed. These experiments aim to evaluate how well DeepProbCEP fulfils the four objectives defined in Section 1. Fig. 3 shows a high level flowcharts of the different approaches. More details for each approach are provided in the following sections.

In *Experiment 1: Performance with sparse data*, we aim to find how much each approach is affected by using small datasets for training. This allows us to evaluate how well the objective of limiting the cost

of obtaining training data is fulfilled. The data-scarcity datasets are used for this experiment. After training each approach with each of the dataset sizes, we compare their performance in terms of classification accuracy. Naturally, all approaches are expected to perform worse as the size of the training dataset is reduced. However, we are interested in which approach is the least affected by this reduction of the training dataset size. The approach that is the least affected will be the most robust when training with sparse data. As complex events tend to be rare, this would be desirable, because training datasets will tend to be small in real world scenarios, as explained in Section 1.

Experiment 2: Performance on simple events focuses on the accuracy when classifying simple events after training in an end-to-end manner. This is thanks to another advantage of the neuro-symbolic approaches, which allows them to learn how to classify simple events despite the fact that they have only been trained with information about the complex events. This is made possible by the introduction of human knowledge into the system, which allows it to learn to classify simple events in a manner understandable by humans. Therefore, a byproduct of training a neuro-symbolic system is a simple event classifier. This could be quite useful in a real world scenario, as we are working under the assumption that the labels to directly train a simple event classifier are not available. Our second experiment evaluates the performance of DeepProbCEP and Neuroplex when classifying these simple events. For that purpose, as in Experiment 1, this experiment also uses the data-scarcity data training datasets. This also allows us to see how training with smaller datasets affects each of the approaches when classifying simple events.

Experiment 3: Time efficiency compares the time efficiency of DeepProbCEP and Neuroplex when training. As explained above in Section 2.2.2, using a neural network to emulate the behaviour of a logic layer does allow Neuroplex to perform training and inference faster than using the logic layer directly. While the direct use of a logic layer in DeepProbCEP does provide other advantages, such as making it easier to use different rules, it also causes DeepProbCEP to take a longer time for each training epoch. Experiment 3 explores how significant the difference in time efficiency between Neuroplex and DeepProbCEP is.

After that, in *Experiment 4: Robustness against poisoning attack with random noise*, the robustness of DeepProbCEP against training with random noise is evaluated, followed by evaluating the robustness against targeted poisoning datasets (*Experiment 5: Robustness against targeted poisoning attacks*). While further experiments should be performed with other types of attacks, this allows us to demonstrate that DeepProbCEP is robust against attacks that focus on the training data.

Experiment 6: Performance when modifying the rules demonstrates that DeepProbCEP allows the user to modify the rules for the complex events without requiring re-training. For that purpose, we first train DeepProbCEP using one of the data-scarcity datasets. Then, the complex event rules are changed and the updated system is evaluated against a test dataset generated under those new complex event rules. DeepProbCEP should be expected to still achieve a very high performance when using these new rules. This demonstrates that DeepProbCEP is both flexible and modular on the rule definitions, as per objective 2.

Finally, *Experiment 7: Performance in an audio setting* evaluates how well DeepProbCEP performs when using a different type of input data. More specifically, a stream of audio is used as an input. This allows us to further evaluate how well the first objective of being able to operate on different types of subsymbolic data is fulfilled.

It is important to note that, in all cases, the approaches are tested with a dataset with the same window size as the dataset they were trained in.

The following sections describe the architectures for the existing approaches against which we are comparing DeepProbCEP. These comprise Neuroplex and two neural-only approaches, one using a Long Short Term Memory (LSTM) architecture and one using a Convolutional 3D (C3D) architecture. First, we describe the specific neural network architecture used by each of the neural-only approaches for experimentation. Then, we describe the adaptations made to Neuroplex to work with the datasets used for experimentation in this paper.

Table 4
Architecture for the LSTM approach. W is an input variable that represents the size of the window.

Layer type	Input size	Output size
Reshape	W, 28, 28	W, 784
LSTM	W, 784	W, 100
Linear	100	11
Softmax	11	11

Table 5
Architecture for the C3D approach. $T1$ and $T2$ are variables that change depending on the $Window$ that is being used. More specifically, $T1 = ceil(Window/2)$ and $T2 = ceil((Window + 1)/2)$, where $ceil$ rounds up to the nearest integer.

Layer type	Output channels	Kernel size	Stride
Conv3D	16	($T1$, 5, 5)	(1, 1, 1)
MaxPool3D	16	(1, 2, 2)	(1, 2, 2)
Conv3D	32	($T2$, 5, 5)	(1, 1, 1)
MaxPool3D	32	(1, 2, 2)	(1, 2, 2)
Linear	256	–	–
Linear	128	–	–
Linear	64	–	–
Linear	32	–	–
Linear	11	–	–

5.1. LSTM approach

For the LSTM approach, we combine an LSTM cell with a Multi Layer Perceptron (MLP). As shown in Fig. 4, each of the simple events in the given window is fed into the LSTM cell. The order in which the events are fed into the LSTM cell is the same as the order in which they come in. The hidden state is passed through the LSTM cells and, finally, the prediction is based on the output of the last LSTM in the window (see Table 4). When using the LSTM approach, we have set a maximum number of epochs of 500. However, early stopping is also used with a patience of 15. This means that, if performance on the validation dataset does not improve for 15 epochs, the training is stopped early. Then, the weights that gave the best performance on the validation dataset are used to test the model. This is done to avoid overfitting the model. In our experiments, this approach stopped before reaching 500 epochs in all cases.

5.2. C3D approach

For the C3D approach, we used the architecture shown in Table 5. As it can be seen in the table, the architecture consists of two pairs of a Conv3D layer plus a MaxPool3D layer. This is followed by a set of linear layers. Table 5 also shows that the first dimension for the Convolutional 3D layers depend on the window size. This is because that is the time dimension, meaning that this part of the architecture needs to be consistent with the number of simple events that are being considered within the window.

For this approach we also use a maximum number of epochs of 500 and an early stopping with a patience of 15. In our experiments, this approach stopped before reaching 500 epochs in all cases as well.

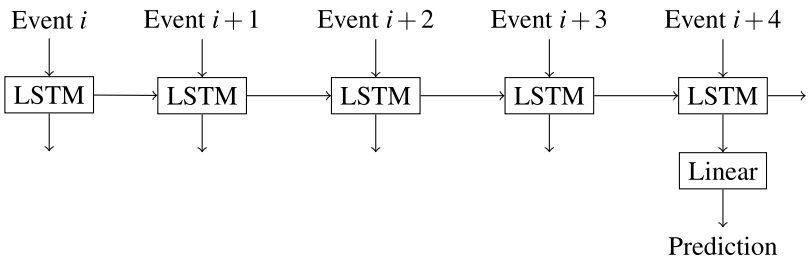


Fig. 4. Architecture for the LSTM approach for a window of 5. A smaller window would have less input events and a bigger window would have more. As it can be seen in the figure, each hidden state is passed from each LSTM to the next. Finally, the output for the last LSTM is passed through a linear layer which outputs the prediction for the system.

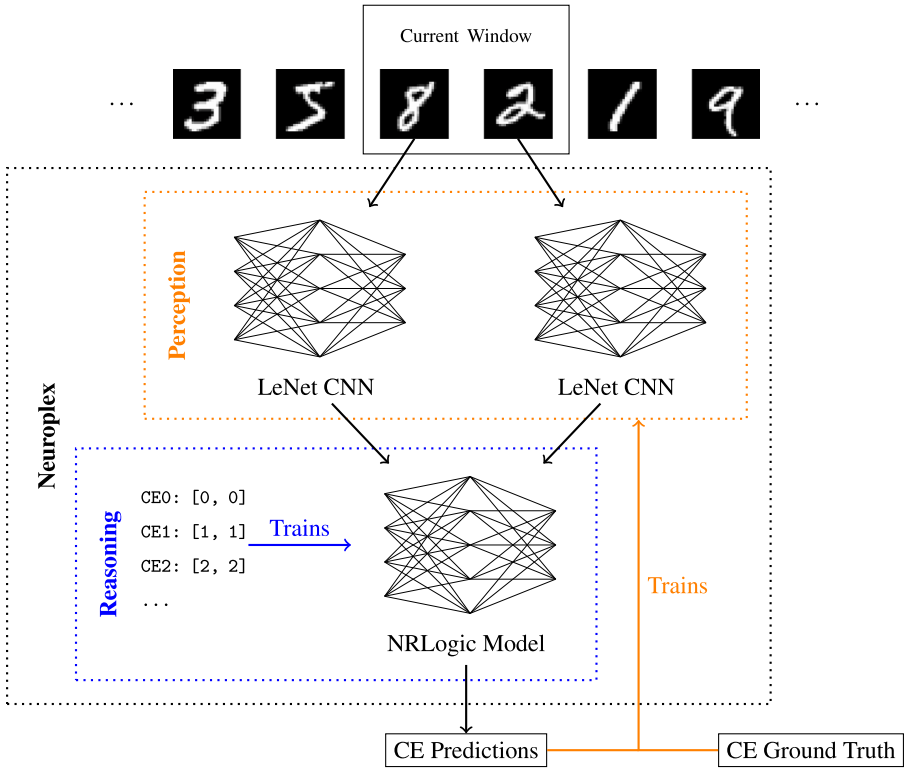


Fig. 5. Overall architecture for Neuroplex in our experiments. First, the knowledge from the user-provided rules is distilled into the NRLogic Model. Once this step is complete, the input stream of MNIST digits is fed into the system, going through the perception and reasoning layers. This outputs a prediction for the state of the complex events, which can be used to train the perception layer based on the ground truth.

5.3. Neuroplex approach

For our experiments, we use the most recent implementation of Neuroplex,[3] at the time of writing. While the Neuroplex architecture is designed to allow for the use of the full expressivity of DeepCEP (Xing et al., 2019), the current implementation only supports the detection of sequences, which is the reason our experiments focus on complex events that use this type of definition. Fig. 5 shows the architecture used for Neuroplex.

We have, however, slightly modified this implementation so that it represents the same problem as the one considered by DeepProbCEP. This is because, while both problems are quite similar, there are some small differences. In both problems, we are trying to detect certain predefined patterns that form complex events within a certain window. Also in both cases, only one type of complex event can happen for a given window. While both approaches would be capable of dealing with multiple complex events happening at the same time, this has not been considered in the datasets used to train or test them for simplicity of evaluation. However, there is a key difference between the problems. In Neuroplex, the output of the system is the number of times the given pattern happens in the given window. This means that, for example, if we are looking for complex event $ce0$ in the window $[0, 4, 0, 1, 0]$, the system would be expected to output that $ce0$ happens 2 times. This is because $ce0$ is defined as a 0 at the last position in the window and at some other point in the window. Since there is a 0 at the last position and two 0 s earlier in the window, Neuroplex should say that $ce0$ is happening 2 times. In reality, the current implementation of Neuroplex outputs a float for each type of complex event. This float is then rounded to the nearest integer to generate the prediction that is shown to the user.

In contrast, for the problem considered in our experiments, we are not concerned about how many times the pattern for the complex event happens. Instead, we only consider two options: (i) the complex event happens (without mattering if it is one or more times) or (ii) it does not happen. As such, for ease of testing, we treat the system as outputting which complex event is happening. This allows us to evaluate the

[3] This implementation can be found on https://github.com/nesl/Neuroplex/tree/938221da14cdc71463775be1f78c1bf95e1bc106 (on 15th of August 2022).

performance of the system as a multi-class classification problem. As such, for the dataset used in this paper, 11 classes are used. 10 of these classes correspond to each of the complex event types. Meanwhile the last class corresponds to the case where none of the complex events are occurring, which we refer to as the null class.

In order to compare the accuracies between Neuroplex and the other approaches, the Neuroplex output is converted into the format used by other approaches. This is done by finding the complex event for which Neuroplex predicts the most occurrences (before rounding), and using that complex event as the predicted class. In the unlikely case that two complex event classes output the exact same float value, the first class in alphabetic order would be selected. There is also a special case if Neuroplex predicts zero occurrences for all complex events after rounding (that is, if the float outputted for each complex event class is less than 0.5). In that case, Neuroplex is predicting that none of the complex events is happening in the given window, so the null class is used as the prediction. This allows us to treat the output of Neuroplex as a multi-class classification problem as well, and thus allows us to calculate the accuracy in the same way as the other approaches.

