Robust Learning and Reasoning for Complex Event Forecasting

Project Acronym:	EVENFLOW
Grant Agreement Number:	101070430 (HORIZON-CL4-2021-HUMAN-01-01 – Research and Innovation Action)
Project full title:	Robust Learning and Reasoning for Complex Event Forecasting

DELIVERABLE

D4.2 – Final Version of Online Neuro-Symbolic Learning & Reasoning Techniques

Dissemination Level	PU – Public, fully open
Type of Deliverable	R – Document, report
Contractual Date of Delivery:	31 December 2025
Deliverable leader:	NCSR "Demokritos"
Status - version, date:	Final, v1.0, 2024-12-23
Keywords:	Complex Event Recognition and Forecasting, Neuro-symbolic learning and reasoning

Funded by the European Union

This document is part of a project that is funded by the European Union under the Horizon Europe agreement No 101070430. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Commission or the granting authority. The document is the property of the EVENFLOW project and should not be distributed or reproduced without prior approval. Find us at www.evenflow-project.eu.

Executive Summary

Deliverable D4.2 reports the WP4 progress during EVENFLOW's second half (M19-M39) on neuro-symbolic (NeSy) learning and reasoning techniques for Complex Event Recognition and forecasting (CER/F). The technical outcomes in WP4 in the 2nd half of the project are summarized as follows:

(i) We present NeSyA (Neuro-Symbolic Automata), a formal probabilistic framework for joint NeSy training of neural and symbolic temporal models; We present progress on learning complex event patterns from perceptual data, in order to use them for NeSy CER/F. The reported contributions are: (ii) NeurASAL (Neural Answer Set Automata Learning), a combination of NeSyA and ASAL (which was presented in Deliverable D4.1) logical structure of event patterns, while simultaneously training a neural component to map percepts to symbols that these patterns use; (iii) ∂SFA (Differentiable Symbolic Automata), a fully differentiable, NeSy event pattern learning method that can learn from perceptual sequences and alleviates the combinatorial complexity of purely symbolic methods, such as ASAL; (iv) we present approaches on using active learning principles and data programming techniques with NeSy temporal learning, in order to compensate for the lack of dense ground in event-based applications. (v) we present MiMM (Mutual-Information Markov Models), a novel method for discovering discrete latent states and transition dynamics directly from high-dimensional Markov data (e.g., image streams) without reconstructing the observations, and using these models for NeSy forecasting.

All contributions reported in this deliverable are directly related to the WP4 tasks and objectives. In particular: NeurASAL and ∂SFA are part of *T4.2, "Online Neuro-Symbolic Learning of Complex Event Forecasting Patterns"*, since they are NeSy learners for event structure discovery and they are inherently online, supporting continuous revision of such patterns on incoming data; the active learning and data programming techniques are part of *T4.4 "Reasoning-Assisted Data Programming"*, aiming to account for limited ground truth availability and assist NeSy training in scarse and indirect supervision training settings. NeSyA and MiMM are also directly related to T4.2. Although NeSyA is not concerned with learning novel patterns, it provides the formal NeSy reasoning machinery that is utilized by our structure discovery methods. MiMM learns interpretable structure in the form of latent states of a Markov Chain that captures domain dynamics. It also supports interpretable NeSy forecasting, therefore, realizing *T4.3 "Forecast Explainability"*.

In addition to the above, a substantial amount of WP4 work related to the application of WP4 techniques to the EVENFLOW's use cases is not included in this deliverable, but it is instead reported in *D3.3 "Final Use Case Evaluation"*. This work involves: (i) applying NeSyA, ASAL and ∂SFA for NeSy training and NeSy event pattern learning on the Personalized Medicine (BSC) and Industry 4.0 (DFKI) use cases; (ii) combining use case-specific perception neural networks, developed in the context of the WP4 task *T4.1 Neural Learning for Simple Event Extraction*, with learned and hand-crafted event patterns and with Wayeb, our benchmark event forecasting tool [3], for interpretable event forecasting in these domains. The purpose was to both detect in a post-hoc fashion, and also forecast ahead of time cancer progression in the BSC use case and robot deadlock incidents in the DFKI use case. Our results in D3.3 show that the NeSy techniques achieve comparable, or even superior performance in comparison to purely neural, black box techniques; (iii) applying MiMM for demonstrating a NeSy forecasting scenario on water pipe leakage incidents in the EKSO use case.

Deliverable leader:	NCSR "Demokritos"
Contributors:	Nikos Katzouris, Tatiana Boura, Nikos Manginas, Vasilis Manginas, Georgios Paliouras, Elina Syrri
Reviewers:	Abdelrahman Hekal (ICL), Satyam Dudhagara (DFKI)
Approved by:	Athanasios Poulakidas, Eleni Provopoulou (INTRA)

Document History:

Version	Date	Contributor(s)	Description
0.1	15/10/2025	Nikos Katzouris	Document skeleton & ToC creation.
0.2	15/10/2025	Nikos Katzouris	Chapter 1
0.3	10/11/2025	Nikos Katzouris	Chapters 6, 7
0.4	20/11/2025	Nikos Katzouris, Tatiana Boura	Chapter 5
0.5	21/11/2025	Nikos Katzouris, Nikos Manginas	Chapter 4
0.6	10/12/2025	Nikos Katzouris	Chapters 2, 3.
0.7	12/12/2025	Nikos Manginas & Nikos Katzouris	Chapter 8
0.8	15/12/2025	Nikos Katzouris, Georgios Paliouras, Nikos Manginas, Vasilis Manginas, Tatiana Boura, Elina Syrri	Version for internal review.
0.9	19/12/2025	Nikos Katzouris	Final version after internal review.
1.0	23/12/2025	Athanasios Poulakidas, Eleni Provopoulou, Nikos Katzouris	QA and final version for submission.

Table of Contents

List of Figures

List of Tables

Definitions, Acronyms and Abbreviations

ASAL	Answer Set Automata Learning
ASP	Answer Set Programming
BCE	Binary Cross Entropy
BK	Background Knowledge
CER/F	Complex Event Recognition & Forecasting
CE	Complex Event
CNN	Convolutional Neural Network
DFA	Deterministic Finite Automaton
d-DNN	Deterministic Decomposable Negation Normal Form
DNF	Disjunction Normal Form
DSFA	Deterministic Symbolic Finite Automaton
EQ	Equivalence Query
HMM	Hidden Markov Model
KS	Kolmogorov-Smirnov distance
LF	Labelling Function
LLM	Large Language Model
LSTM	Long Short Term Memory network
LTL	Linear Temporal Logic
LTL_f	Linear Temporal Logic on Finite Traces
MCTS	Monte Carlo Tree Search
MDL	Minimum Description Length
MI	Mutual Information
MQ	Membership Query
MLP	Multi Layer Perceptron
MM	Markov Model
NeSy	Neuro-symbolic
NeSyA	Neuro-symbolic Automata
NSFA	Non-Deterministic Symbolic Finite Automaton
OOD	Out of Distribution
RL	Reinforcement Learning
SAT	Satisfiability
SE	Simple Event
SFA	Symbolic Finite Automaton
WMC	Weighted Model Counting

1 Introduction

1.1 Project Information

EVENFLOW develops hybrid learning techniques for complex event forecasting, which combine deep learning with logic-based learning and reasoning into neuro-symbolic forecasting models. This approach combines neural representation learning techniques that construct event-driven features from streams of perception-level data with powerful symbolic learning and reasoning tools, which utilize such features to synthesize high-level, interpretable patterns for forecasting critical events.

To deal with the brittleness of neural predictors and the high volume/velocity of temporal data flows, the EVENFLOW techniques rely on novel, formal verification techniques for machine learning, in addition to a suite of scalability algorithms for training based on data synopsis, federated training and incremental model construction. The learnt forecasters will be interpretable and scalable, allowing for explainable and robust insights, delivered in a timely fashion and enabling proactive decision making.

EVENFLOW is evaluated on three use cases related to (1) oncological forecasting in healthcare, (2) safe and efficient behaviour of autonomous transportation robots in smart factories and (3) reliable life cycle assessment of critical infrastructure.

Table 1: The EVENFLOW consortium.