Another change is also needed in order to train Neuroplex. This is because, in the same way as the output, Neuroplex uses the number of times each complex event is happening in order to train. For all other approaches, we provide the system with the class of complex event that occurs in the given window (or the null class if no complex event occurs). However, Neuroplex is also provided with the number of times it happens in the given window. This may give a slight bias towards Neuroplex, as it is provided with more information than the other approaches. However, we preferred to do this instead of changing the architecture for Neuroplex to work directly with our dataset, as some experimentation with trying to adapt the Neuroplex architecture to work as a multi-class classifier significantly worsened its performance.

For the Neuroplex approach, a maximum number of epochs of 200 is used, with an early stopping with a patience of 20 epochs. For all other parameters, the default values from the original Neuroplex paper (Xing et al., 2020) are used. Some experimentation was done on optimizing these parameters for this problem. However, other configurations either showed no improvement or provided a worse performance.

6. Results

This section explores the results for the experiments defined in Section 5. The following sections compare the performance of DeepProbCEP with the state-of-the-art approaches introduced in Section 2. They also evaluate how well DeepProbCEP fulfils the objectives defined in Section 1.

All the values displayed on the graphs and tables in the following sections are the result of averaging the accuracies of 3 different executions. The standard deviations of these accuracies are also shown in the graphs as error bars.

6.1. Performance with sparse data

Fig. 6 shows the classification accuracy for each approach in the different window sizes and after training on the data-scarcity datasets. As shown in the figure, DeepProbCEP consistently outperforms the neural-only approaches, obtaining performances of 99.28%, 98.62%, 97.95%, 97.39%, 96.49% and 96.23% for windows 2, 3, 4, 5, 10 and 15 respectively when using 5000 training points. The graphs only show results for DeepProbCEP with datasets of up to 5000 training points. This is because DeepProbCEP takes a significantly longer time to train than the other approaches. Despite this, training DeepProbCEP with just 100 data-points can result in higher complex event classification accuracy than either of the neural-only approaches achieve when trained over 400,000 data-points. This is despite the fact that we are using the best performing architecture we have been able to find through manual optimization for both the LSTM and C3D approaches, which is reflected by the performances of both approaches when trained with enough data. However, by its very nature, any neural-only approach used to solve this problem will need to learn to differentiate each of the simple event types at the same time that it is learning the rules that define the complex events. This is certainly possible given enough training data, as shown in the graphs. However, in problems where data is sparse the performance will be severely impacted.

In order to solve this issue, neuro-symbolic approaches make use of human knowledge, which significantly reduces the amount of training data required. For this problem, the user must specify the rules that define a complex event, which allows the system to focus on learning how to classify the simple events. This is reflected on the performances obtained by both neuro-symbolic approaches, Neuroplex and DeepProbCEP. As we can see in Fig. 6, both neuro-symbolic approaches classify complex events with an accuracy significantly higher than the neural-only approaches. However, we can also see that DeepProbCEP significantly outperforms Neuroplex, particularly when considering small window sizes, see Figs. 6(a), 6(b), 6(c), 6(d) (Window sizes of 2 to 5). Furthermore, even in the bigger window sizes where Neuroplex performs its best, DeepProbCEP gets slightly better and more consistent results, as shown in Figs. 6(e) and 6(f) (Window sizes of 10 and 15, respectively). It is important to remark that, before the experiments in this paper, Neuroplex had almost exclusively been tested with window sizes of 10 and bigger. The only exceptions are two cases in Xing et al. (2020) that are intentionally designed to have significantly less complexity in order to allow the baseline models to learn. Therefore, some fine tuning of the architecture might be necessary to get better performances in smaller window sizes. In contrast, the fact that DeepProbCEP directly uses logic programming for the logic layer means that it will always perform as expected, without requiring any fine tuning. This makes DeepProbCEP more robust, as defined in our fourth objective from Section 1.

6.1.1. Types of error

This section explores what types of error each of the approaches tend to make by looking at their confusion matrices, shown in Figs. 7, 8, 9, 10, 11, 12. As we can see, for the bigger window sizes the confusion matrices are not significantly different between DeepProbCEP and Neuroplex. However for smaller window sizes Neuroplex seems to incorrectly classify some of the cases where a complex event occurs as the null class (represented in the last column and row in the confusion matrices). This matches our observations that Neuroplex does not perform as well with smaller window sizes.

The most common mistake in smaller window sizes (between 2 and 5) for both neural-only approaches (LSTM and C3D) is also classifying cases with a complex event as the null class. However, there is a clear difference between the approaches as Neuroplex seems to consistently fail at classifying a few of the classes, but getting the rest of them correctly almost all of the time. In contrast, the neural-only approaches tend to incorrectly classify a small percentage of almost every other class as the null class.

The contrast between the neuro-symbolic and the neural-only approaches is even clearer in the bigger window sizes (10 and 15), as both neuro-symbolic approaches classify almost all of the cases correctly, as shown in Figs. 11(a), 11(b), 12(a), 12(b). Meanwhile, Figs. 11(c), 11(d), 12(c), 12(d) show how the neural-only approaches, and particularly the LSTM approach, seem to perform very fairly well at classifying the cases where a complex event is happening. However, both approaches perform quite badly when classifying a case that belongs to the null class, where we are trying to identify that none of the patterns that define a complex event is occurring. The errors shown by Neuroplex, the LSTM and the C3D are all related to the null class, which indicates that correctly identifying whether a case is part of this class or not is the hardest part of the problem.

The fact that it is hard to correctly classify the null class is also quite relevant in terms of the performance of the whole system as, both

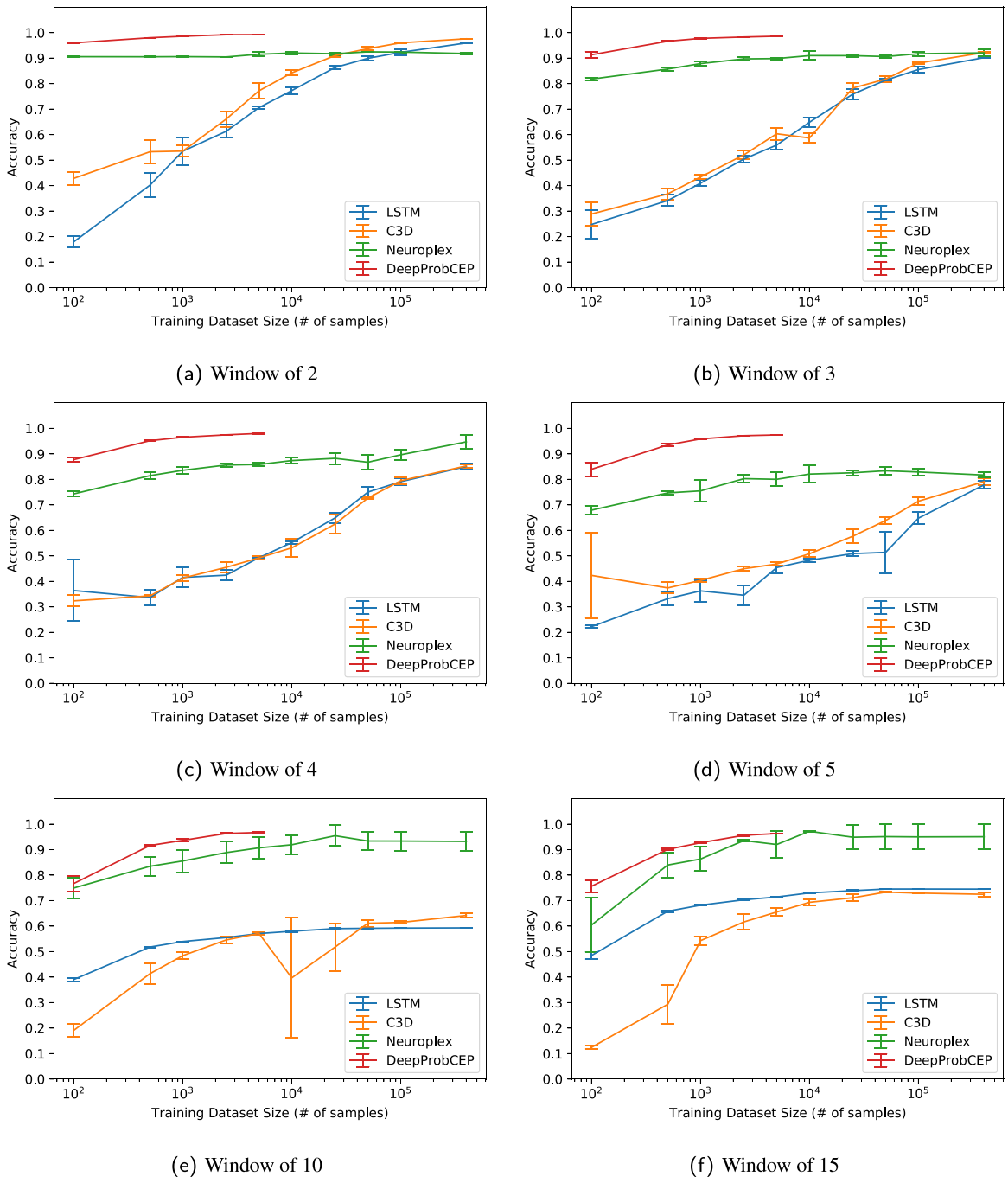


Fig. 6. Accuracy when detecting complex events for each of the considered approaches in different windows. Horizontal axis indicates the size of the training dataset used (in number of samples in the dataset).

in real scenarios and in our testing dataset, the null class is the most prominent, since complex events tend to be rare. This is especially true for smaller window sizes, as considering less complex events makes it less likely that a specific pattern will occur. This means that performing well at classifying the null class is more important for the overall accuracy than correctly classifying every single type of complex event. The higher percentage of null classes in the validation dataset for smaller window sizes, combined with the difficulty of classifying the null class, might be rewarding the systems for defaulting to the null class when they are unsure of which class they should decide on, which would explain why most of the cases that are incorrectly classified select the null class. However, it is important to note that this is only an effect of selecting the best performing weights on the validation dataset, as the training dataset is balanced, with the null class appearing the same number of times as each of the other classes.

6.2. Performance on simple events

Thanks to the fact that the neuro-symbolic approaches use two distinct levels to solve this problem, we are able to obtain a simple event classifier as a byproduct of training them. This classifier can be obtained by extracting the low level perception neural network from either of the neuro-symbolic approaches after training.

This is not possible in neural-only approaches, as they do not have an architecture separated into two levels. Furthermore, even if a similar architecture using a perception level was used, the lack