Number	Role	Name	Country	Short name
1	(CO)	NETCOMPANY-INTRASOFT	Belgium	INTRA
1.1	(AE)	NETCOMPANY-INTRASOFT SA	Luxemburg	INTRA-LU
2		NATIONAL CENTER FOR SCIENTIFIC RESEARCH “DEMOKRITOS”	Greece	NCSR
3		ATHINA-EREVNITIKO KENTRO KAINOTOMIAS STIS TECHNOLOGIES TIS PLHROFORIAS, TON EPIKOINONION KAI TIS GNOSIS	Greece	ARC
4		BARCELONA SUPERCOMPUTING CENTER–CENTRO NACIONAL DE SUPERCOMPUTACION	Spain	BSC
5		DEUTSCHES FORSCHUNGSZENTRUM FÜR KÜNSTLICHE INTELLIGENZ GMBH	Germany	DFKI
6		EKSO SRL	Italy	EKSO
7	(AP)	IMPERIAL COLLEGE OF SCIENCE TECHNOLOGY AND MEDICINE	United Kingdom	ICL

1.2 Document Scope

This document presents the advancements made in WP4 within the scope of EVENFLOW in the second half of the project (M18-M36). WP4 in EVENFLOW focuses on the development of neuro-symbolic (NeSy) learning and reasoning techniques in temporal domains, towards building event recognition and forecasting systems that are able to operate on sub-symbolic, perceptual input, and integrate symbolic predictive models with neural networks. This deliverable elaborates on the generic learning, reasoning and forecasting techniques developed in the second half of the project and their evaluation on challenging surrogate and EVENFLOW data.

1.3 Document Structure

This document consists of the following chapters:

- Chapter 2 presents a recap of Complex Event Recognition, Forecasting and Neuro-Symbolic AI.
- Chapter 3 presents an outline of the material presented in this deliverable.
- Chapter 4 presents NeSyA, our novel neural/probabilistic framework for scalable NeSy learning and reasoning in temporal domains.
- Chapter 5 presents an application of NeSyA on ROAD-R, a challenging real-world autonomous driving dataset .
- Chapter 6 presents our novel NeSy event pattern learning techniques, in particular, NeurASAL, a combination of ASAL and NeSyA with active learning, and ∂SFA, our differentiable event pattern learning method.
- Chapter 7 presents results on using data programming tools for compensating for the lack of ground truth data in indirect supervision NeSy training.
- Chapter 8 presents MiMM our novel technique for NeSy forecasting based on learning discrete latent states and transition dynamics directly from high-dimensional Markov data and using them for NeSy forecasting.

2 Recap of Complex Event Recognition/Forecasting and Neuro-Symbolic AI

In this section we review some basic notions from CER/F and NeSy AI that are necessary for what follows. Complex Event Recognition [41] and Forecasting [3] (CER/F) systems seek to detect, or even forecast ahead of time, occurrences of special events of interest, across a set of input data streams. The input streams consist of *simple events*, which are time-stamped pieces of information, and the output are the detected/forecast instances of the target situations, which are called *complex events* and are usually defined as spatio-temporal combinations of the simple events.

CER/F systems typically rely on a set of complex event patterns, which are declarative specifications of the interesting situations to be monitored across the input datastreams. Such situations usually involve sets of correlated events that are expected to occur in a sequential fashion. Due to the sequential nature of such complex event patterns, the computational objects that correspond to such patterns are some type of automata (finite state machines), typically, *symbolic automata*, where the transitions are guarded by predicates, rather than by mere symbols from a finite alphabet. The recognition process then amounts to matching such automata-based patterns against the simple event input, i.e. reaching an accepting state in the automaton during processing the input stream. The forecasting task amounts to deriving probabilistic estimates of future full pattern matches from partial matches that have been observed so far.

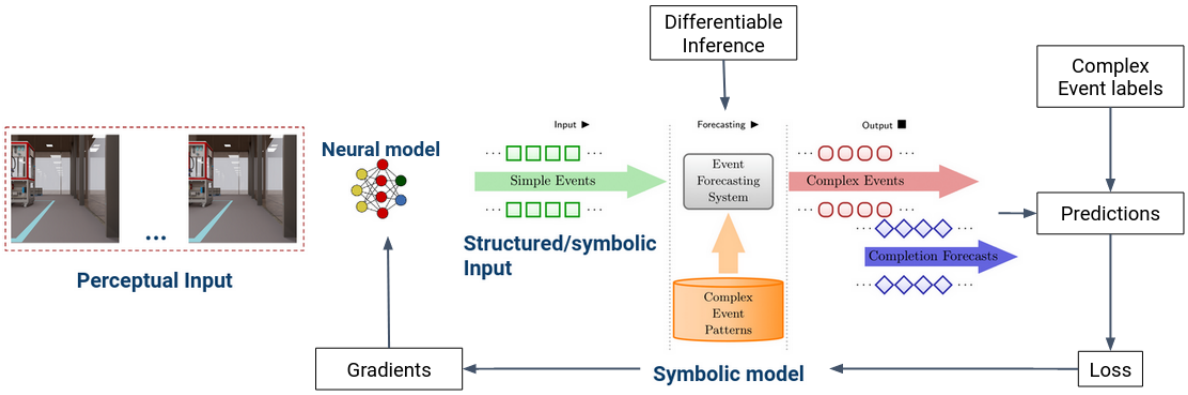


Figure 1: Neuro-symbolic Complex Event Recognition & Forecasting.

The symbolic nature of CER/F systems restricts their applicability to symbolic input. However, numerous applications deal with sub-symbolic, perceptual level input, such as sequences of images, or high-dimensional time series. A typical baseline approach in such cases is to train a neural predictor to map the sub-symbolic input to a set of symbols, corresponding to the simple events in our case, which are then passed to the symbolic model that handles the downstream CER/F task. Such approaches are of neuro-symbolic (NeSy) nature, since they combine neural and symbolic components, albeit in a loosely coupled fashion, and are thus often sub-optimal: the neural predictors are trained in isolation, ignoring the downstream task and the symbolic components ignore the stochastic, error-prone nature of the neural grounding process that produces its input symbols. They are also often infeasible, since they require large amounts of simple event-labeled data, which are usually difficult to obtain.

In contrast, tightly integrated NeSy AI approaches treat perception and symbolic reasoning as a single, coupled learning problem, allowing the CER/F model to shape how symbols are grounded and, conversely, allowing uncertainty in the grounding process to propagate through the temporal reasoning

layer. Neural components can be trained under losses derived from temporal logics and automata that define complex events, so that they learn to align their predictions with the requirements of temporal reasoning. At the same time, symbolic components can be extended with probabilistic semantics to account for the graded, error-prone outputs of neural predictors, enabling robust probabilistic forecasts over streams of noisy simple events. This tighter NeSy integration promises CER/F systems that require far fewer simple-event labels, generalize better to out-of-distribution temporal patterns, and provide explanations in terms of human-understandable patterns (rules, automata) that remain anchored to the underlying perceptual data.

3 Outline of the Contributions in this Deliverable

In this section we provide an overview of the techniques presented in this deliverable and link them to specific challenges in temporal NeSy AI.

Temporal NeSy Learning and Reasoning. A central challenge in temporal NeSy AI is to couple perception models with rich temporal knowledge for reasoning over long, noisy data streams (video, multivariate time series) and answer queries that are naturally probabilistic and counterfactual – e.g., "what is the probability that this pattern will occur within the next ten steps?". To address this problem we present NeSyA (Neuro-Symbolic Automata), a novel NeSy technique for seamlessly integrating perception neural networks with symbolic temporal knowledge, first steps towards which were presented in D4.1. NeSyA is a formal probabilistic framework for joint NeSy training of neural and symbolic temporal models. It allows for exact probabilistic inference over the neural predictions across sequences of perceptual data, supporting queries that are directly related to explainability, such as marginals and most likely explanations for observed events. Despite the fact that such queries are intractable, we get them "for free" in NeSyA, thanks to its backbone of knowledge compilation and probabilistic circuits, which allow for answering such queries in time linear in the circuit size. NeSyA has been thoroughly evaluated on both EVENFLOW and surrogate data, including a challenging, real-world autonomous driving application, which we present in detail in this deliverable.