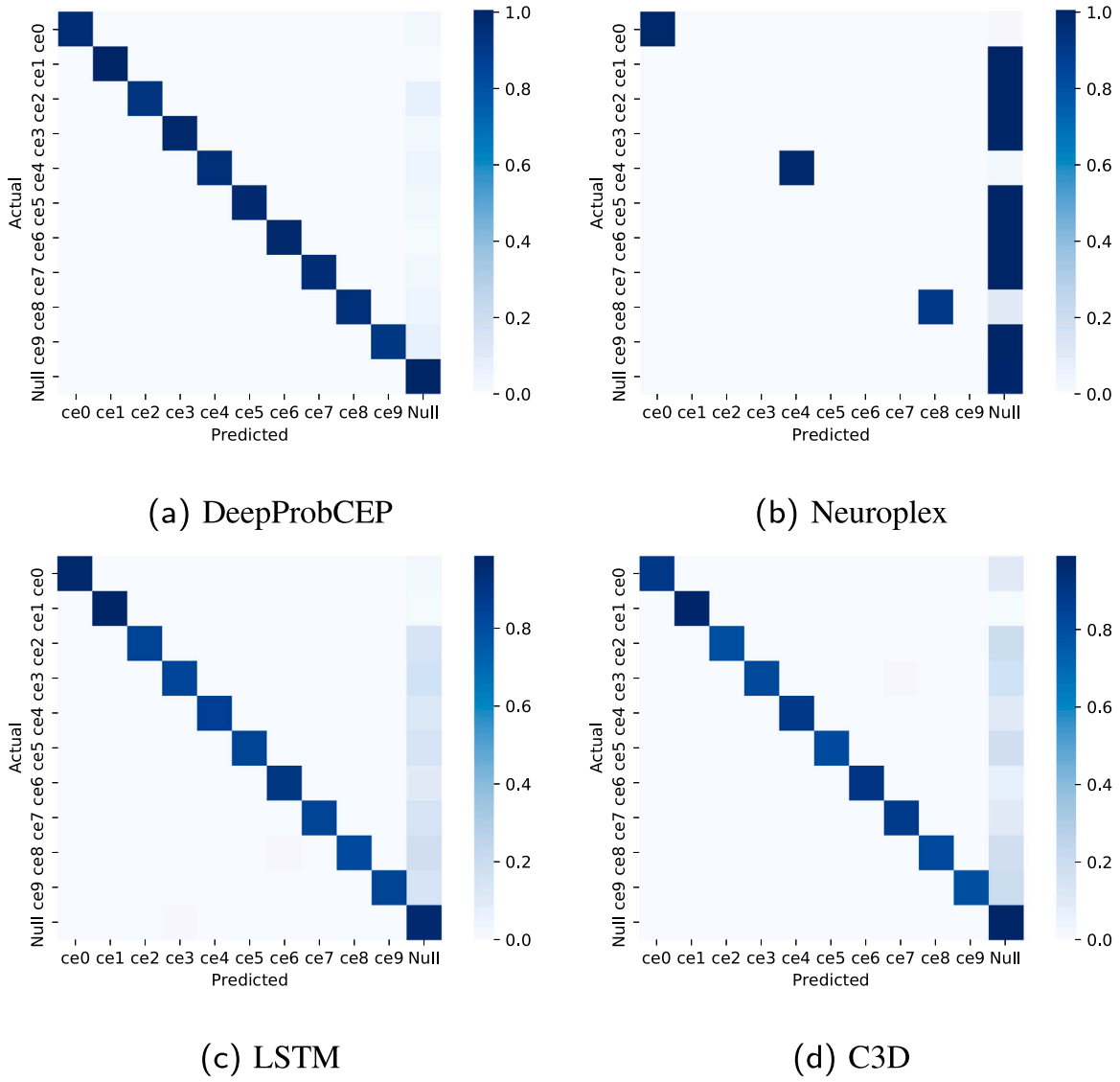


Fig. 7. Confusion matrices for Window of 2, using the best performing model from the data-scarcity datasets for each approach.

of human knowledge would prevent it from being able to learn to classify the simple events in a human intelligible way, as shown in Xing et al. (2020). Instead, the neural network would simply create its own representation of the simple events and rules, which would not be human-interpretable.

In Fig. 13, we can see a comparison on the performance of both DeepProbCEP and Neuroplex on single events. That is, it shows the accuracy performance of the low level neural network at classifying MNIST digits, for both approaches. Interestingly, the performance with Neuroplex does not seem to be very consistent with smaller windows, where it provides very poor performances. This is particularly true for a window size of 2, although it gives accuracy performances around 50% or lower in almost every case with a window size of 5 or smaller. This would seem to reinforce the hypothesis that Neuroplex does not work well with smaller window sizes as it is now. By comparison, we can see that DeepProbCEP performs quite well in any window size, especially when trained with 500 training points or more.

Furthermore, Neuroplex seems to have a relatively high standard deviation when compared to DeepProbCEP, even in the windows of 10 and 15 where Neuroplex seems to perform the best. This matches the results we saw for the system as a whole, which would seem to indicate that, at least for the problem we are looking at, DeepProbCEP provides more consistent results. As such, we can also consider DeepProbCEP more robust, as it will perform closer to our expected accuracy.

6.3. Time efficiency

However, one of the main disadvantages when using probabilistic logic programming in the architecture is a substantial decrease in speed when compared to a neural network. This means that DeepProbCEP is significantly slower in both training time and inference time. This is the reason why we have not tested how DeepProbCEP performs after training with the bigger dataset sizes, as this would take a significantly longer time than with other approaches.

The exact difference on time efficiency between DeepProbCEP and other approaches can vary depending on a number of factors, with the most important one seeming to be the window size of the dataset. Table 6 shows the training time efficiency of DeepProbCEP and Neuroplex in iterations per second (number of training points that can be processed each second). Higher values are better, as they allow the system to train with larger datasets in a shorter span of time. As we can see in Table 6, bigger window sizes tend to be slower, as there is more data to process. Furthermore, for the DeepProbCEP approach the logic layer will also increase in complexity as the window size increases,

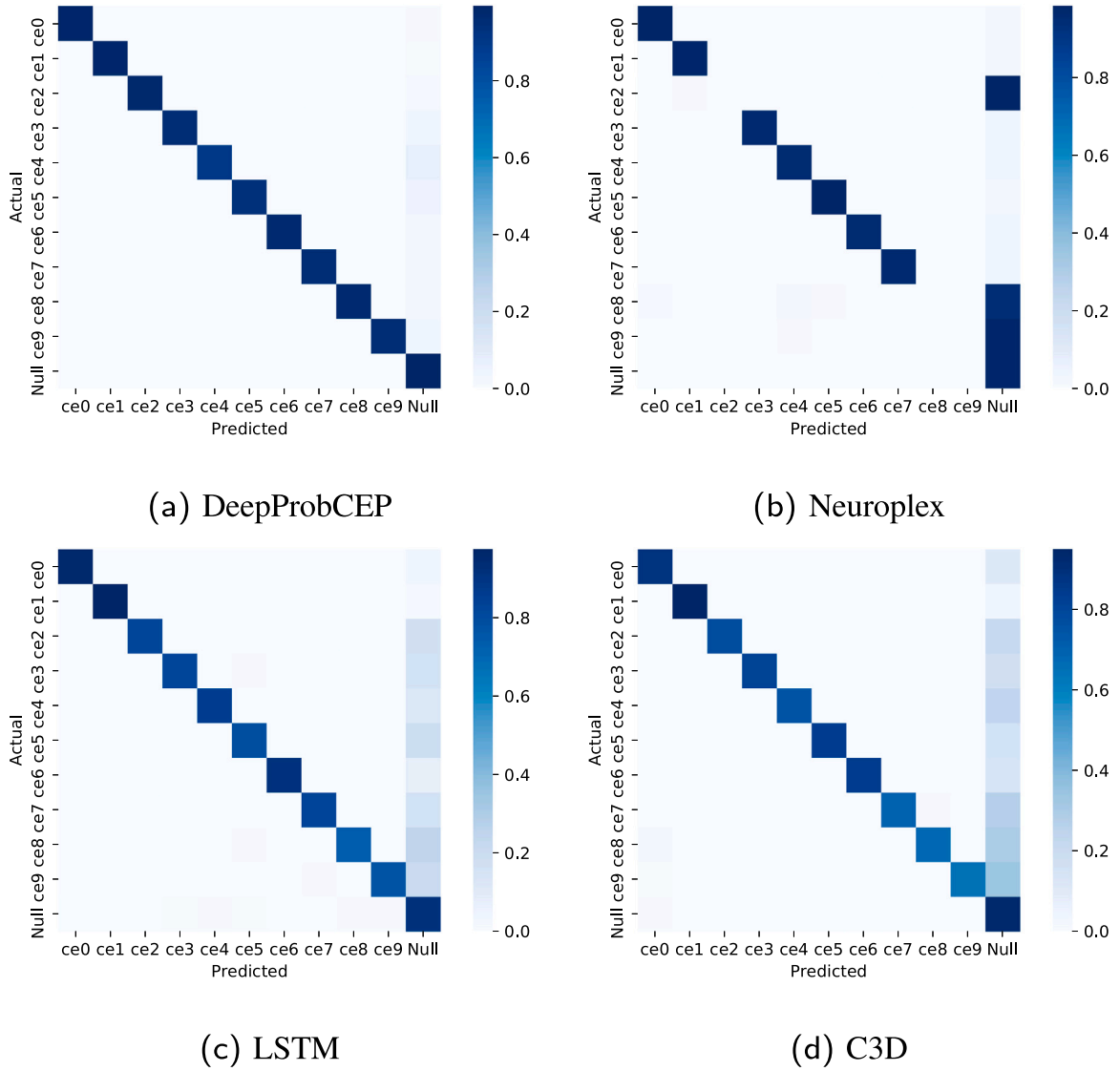


Fig. 8. Confusion matrices for Window of 3, using the best performing model from the data-scarcity datasets for each approach..

which will also decrease the speed. The same is true for the high level reasoning neural network used in Neuroplex, but the effect seems to be slightly lower, as the speed decreases at a slightly slower rate. The results show that Neuroplex is roughly 15 to 25 times faster than DeepProbCEP on training time. This is because Neuroplex is using a neural network to perform the high level reasoning, which is implemented using Tensorflow and Keras frameworks. This allows Neuroplex to make use of the highly optimized back-end code from this libraries, as well as easy access to batch training and GPU use to increase the speed of training. In contrast, DeepProbCEP makes use of DeepProbLog for the high level reasoning, which does not currently support batch training and is significantly less optimized for fast training than Tensorflow or Keras. As we have said, the most significant cost in terms of time comes from using probabilistic logic programming. More specifically, the cost of calculating the output probabilities based on the input probabilities that come from the prediction made by the neural network. In order to correctly calculate these output probabilities, the system needs to transform the logic defined by the user into an arithmetic circuit. Due to the current design of DeepProbLog, this transformation needs to be performed every epoch for every training case. As this transformation into an arithmetic circuit is, by far, the most time consuming step of the training process, this is a very significant time cost.

DeepProbLog does offer a cache system that allows us to generate this arithmetic circuit on the first epoch and then re-use it on further epochs. Our experiments have shown that using this cache system can increase the speed by as much as 4 times, and potentially more if combined with other improvements. While this would be slower than Neuroplex, the difference would be much less significant. However, this approach requires us to keep a cache with all of the arithmetic circuits in memory. This has prevented us from using it on the bigger datasets, such as data-scarcity datasets of a size bigger than 5000. This is because the amount of memory needed exceeds the limits of the RAM in our server. Therefore, we believe that further research is necessary to make the most out of this approach to increase the speed of the system.

6.4. Robustness against poisoning attack with random noise

After seeing that DeepProbCEP is capable of training even with small amounts of data, we also want to know how it performs under different types of adversarial attacks. For this purpose, we have prepared different types of poisoning attacks at different levels of noise, as explained in Section 4. For all of the graphs, the system has been tested with levels of noise between 0.0 and 1.0 (both included) with a step of 0.2. However, in order to make it easier to see the error bars in