NeSy Event Pattern Learning. In many temporal, event-based applications, the complex patterns that we care about are not fully specified in advance, evolve over time, or differ across environments. This makes necessary the development of techniques for the induction of temporal structure from data, and for grounding this structure in perceptual input when only sparse supervision is available, as it is typically the case in event-based applications. To address such issues we present two novel techniques for event pattern learning. The first contribution is a method for learning the logical structure of event patterns, while simultaneously training a neural component to map percepts to symbols that these patterns use. To compensate for the lack of dense supervision in terms of such symbols' ground truth, we use active learning principles that allow to query an oracle (typically, human annotator) for the most informative points to label. Our new technique, which we call NeurASAL, is essentially a combination of NeSyA with ASAL, a symbolic event pattern learner, which was presented in Deliverable D4.1, tied together via active learning.

Secondly, aiming to improve the scalability of the ASAL learner, we present a novel, fully differentiable approach to learning symbolic automata-based complex event patterns via gradient-based techniques. Our new method, which we call ∂SFA, learns the guards of such automata in the form of neural rules in Disjunctive Normal Form (DNF), using fuzzy conjunction and disjunction operators, and their temporal structure via a differentiable forward recursion scheme. The learned neural models can be discretized into crisp automata via weight pruning, thresholding and automata revision via ASAL. We show that our new approach is significantly more scalable than learning from scratch with ASAL from the symbolic (argmaxed) sequences predicted by a perception network, and learns fully interpretable models without compromizing their predictive performance.

Data Programming for NeSy Training with Weak Supervision. When NeSy models are trained primarily from indirect supervision (e.g., only complex event labels over entire sequences), they are

prone to *reasoning shortcuts*: they can have high accuracy by learning to map percepts to symbols in a way that, despite being wrong, happens to work adequately for the downstream task. This undermines both interpretability and out-of-distribution robustness.The most straightforward mitigation strategy is using dense latent concept supervision, which beats the whole purpose of NeSy training and is practically infeasible in high volume/velocity event-based applications.

We show how to strengthen the standard indirect-supervision setting by combining "strong" but indirect sequence-level labels (complex events) with *weak*, programmatically generated latent labels (simple events) obtained via data programming. Concretely, we construct multiple noisy labelling functions for simple events in the form of a pool diverse CNNs with varying architectures and regularization schemes, aggregate their outputs into probabilistic labels using an off-the-shelf labelling model, and inject these labels into the NeSy training loop alongside the complex-event supervision. We demonstrate that this hybrid supervision strategy significantly improves the performance of the trained models, both on latent concept prediction and on the target complex event detection task, thereby mitigating shortcut behaviour and leading to better grounded, more robust NeSy models.

NeSy Markov Chain Learning and Forecasting. Many temporal phenomena in CER/F can be seen as Markov processes over abstract states. Learning *interpetable* Markov abstractions directly from high-dimensional streams is an important open problem with clear links to forecasting: good abstractions should encapsulate the temporal information needed to accurately predict future events. Existing techniques, such as HMMs, and their extensions (including neural extensions) attempt to generatively model the data, which assume a process of reconstructing the observations given the states. However, this reconstruction process is at odds with interpretability, since it requires to encapsulate into the states a large amount of otherwise irrelevant data characteristics. In this context, we present Mutual-Information Markov Models (MiMM), a novel NeSy method for discovering discrete latent states and transition dynamics directly from high-dimensional Markov data (e.g., image streams), *without reconstructing the observations*. A neural network maps each observation to a probabilistic assignment over a finite set of latent states, and its parameters are trained to maximize the mutual information between successive latent states, yielding an abstract Markov chain that preserves the essential temporal structure of the original process, while remaining low-dimensional and interpretable. Prior knowledge about the system's dynamics (e.g., a PRISM model) can be injected by regularizing the learned transition matrix towards a symbolic prior, ensuring that the discovered states and transitions respect known behaviour. MiMM provides a way to derive such latent Markov models from raw data and then use probabilistic model checking to answer forecasting queries, such as the probability of reaching a critical or goal state within a given time horizon, thereby directly supporting the EVENFLOW's objectives on neuro-symbolic complex event forecasting and robust, explainable decision support.

4 NeSyA: Neurosymbolic Automata

4.1 Introduction

Sequence classification/tagging is a ubiquitous task in AI. Purely neural models, including LSTMs [49] and Transformers [96], have shown exemplary performance in processing sequences with complex high-dimensional inputs. Nonetheless, various shortcomings still exist in terms of generalization, data-efficiency, explainability and compliance to domain or commonsense knowledge. NeSy AI [37] aims to integrate neural learning and symbolic reasoning, possibly aiding in the aforementioned limitations of purely neural systems. Recently, various NeSy systems have been developed for sequential/temporal problems [99, 31, 89], with large differences among them, in terms of semantics, inference procedures, and scalability of the proposed hybrid models.

In this work, we identify symbolic automata as an attractive low-level representation of complex temporal properties. These differ from classical automata as they support symbolic transitions between states (defined in propositional logic), thus combining temporal reasoning (through the automaton) and atemporal reasoning (through the logical transitions). We show that symbolic automata can be efficiently integrated with neural-based perception and thereby extended to subsymbolic domains. Figure 2 illustrates the core NeSyA architecture in a running example, which is used throughout the paper.

The key characteristics of NeSyA that can be used for comparison to existing NeSy systems, are: **[C1]** its focus on temporal domains, **[C2]** its probabilistic semantics, **[C3]** its capacity to integrate static logical reasoning into temporal patterns, **[C4]** its efficient and exact inference scheme based on matrices and knowledge compilation [29].

The closest system to our work is FuzzyA [89] which attempts to address the NeSy integration of LTL_f with neural networks. That system differs from NeSyA primarily in terms of **[C2]**, as it is based on fuzzy logic and specifically on Logic Tensor Networks [15]. As we shall show in this paper, probabilistic semantics can provide significant benefits, in terms of predictive accuracy, over fuzzy logic, as used in FuzzyA.

On the other hand, NeSyA differs from approaches like the Semantics Loss (SL) [103] and DeepProbLog [67], in terms of **[C1]**, as they are not tailored to temporal reasoning. In this paper, we shall show that this makes them scale considerably worse than NeSyA when faced with problems with a temporal component.

The more recent DeepStochLog system [99] is based on unification grammars and therefore differs from NeSyA in terms of both **[C1]** but mostly **[C4]**. Our experiments show that this difference makes DeepStochLog orders of magnitude slower than NeSyA.

Further, systems based on neural networks and classical automata, such as [87, 88] differ from NeSyA in terms of **[C3]**, since classical automata lack symbolic transitions and support for atemporal reasoning.

Lastly, [31] is based on very expressive models in mixed discrete and continuous domains. It is based on approximate inference, thus differing from NeSyA in terms of **[C4]**.

Our contributions are as follows:

- We introduce NeSyA a probabilistic NeSy system for sequence classification and tagging, which combines automata, logic and neural networks.

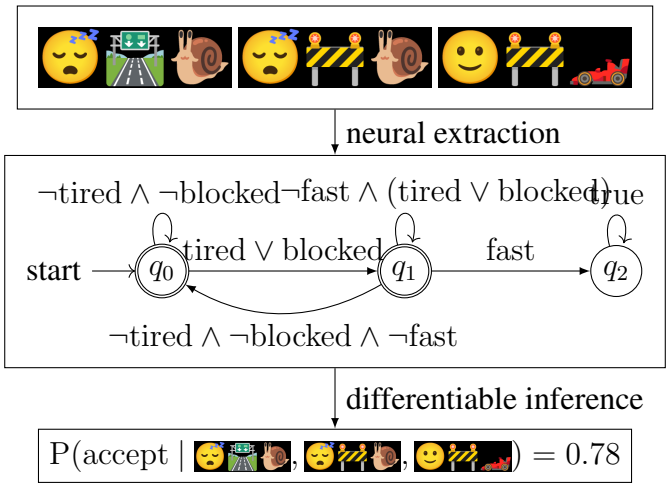


Figure 2: Symbolic automata (middle) are used to reason over sequences of subsymbolic inputs (top) from which information is extracted with the aid of a neural network, performing multilabel classification. For instance, for the image , the correct symbol grounding is $\{\text{tired}, \neg\text{blocked}, \neg\text{fast}\}$. The symbolic automaton shown captures the following logic: If the driver is tired or the road is blocked, then in the next timestep they should not be going fast. NeSyA computes the probability of the SFA accepting the input sequence (bottom), which is then used for learning.