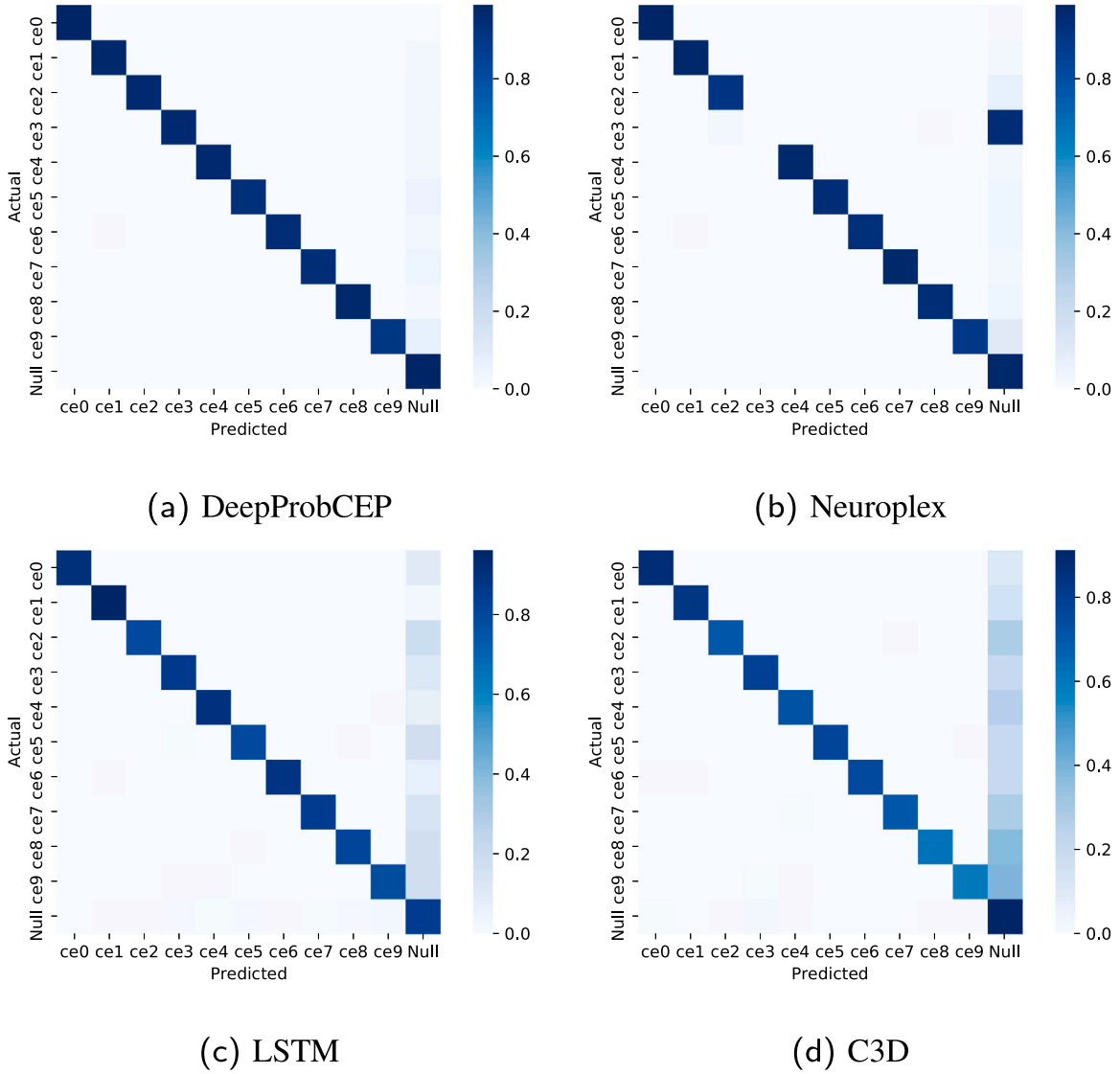


Fig. 9. Confusion matrices for Window of 4, using the best performing model from the data-scarcity datasets for each approach.

Table 6
Training time efficiency in iterations per second. That is, number of training points that can be processed each second (higher is better). Values obtained by calculating the training time for a single epoch of training and dividing it by the number of data points in the training dataset. The data-scarcity dataset of size 2500 was used for this experiment.

	Window size					
	2	3	4	5	10	15
DeepProbCEP	33.62	23.01	17.31	13.22	6.63	4.39
Neuroplex	500.00	333.33	333.33	333.33	142.86	111.11

the graph, these have been slightly offset in the x axis from their actual value.

In Fig. 14 we explore how the performance of DeepProbCEP is affected by random noise in the training data, meaning that a percentage of the labels have been randomly changed (as explained in Section 4.3 above). As can be seen in the graphs, DeepProbCEP performs very well even under significant levels of noise, only starting to be affected when almost all of the data is poisoned. As it should have been expected, there is a correlation between the accuracy when detecting the complex events and the accuracy on individual digits. This is because the performance for DeepProbCEP as a whole system is directly linked to its performance detecting individual events, assuming that the rules are correct.

Also as expected, with a noise percentage of 1.0 the system obtains a very bad accuracy, performing like a random classifier. This is because, in this situation, all of the data is noise, which means that nothing useful can be learnt from it.

6.5. Robustness against targeted poisoning attacks

The following sections evaluate how DeepProbCEP performs after training with the targeted poisoning datasets.

6.5.1. Targeted attack against a single label

First, we have a look at the targeted attack for a single label, where only one of the classes is affected by the attack. As we can see in Fig. 15, the performance for both the system as a whole and the individual digits is almost unaffected. In fact, even the performance on the dataset with a noise percentage of 1.0 might seem acceptable for some situations. This is because only one of the classes is being affected, meaning that the other 10 classes (the 9 other patterns plus the null class) are still performing the same. However, one of the classes

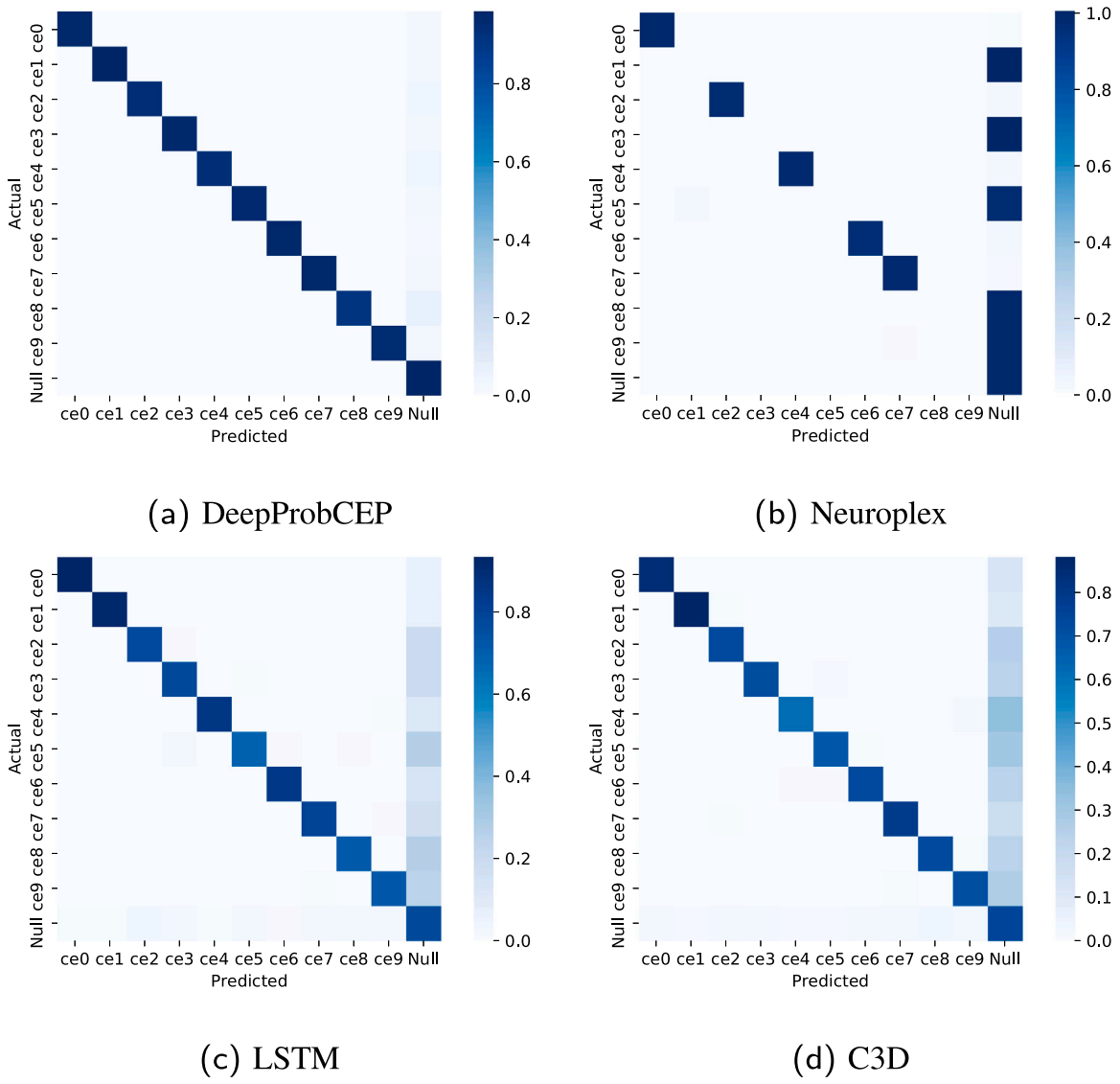


Fig. 10. Confusion matrices for Window of 5, using the best performing model from the data-scarcity datasets for each approach.

is being incorrectly classified a fairly large percentage of the time, which would significantly affect how the system performs in a real life scenario. This can be more clearly seen in the confusion matrices shown in Fig. 16, which shows the results of this attack on DeepProbCEP with a window of 4. The confusion matrices show that the targeted class tends to be miss-classified. With bigger noise percentages (Figs. 16(c) and 16(d)) the targeted class (in this case, *ce*2) tends to be classified as the substitute class (in this case, *ce*6). This is because, with high levels of noise, we are basically training the targeted class for the wrong label. Meanwhile, for lower percentages of noise the targeted class is mostly classified correctly. However, it is sometimes miss-classified as the null class (Figs. 16(a) and 16(b)). In all cases, the other classes are mostly unaffected.

Another way in which we are able to see how the different noise percentages affect the performance of the system is by looking at the recall for the targeted class, shown in Fig. 17. As we can see, the recall for both the affected digit and complex event class quickly decrease, ending up very close to 0 on higher noise levels. This shows that, if we want to make sure that none of the classes have been targeted it is necessary to check the performance for each of them individually.

6.5.2. Targeted attack over multiple labels

Next we look at the attack that affects all of the labels, the results of which can be seen in Fig. 18. As can be seen in the graphs, the performance for DeepProbCEP rapidly decays as we increase the percentage of noise present in the training data. This could have been expected since, again, what we are doing once the noise percentage is higher than 50% is training the neural network to recognize the incorrect label. This can be seen quite clearly in Fig. 19. Particularly in Fig. 19(d), where all classes are consistently being classified as their substitute.

However, it seems unlikely that such a big percentage of incorrect labels would make it to the training dataset without anyone realizing, as most of the training data would need to be consistently incorrectly labelled for this to happen. In contrast, for reasonable amounts of noise levels (such as 20% and, to a certain degree, 40%) DeepProbCEP still performs fairly well, which shows us that it should be fairly robust against this type of attack.

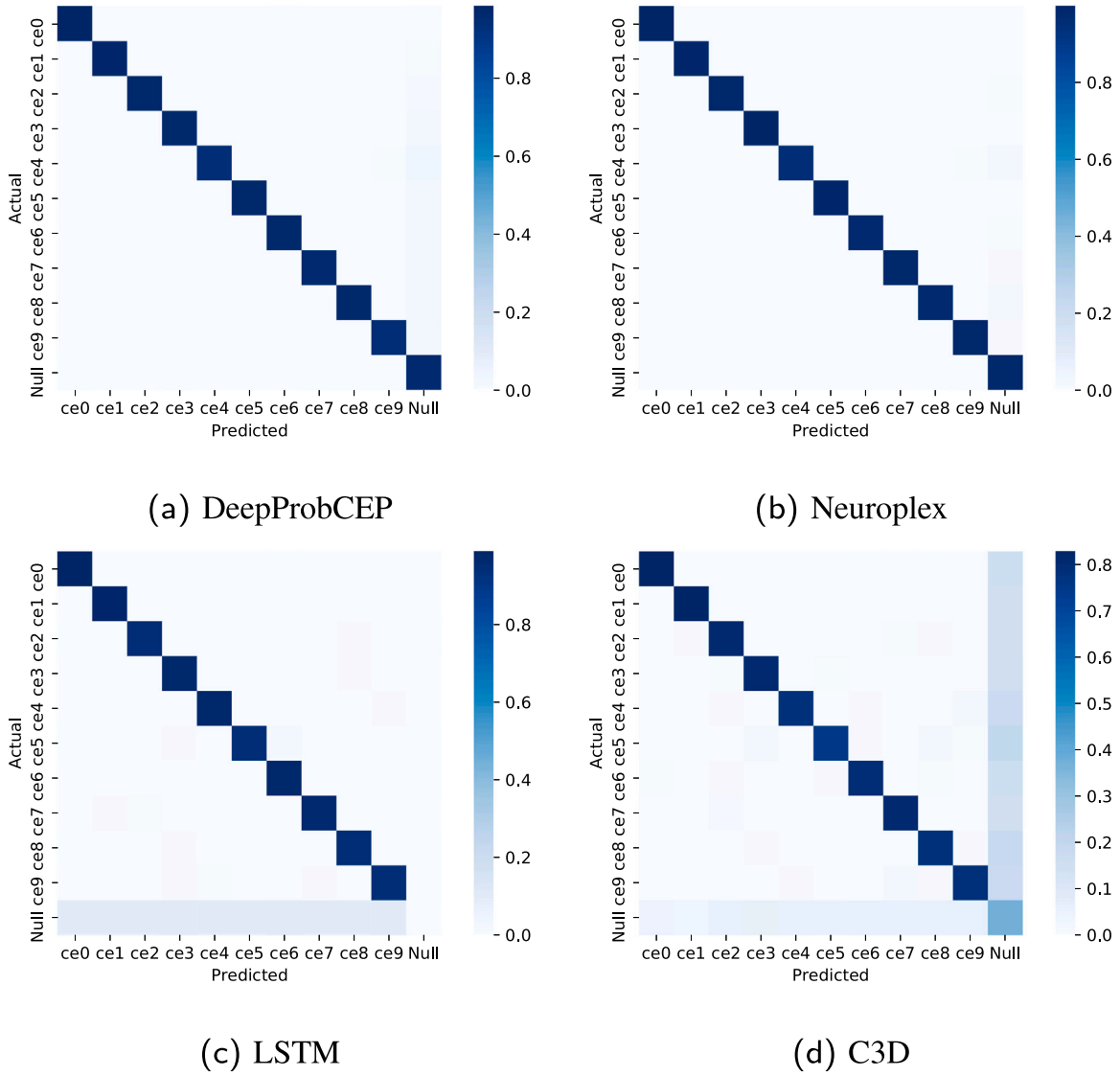


Fig. 11. Confusion matrices for Window of 10, using the best performing model from the data-scarcity datasets for each approach.

6.6. Performance when modifying the rules

In this experiment, we show how DeepProbCEP allows the user to modify the rules without requiring any re-training, as long as these changes do not require new types of simple events. In order to demonstrate this, we have trained DeepProbCEP with a data-scarcity dataset of size 5000 and a window of 5. After training with this dataset, the system archives a 97.5% accuracy when detecting complex events and 97.8% accuracy on simple events.

Then, we modify the complex event rules in the reasoning layer of DeepProbCEP. The new rules, which can be seen in Listing 3, detect complex event $ce0$ when a 0 appears in the window and a 1 appears in the last position of the window, without a 9 appearing between them. $ce1$ is detected if a 1 appears in the window and a 2 appears in the last position without a 9 between them, and so on.

Note that the following important points have changed:

- The number of complex event types has been reduced from 10 to 8.
- The window size has changed from 5 to 10. This is defined in the first line of Listing 3.

```
1  window(10).
2
3  happensAt(ce0, T) :- window(Window),
4      sequence([0, not(9), 1], Window, T).
5  happensAt(ce1, T) :- window(Window),
6      sequence([1, not(9), 2], Window, T).
7  happensAt(ce2, T) :- window(Window),
8      sequence([2, not(9), 3], Window, T).
9  happensAt(ce3, T) :- window(Window),
10     sequence([3, not(9), 4], Window, T).
11 happensAt(ce4, T) :- window(Window),
12     sequence([4, not(9), 5], Window, T).
13 happensAt(ce5, T) :- window(Window),
14     sequence([5, not(9), 6], Window, T).
15 happensAt(ce6, T) :- window(Window),
16     sequence([6, not(9), 7], Window, T).
17 happensAt(ce7, T) :- window(Window),
18     sequence([7, not(9), 8], Window, T).
```

Listing 3: New complex event rules to evaluate how DeepProbCEP performs after changing the complex event rule definitions.

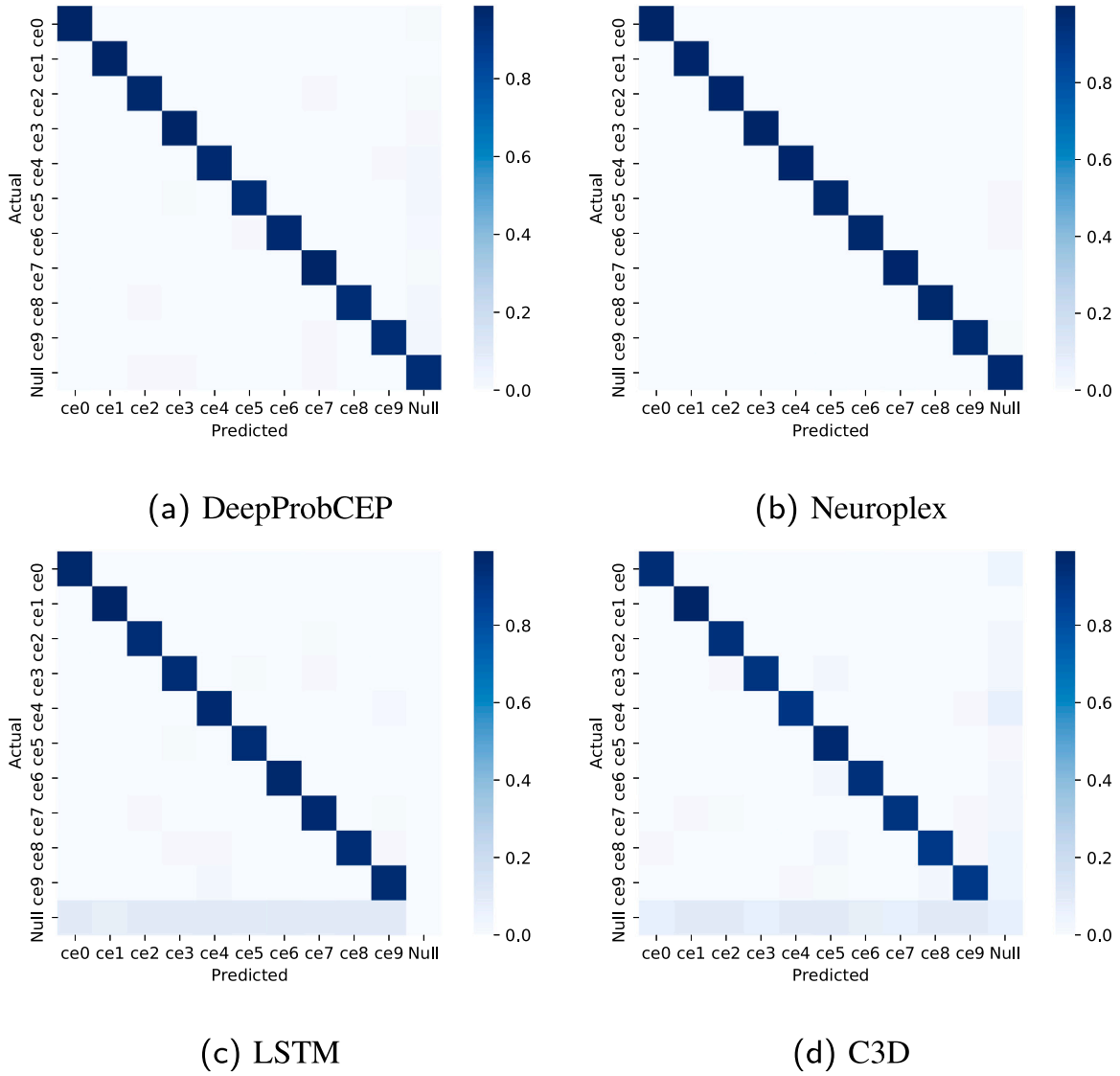


Fig. 12. Confusion matrices for Window of 15, using the best performing model from the data-scarcity datasets for each approach.

- The definition for each of the complex events is significantly different. Most notably, these definitions make use of the negated value, which indicates that a certain simple event does not occur between two other complex events. This was not used previously as the current implementation of Neuroplex does not allow for this definition.

Finally, the updated system is evaluated against a testing dataset generated with those complex event rules. It is important to note that the perception level neural network uses the weights resulting from the training with the first dataset, and no further training is done at all. As such, the performance on the simple events remains the same, as it should have been expected. More interesting, however, is the fact that the updated system archives a 96.3% accuracy on the testing dataset with the new complex event rules. This demonstrates that DeepProbCEP allows the user to modify the rules without requiring any further training. It would also be possible to make much smaller modifications to refine the definition for the complex events. However, for demonstration purposes, we wanted to make it clear that this was not limited to small changes. In fact, the logic layer can be modified as much as the user desires as long as the simple event classes remain the same.

6.7. Performance in an audio setting

In this experiment, we show that DeepProbCEP can also be used to work with other types of input data. In this case, we use it to work with an audio stream, instead of the MNIST digits used in the previous sections. For this purpose, we have created another synthetically generated dataset following the same steps described in Section 4 with the only difference being that instead of using MNIST images as the simple events, we are using samples of audio of 1 s of length each. These audio samples have been obtained from the dataset Urban Sounds 8K (Salamon, Jacoby, & Bello, 2014), which contains short labelled sound excerpts (4 s or less) of 10 classes of urban sounds. In order to standardize the length of the sound files, only the first second of each of the files has been used (with the small number of files shorter than that length being ignored).

The following changes have been made to use DeepProbCEP with an audio stream:

- In order to extract as much information from the input data, we use a more sophisticated pre-processing. More specifically, we are using a PyTorch implementation of VGGish, a feature embedding

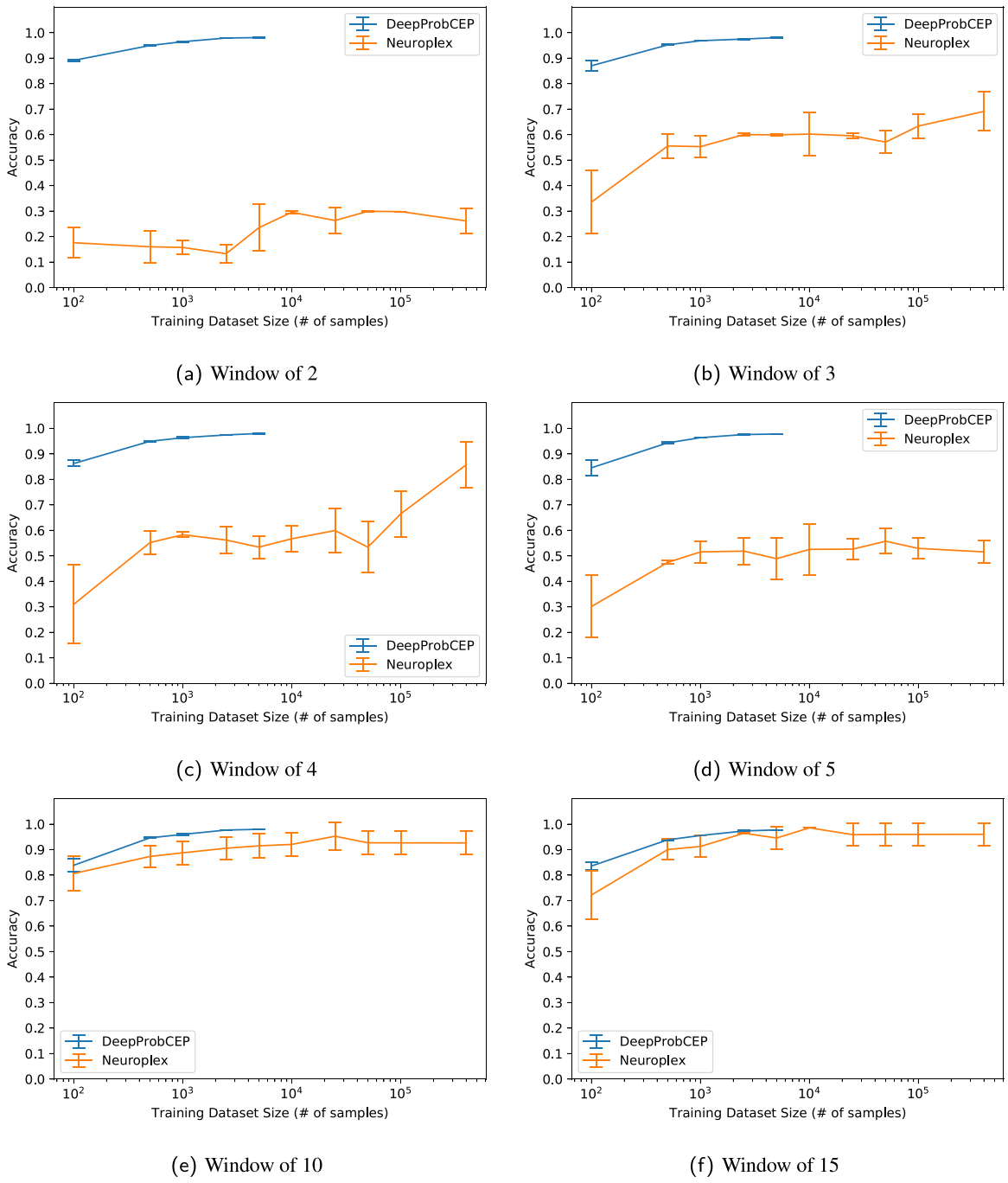


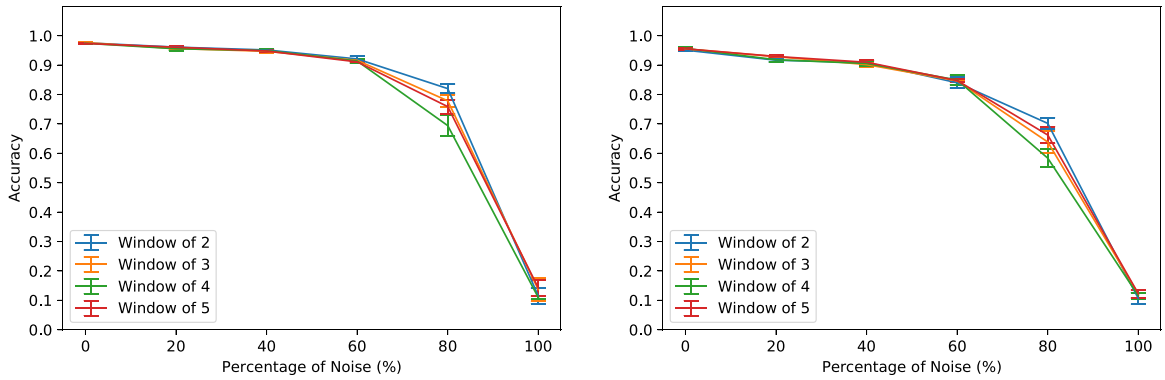
Fig. 13. Accuracy for simple events (individual MNIST digits) for DeepProbCEP and Neuroplex in different windows (the other approaches cannot provide simple event classifications). Horizontal axis indicates the size of the training dataset used (in number of samples in the dataset).

frontend for audio classification models (Hershey et al., 2017).[4] VGGish takes an audio file as an input and outputs a matrix $128 \times N$—with N the length of the audio file in seconds—with values between 1 and 255.

- The neural network used to classify the simple events into their classes has also been changed. Thanks to the fact that the pre-processing already extracts most of the information, we are able to use a simple Multi Layer Perceptron (MLP) to obtain a fairly good accuracy when classifying the audio segments. This MLP consists of 5 linear layers, with 100, 80, 50, 25 and 10 neurons each, respectively. A ReLU activation function is used between each linear layer, and a Softmax activation function is used after the last layer.
- Finally, the rules for recognizing the complex events have been changed. Since we have generated the dataset with the same rules where two instances of the same class within the window size constitute a complex event, we only needed to change the names of the identifiers for the different classes. However, if we wanted to detect other patterns that are more interesting to us this could also be easily done.

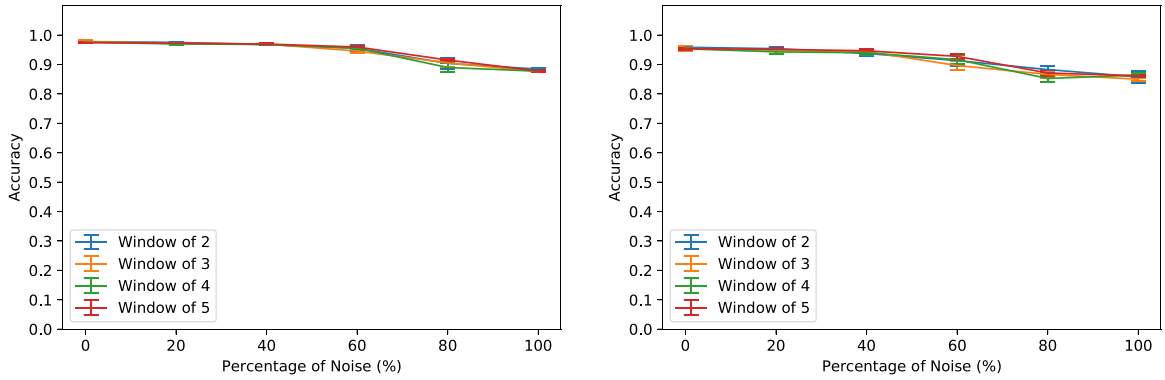
In Table 7 we can see the results of training DeepProbCEP on a balanced dataset with 1000 training data points. We can see that

[4] Available at https://github.com/harritaylor/torchvggish.



(a) Digit accuracy (b) Complex Event accuracy

Fig. 14. In this scenario, we test the performance of the system with different levels of noise in the training data for window sizes between 2 and 5. The horizontal axis indicates the percentage of training data points where the class has been randomly selected (0% for none and 100% for all). The vertical axis indicates the accuracy of the system for individual classifications and complex event detection, respectively.



(a) Digit accuracy (b) Complex Event accuracy

Fig. 15. In this scenario, we test the performance of the system with different levels of noise in the training data for window sizes between 2 and 5. The horizontal axis indicates the percentage of training data points of the chosen class that have been swapped for its pre-defined substitute class (0% for none and 100% for all). The vertical axis indicates the accuracy of the system for individual classifications and complex event detection, respectively.

Table 7
Accuracy results (average and standard deviation over 3 executions) for DeepProbCEP for both complex events and individual sounds.

Window size	Complex event Acc	Complex event STD	Sound Acc	Sound STD
2	0.8657	0.0041	0.6696	0.0100
3	0.7645	0.0109	0.6497	0.0265
4	0.7069	0.0191	0.6501	0.0284
5	0.6401	0.0225	0.6410	0.0694

DeepProbCEP still obtains a fairly high accuracy on this dataset, despite the fact that the problem is significantly more complex. We have also performed the same robustness experiments described above for the MNIST approach on this audio case. The results are consistent with those obtained in the MNIST case, albeit with lower performance, as expected. These results can be seen in Appendix B.

7. Discussion

In this paper, we have presented DeepProbCEP, a neuro-symbolic approach capable of performing CEP on subsymbolic data. We have demonstrated that DeepProbCEP can work with images and audio as input data, fulfilling the first objective defined in Section 1.

We have also shown how the reasoning layer of DeepProbCEP can be changed even after training, and explained how this reasoning layer can be used to define any configuration of complex events, as the whole expressivity of ProbLog is available to describe them. This is further proven by the demonstration shown as the motivating example, which makes use of a far more complex definition for the complex event. As such, we would argue that the second objective has also been fulfilled.

DeepProbCEP has also outperformed the neural-only approaches and Neuroplex in terms of accuracy, particularly when using sparse data. This, together with DeepProbCEP's capability of using end-to-end training, makes is easy and cheap to obtain enough data to train the system, thus fulfilling the third objective.

Finally, DeepProbCEP's robustness against adversarial attacks on the training data fulfils the fourth objective. This paper has focused on attacks on the training data, as we believe that these are quite feasible to happen, even by accident. Despite this, further experiments should be performed to evaluate DeepProbCEP's robustness against other types of attacks.

Furthermore, we have also demonstrated that DeepProbCEP can train neural networks to classify simple events, even when training in an end-to-end manner. This could be a useful byproduct, as it means that we can train neural networks to classify data even when we do not have direct labelling of this data by injecting human knowledge into the system.

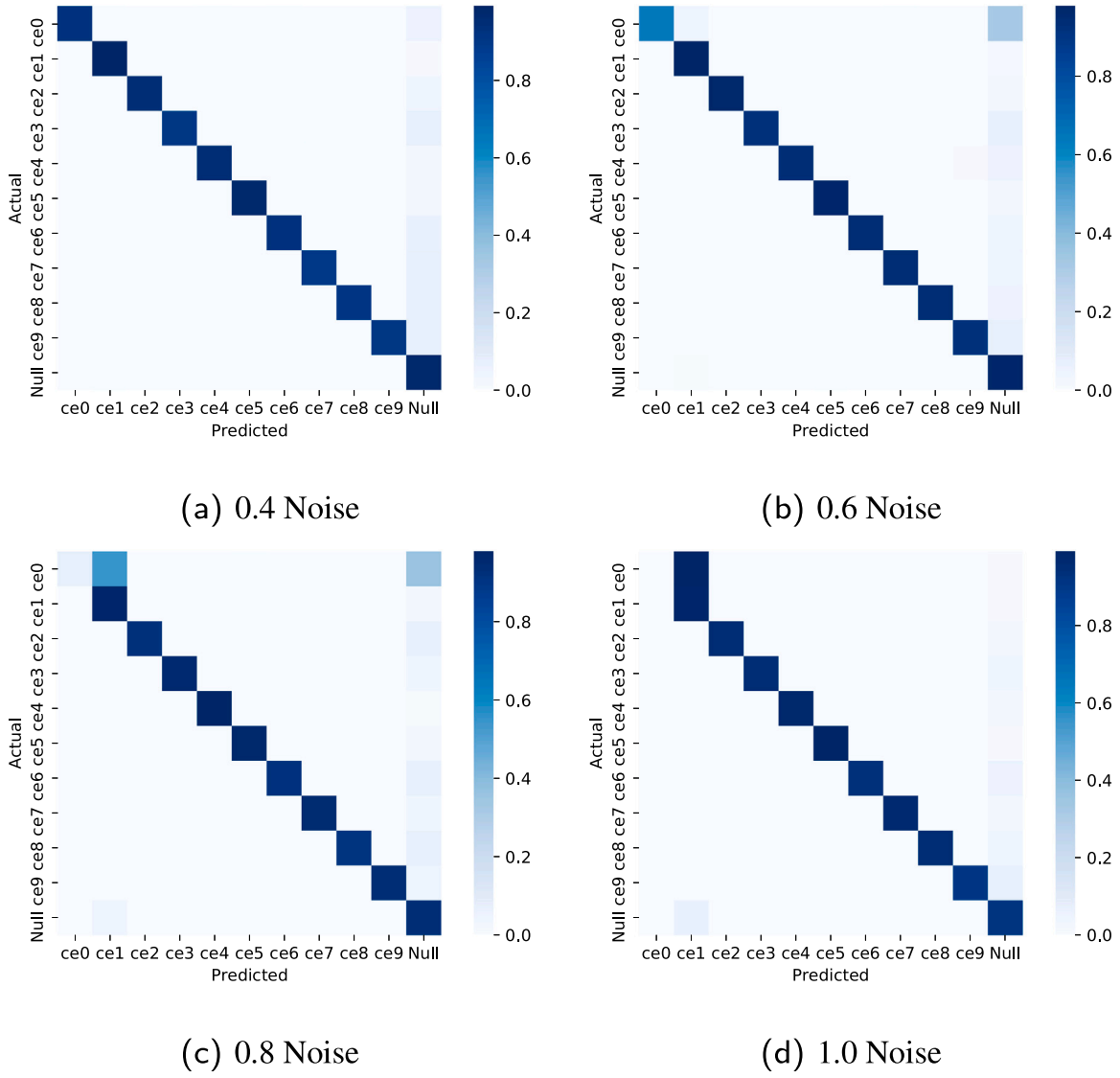


Fig. 16. Confusion matrices for DeepProbCEP after being trained with datasets under a Single Label attack with different noise percentages. All confusion matrices show results for a window of 4. For this case, the targeted class was $ce0$ and the substitute was $ce1$.

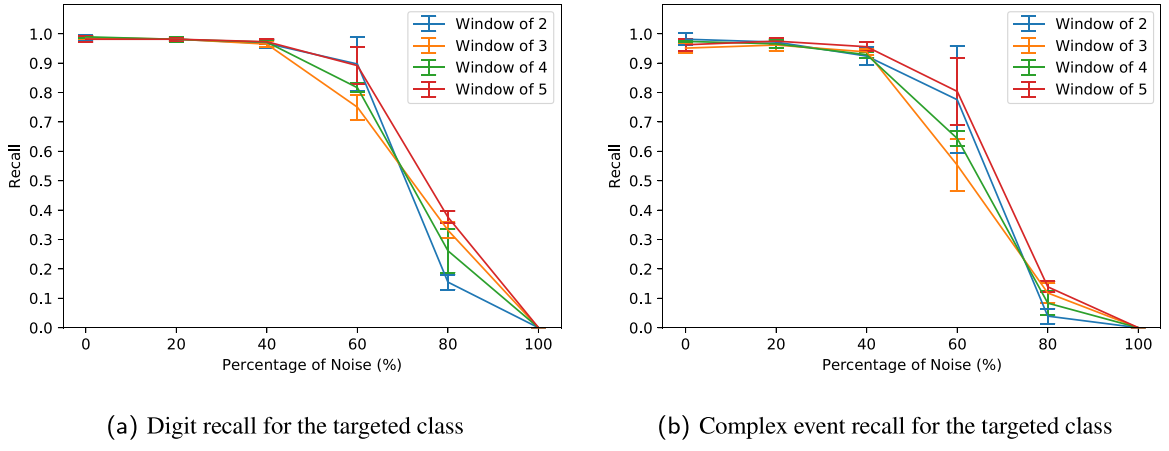


Fig. 17. In this scenario, we test the performance of the system for the chosen class with different levels of noise in the training data for window sizes between 2 and 5. The horizontal axis indicates the percentage of training data points of the chosen class that have been swapped for its pre-defined substitute class (0% for none and 100% for all). The vertical axis indicates the recall for the chosen class, either individually or in the pattern formed by that class.

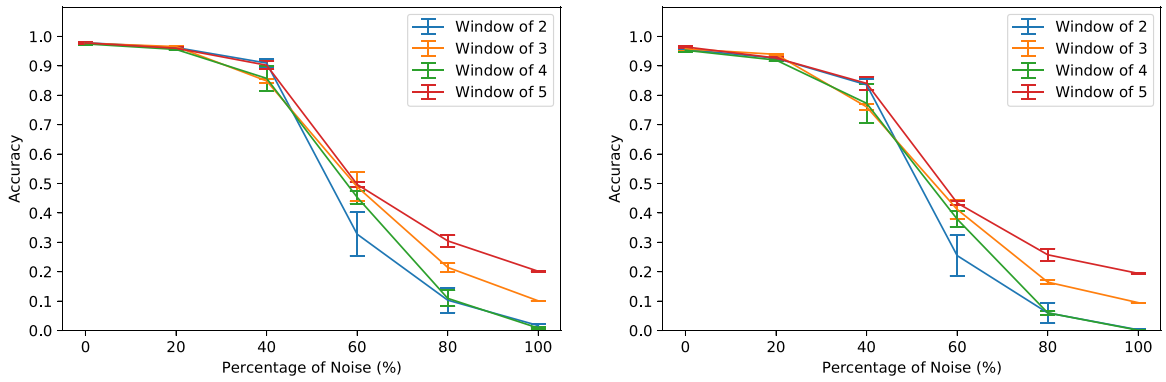


Fig. 18. In this scenario, we test the performance of the system with different levels of noise in the training data for window sizes between 2 and 5. The horizontal axis indicates the percentage of training data points where the class has been swapped for its pre-defined substitute (0% for none and 100% for all). The vertical axis indicates the accuracy of the system for individual classifications and complex event detection, respectively.

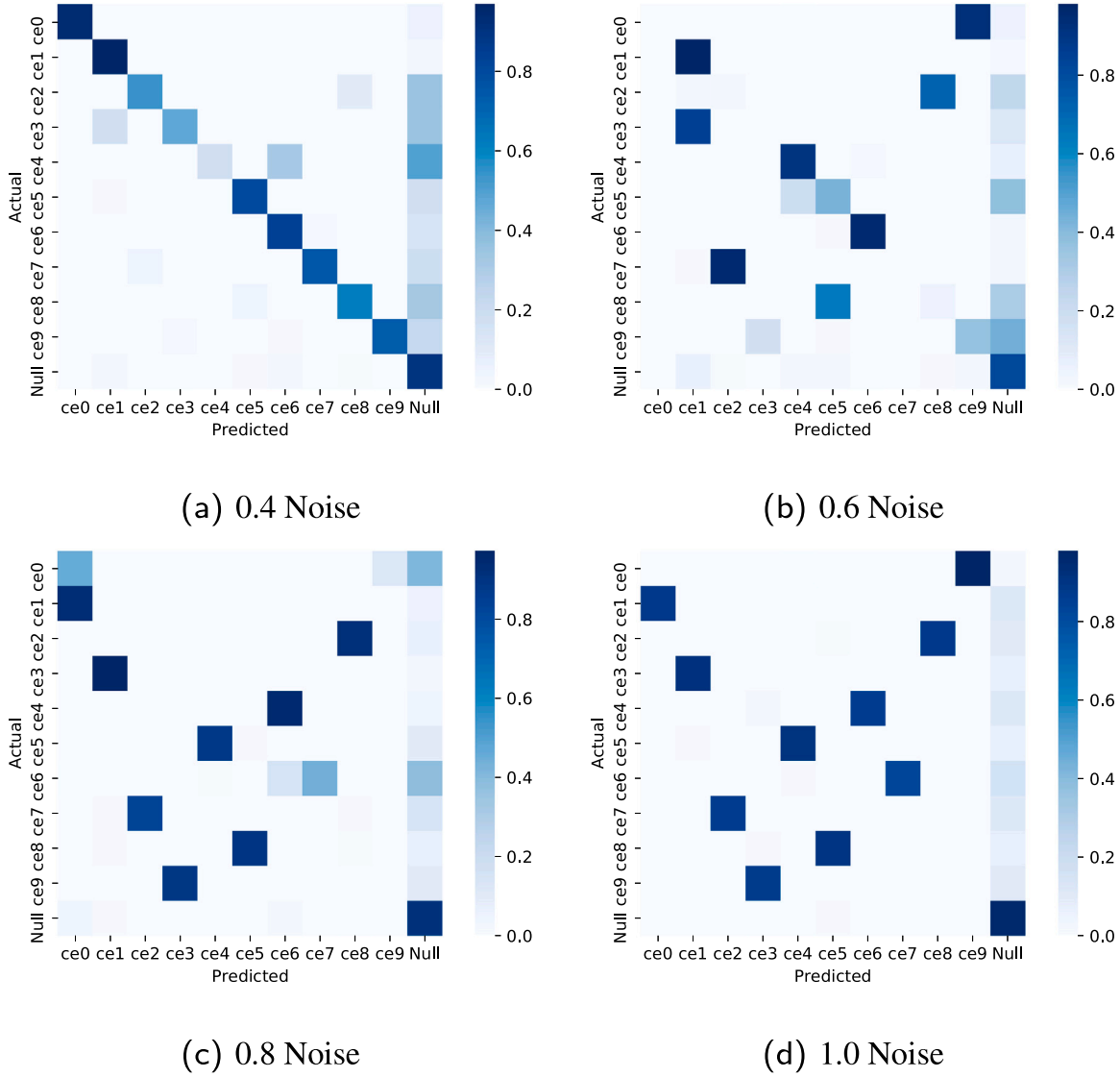


Fig. 19. Confusion matrices for DeepProbCEP after being trained with datasets under Multiple Labels attack with different noise percentages. All confusion matrices show results for a window of 4.

Despite this, DeepProbCEP does have some limitations, which future work should aim to reduce.

7.1. DeepProbCEP's limitations

While the use of a logic layer does offer a number of advantages that make it possible to fulfil the objectives defined above, it does also introduce some limitations. The most relevant limitation is the significantly slower training time, which can prevent users from training with particularly big datasets. These limitations come from the fact that the logical inference is significantly slower than propagating the values through a logic layer. This is partly due to implementation inefficiencies, as the ProbLog inference is likely not as optimized as the machine learning libraries used to implement Neuroplex. However, a more significant time loss comes from the fact that the logic program needs to be complied each time an inference is performed, which is a time consuming process.

Another of DeepProbCEP's limitations is the fact that it requires the user to provide the rules for the complex events, which must be correctly defined. While this is quite standard for CEP approaches, it does mean that DeepProbCEP cannot be used in situations where such rules are incorrect or not available at all. In those cases, DeepProbCEP would not provide any significant advantage against the neural-only approaches, as the lack of injected knowledge would make it impossible to reduce the amount of training data required. Furthermore, in the case where the provided rules are incorrect this could even negatively impact the training making it worse than neural-only approaches, as DeepProbCEP would be learning incorrect values. As such, ensuring that rules are provided and that such rules are correct is a must when using DeepProbCEP.

8. Conclusion and future work

As demonstrated above, DeepProbCEP fulfils the four objectives we defined in Section 1. However, further work could be performed to reduce the limitations of the approach.

As explained in Section 6.3, we believe that future research could be done on improving the time efficiency of DeepProbLog, which could significantly reduce the difference in time efficiency between DeepProbCEP and Neuroplex.

Another aspect in which future research could be performed is in automatically learning the rules for complex events, even in situations where subsymbolic data is used as an input. As explained in Section 2, some approaches exist that are capable of learning the complex event rules when using symbolic data. This is useful when experts are not able to define the rules for the complex events. When using subsymbolic data, we have shown that a neural network approach can be used. However, as we have also shown, this requires very large amounts of training data. As such, we believe that an inductive logic programming approach, combined with a neuro-symbolic architecture, would be a better approach when dealing with sparse data. This could reduce the need for user defined rules, making it possible to use DeepProbCEP in situations where these are not available.

CRediT authorship contribution statement

Marc Roig Vilamala: Conceptualization, Methodology, Software, Investigation, Writing – original draft, Visualization. **Tianwei Xing:** Software. **Harrison Taylor:** Software. **Luis Garcia:** Writing – review & editing. **Mani Srivastava:** Conceptualization, Writing – review & editing. **Lance Kaplan:** Conceptualization, Writing – review & editing. **Alun Preece:** Conceptualization, Writing – review & editing, Supervision. **Angelika Kimmig:** Writing – review & editing, Supervision. **Federico Cerutti:** Conceptualization, Writing – review & editing, Supervision.

Declaration of competing interest

One or more of the authors of this paper have disclosed potential or pertinent conflicts of interest, which may include receipt of payment, either direct or indirect, institutional support, or association with an entity in the biomedical field which may be perceived to have potential conflict of interest with this work. For full disclosure statements refer to https://doi.org/10.1016/j.eswa.2022.119376. This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001.

Data availability

The data and code have been publicly uploaded to Code Ocean: https://codeocean.com/capsule/7055867/tree/v1.

Acknowledgements

This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

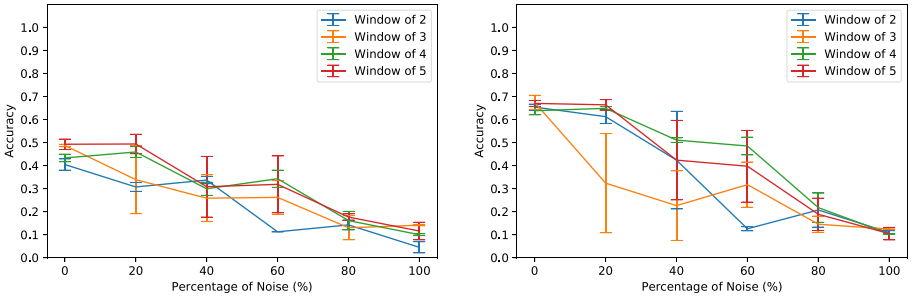
Appendix A. Sequence framework: a tool to detect patterns in streams of data

```
% Main interface for the framework.
sequence(L, W, T) :-
   % Reverse list to simplify rule definitions
   reverse(L, L2),
   sequenceEndingAt(L2, W, T).

% An empty sequence will be within if Excluded
% elements is empty or if W is 0
sequenceWithin([], [], _, _).
sequenceWithin([], _, 0, _).
sequenceEndingAt([X | L], W, T) :-
   W > 0, T >= 0, happensAt(Y, T), wrapper(Y, X),
   NextW is W - 1, allTimeStamps(Timestamps),
   previousTimeStamp(T, Timestamps, Tprev),
   sequenceWithin(L, [], NextW, Tprev).
% If we have detected all simple events but still
% have Excluded elements, check until the end of the window
sequenceWithin([], E, W, T) :-
   W > 0, happensAt(H, T), wrapper(H, Y),
   \+ itemIn(Y, E), NextW is W - 1,
   allTimeStamps(Timestamps),
   previousTimeStamp(T, Timestamps, Tprev),
   sequenceWithin([], E, NextW, Tprev).
% A sequence can be within T if it ends at T
sequenceWithin(L, E, W, T) :- sequenceEndingAt(L, W, T).
% If the next element in the list is a negation, add it to E
sequenceWithin([X | L], E, W, T) :-
   isNegation(X, Y), sequenceWithin(L, [Y | E], W, T).
% A sequence can be within W of T if it is within NextW of Tprev
sequenceWithin([X | L], E, W, T) :-
   W > 0, T >= 0, \+ isNegation(X, _), happensAt(H, T),
   wrapper(H, Y), \+ itemIn(Y, E),
   NextW is W - 1, allTimeStamps(Timestamps),
   previousTimeStamp(T, Timestamps, Tprev),
   sequenceWithin([X | L], E, NextW, Tprev).
```

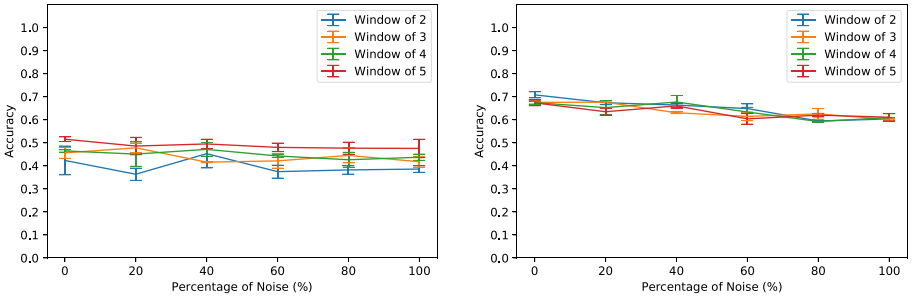
Appendix B. Audio robustness results

See Figs. B.20–B.23.



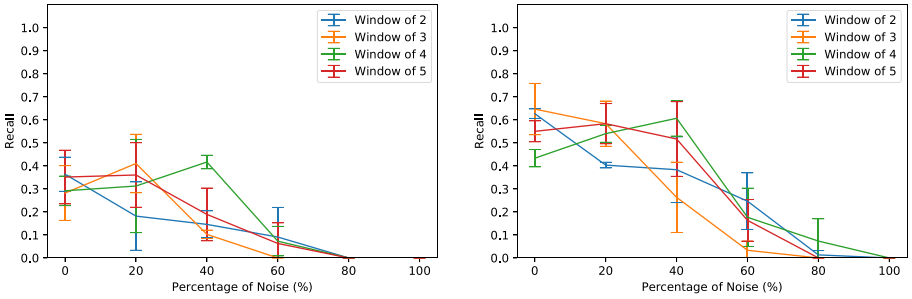
(a) Complex Event Accuracy (b) Sound Accuracy

Fig. B.20. Accuracy results for Random noise for the UrbanSounds setting.



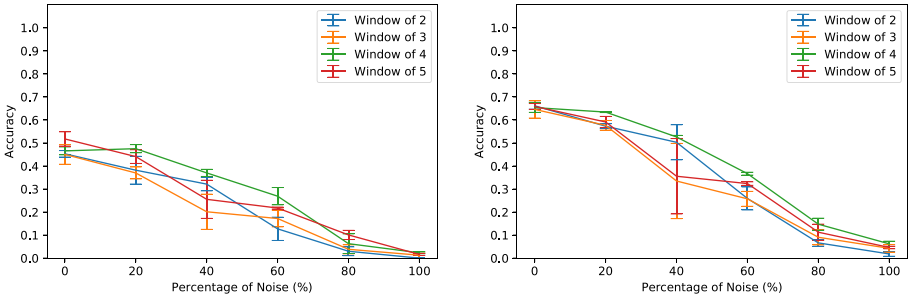
(a) Complex Event Accuracy (b) Sound Accuracy

Fig. B.21. Accuracy results for Targeted attack against a single label for the UrbanSounds setting.



(a) Complex event recall (b) Sound recall

Fig. B.22. Recall results for the targeted simple and complex events under the Targeted attack against a single label for the UrbanSounds setting.



(a) Complex Event Accuracy (b) Sound Accuracy

Fig. B.23. Accuracy results for Targeted attack against multiple labels for the UrbanSounds setting.

Appendix C. Supplementary data

The supplementary data for this article contains a video of the demonstration discussed in Section 1.1.

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.eswa.2022.119376.

References

Al-Rakhami, M. S., Islam, M. M., Islam, M. Z., Asraf, A., Sodhro, A. H., & Ding, W. (2021). Diagnosis of COVID-19 from X-rays using combined CNN-RNN Architecture with transfer learning. *MedRxiv*, http://dx.doi.org/10.1101/2020.08.24.20181339, URL: https://www.medrxiv.org/content/early/2021/08/09/2020.08.24.20181339.

Alevizos, E., Skarlatidis, A., Artikis, A., & Paliouras, G. (2017). Probabilistic complex event recognition: A survey. *ACM Computing Surveys*, *50*(5), http://dx.doi.org/10.1145/3117809.

Anicic, D., Rudolph, S., Fodor, P., & Stojanovic, N. (2012a). Real-time complex event recognition and reasoning-a logic programming approach. *Applied Artificial Intelligence - AAI*, *26*, 6–57. http://dx.doi.org/10.1080/08839514.2012.636616.

Anicic, D., Rudolph, S., Fodor, P., & Stojanovic, N. (2012b). Stream reasoning and complex event processing in ETALIS. *Semantic Web*, *3*, 397–407. http://dx.doi.org/10.3233/SW-2011-0053, URL: https://content.iospress.com/download/semantic-web/sw053?id=semantic-web%2Fsw053.

Bezerra, E. D. C., Teles, A. S., Coutinho, L. R., & da Silva e Silva, F. J. (2021). Dempster–Shafer theory for modeling and treating uncertainty in IoT applications based on complex event processing. *Sensors*, *21*(5), http://dx.doi.org/10.3390/s21051863, URL: https://www.mdpi.com/1424-8220/21/5/1863.

Bruns, R., Dunkel, J., & Offel, N. (2019). Learning of complex event processing rules with genetic programming. *Expert Systems with Applications*, *129*, 186–199. http://dx.doi.org/10.1016/j.eswa.2019.04.007, URL: https://www.sciencedirect.com/science/article/pii/S0957417419302386.

Burgueño, L., Boubeta-Puig, J., & Vallecillo, A. (2018). Formalizing complex event processing systems in Maude. *IEEE Access*, *6*, 23222–23241. http://dx.doi.org/10.1109/ACCESS.2018.2831185.

Chapnik, K., Kolchinsky, I., & Schuster, A. (2021). DARLING: Data-aware load shedding in complex event processing systems. *Proceedings of the VLDB Endowment*, *15*(3), 541–554. http://dx.doi.org/10.14778/3494124.3494137.

De Raedt, L., Kimmig, A., & Toivonen, H. (2007). ProbLog: A probabilistic prolog and its application in link discovery. In *IJCAI international joint conference on artificial intelligence* (pp. 2468–2473). URL: www.ncbi.nlm.nih.gov/Entrez/.

Defense Innovation Board (2019). AI principles: Recommendations on the ethical use of artificial intelligence by the department of defense. In *Supporting document*. Defense Innovation Board.

Giatrakos, N., Alevizos, E., Artikis, A., Deligiannakis, A., & Garofalakis, M. (2020). Complex event recognition in the Big Data era: a survey. *The VLDB Journal*, *29*(1), 313–352. http://dx.doi.org/10.1007/s00778-019-00557-w.

Hershey, S., Chaudhuri, S., Ellis, D. P., Gemmeke, J. F., Jansen, A., Moore, R. C., et al. (2017). CNN architectures for large-scale audio classification. In *2017 Ieee international conference on acoustics, speech and signal processing (Icassp)* (pp. 131–135). IEEE.

Hinton, G., Vinyals, O., & Dean, J. (2015). Distilling the knowledge in a neural network. http://dx.doi.org/10.48550/ARXIV.1503.02531, URL: https://arxiv.org/abs/1503.02531.

Hu, Z., Ma, X., Liu, Z., Hovy, E. H., & Xing, E. P. (2016). Harnessing deep neural networks with logic rules. CoRR, arXiv:1603.06318 URL: http://arxiv.org/abs/1603.06318.

Islam, M. Z., Islam, M. M., & Asraf, A. (2020). A combined deep CNN-LSTM network for the detection of novel coronavirus (COVID-19) using X-ray images. *Informatics in Medicine Unlocked*, *20*, http://dx.doi.org/10.1016/j.imu.2020.100412, URL: https://www.sciencedirect.com/science/article/pii/S2352914820305621.

Kowalski, R., & Sergot, M. (1986). A logic-based calculus of events. *New Generation Computing*, *4*(1), 67–95. http://dx.doi.org/10.1007/BF03037383.

Liu, K., Liu, W., Gan, C., Tan, M., & Ma, H. (2018). T-C3D: Temporal convolutional 3D network for real-time action recognition. In *Proceedings of the AAAI conference on artificial intelligence, Vol. 32*.

Manhaeve, R., Dumancic, S., Kimmig, A., Demeester, T., & De Raedt, L. (2018). DeepProbLog: Neural probabilistic logic programming. In *NIPS2018* (pp. 3749–3759).

Manhaeve, R., Dumanić, S., Kimmig, A., Demeester, T., & De Raedt, L. (2021). Neural probabilistic logic programming in DeepProbLog. *Artificial Intelligence*, *298*, Article 103504. http://dx.doi.org/10.1016/j.artint.2021.103504, URL: https://www.sciencedirect.com/science/article/pii/S0004370221000552.

Mishra, S., Jain, M., Siva Naga Sasank, B., & Hota, C. (2018). An ingestion based analytics framework for complex event processing engine in internet of things. In A. Mondal, H. Gupta, J. Srivastava, P. K. Reddy, & D. Somayajulu (Eds.), *Big data analytics* (pp. 266–281). Cham: Springer International Publishing.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., et al. (2017). Automatic differentiation in PyTorch. In *NIPS workshop*.

Roig Vilamala, M. (2022). LiveEvents demo - based on DeepProbCEP training. URL: https://www.youtube.com/watch?v=dllH0VzppPM.

Roig Vilamala, M., Hiley, L., Hicks, Y., Preece, A., & Cerutti, F. (2019). A pilot study on detecting violence in videos fusing proxy models. In *2019 22th international conference on information fusion (FUSION)* (pp. 1–8).

Roldán, J., Boubeta-Puig, J., Luis Martínez, J., & Ortiz, G. (2020). Integrating complex event processing and machine learning: An intelligent architecture for detecting IoT security attacks. *Expert Systems with Applications*, *149*, Article 113251. http://dx.doi.org/10.1016/j.eswa.2020.113251, URL: https://www.sciencedirect.com/science/article/pii/S0957417420300762.

Salamon, J., Jacoby, C., & Bello, J. P. (2014). A dataset and taxonomy for urban sound research. In *22nd ACM international conference on multimedia (ACM-MM'14)* (pp. 1041–1044). Orlando, FL, USA.

Shi, B., Bai, X., & Yao, C. (2015). An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition. CoRR, arXiv:1507.05717 URL: http://arxiv.org/abs/1507.05717.

Skarlatidis, A., Artikis, A., Filippou, J., & Paliouras, G. (2015). A probabilistic logic programming event calculus. *Theory and Practice of Logic Programming*, *15*(2), 213–245. http://dx.doi.org/10.1017/S1471068413000690.

Teymourian, K., Rohde, M., & Paschke, A. (2012). Knowledge-based processing of complex stock market events. In *Proceedings of the 15th international conference on extending database technology* (pp. 594–597). New York, NY, USA: Association for Computing Machinery, http://dx.doi.org/10.1145/2247596.2247674.

Xing, T., Garcia, L., Vilamala, M. R., Cerutti, F., Kaplan, L., Preece, A., et al. (2020). Neuroplex: Learning to detect complex events in sensor networks through knowledge injection. In *Proceedings of the 18th conference on embedded networked sensor systems* (pp. 489–502). New York, NY, USA: Association for Computing Machinery, URL: https://doi.org/10.1145/3384419.3431158.

Xing, T., Roig Vilamala, M., Garcia, L., Cerutti, F., Kaplan, L., Preece, A., et al. (2019). DeepCEP: Deep complex event processing using distributed multimodal information. In *2019 IEEE international conference on smart computing (SMARTCOMP)* (pp. 87–92). http://dx.doi.org/10.1109/SMARTCOMP.2019.00034.

Yankovitch, M., Kolchinsky, I., & Schuster, A. (2022). HYPERSONIC: A hybrid parallelization approach for scalable complex event processing. *SIGMOD*.