- We introduce an efficient algorithm for inference in NeSyA, utilizing matrix-based automata inference and knowledge compilation based approaches for logical inference [29].
- On a synthetic sequence classification domain, we show that NeSyA leads to large performance benefits over FuzzyA [89] and scales orders of magnitude better than DeepStochLog [99], which is also based on a probabilistic semantics.
- On a real-world event recognition domain we show that NeSyA can lead to a more accurate event recognition system, compared to purely neural approaches.

4.2 Background

4.2.1 Propositional Logic and Traces

We shall use lowercase to denote propositional variables, e.g. blocked. A propositional formula ϕ over a set of variables V is defined as:

$$\phi ::= V \mid \neg\phi_1 \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2.$$

These connectives are sufficient to then define $\rightarrow$, etc. An interpretation $\omega \subseteq V$ assigns a truth value to each variable. We use subsets to denote interpretations. For instance for $V = \{\text{tired}, \text{blocked}, \text{fast}\}$ the interpretation $\omega = \{\text{tired}, \text{blocked}\}$ is shorthand for $\{\text{tired}, \text{blocked}, \neg\text{fast}\}$. If an interpretation ω satisfies a formula ϕ we write $\omega \models \phi$ and ω is called a model of ϕ.

The semantics of propositional logic are given in terms of interpretations. Traces generalize interpretations for temporal domains. A trace over variables V, $\pi = (\omega_1, \omega_2, \ldots, \omega_n)$ is a sequence of interpretations, with $\omega_i \subseteq V$. We use π_t to denote the interpretation ω_t at timestep t.

4.2.2 SFA: Symbolic Automata

A symbolic automaton (SFA) is defined as:

$$\mathcal{A} = (V, Q, q_0, \delta, F),$$

where V is a set of variables, Q a set of states, $q_0 \in Q$ the initial state, $\delta : Q \times Q \to B(V)$ is the transition function and $F \subset Q$ is the set of accepting states. $B(V)$ is used to denote the set of all valid formulae in propositional logic over variables V. The difference between an SFA and a classical automaton is that δ is given in a factored form, e.g. $\delta(q_0, q_1) = \text{tired} \vee \text{blocked}$ in Figure 2. One can always convert the SFA to a classical automaton by replacing each transition $\delta(q, q')$ with multiple ones representing all models $\omega \models \delta(q, q')$. This approach does not scale to complex formulae and large sets of variables, as the number of resulting transitions can be exponential in V. This factored transition function is also common in the Markov Decision Process literature [43] with the goal being to exploit the symbolic nature of the transitions without "propositionalizing".

A symbolic automaton reads traces, i.e. sequences of interpretations $(\omega_1, \ldots, \omega_n)$ $(\omega_i \subseteq V)$ over the variables V. We shall consider deterministic SFAs, in which:

$$\forall q \in Q, \omega \in 2^V : \exists!\, q' : \omega \models \delta(q, q').$$

That is, for any state $q \in Q$ and any interpretation $\omega \in 2^V$ exactly one transition outgoing from state q will be satisfied by ω. For the SFA in Figure 2 consider the transitions outgoing from state q_0. For any interpretation, either $(\neg\text{tired} \wedge \neg\text{blocked})$ or $(\text{tired} \vee \text{blocked})$ will be true. If the SFA ends up in an accepting state after reading the trace π we write $\pi \models \mathcal{A}$.

4.2.3 Probabilistic Logical Inference

Probabilistic logical inference is the task of computing the probability of a logical formula under uncertain input. For a propositional formula ϕ over variables V, let p denote a probability vector over the same variables. Each element $\mathrm{p}[i]$ therefore denotes the probability of the i^{th} symbol in V being true. The probability of the formula given p is then defined as:

$$\begin{aligned} \mathrm{P}(\phi \mid \mathrm{p}) &= \sum_{\omega \models \phi} \mathrm{P}(\omega \mid \mathrm{p}), \\ \text{with } \mathrm{P}(\omega \mid \mathrm{p}) &= \prod_{i \in \omega} \mathrm{p}[i] \prod_{i \notin \omega} 1 - \mathrm{p}[i]. \end{aligned} \tag{1}$$

This task is reducible to weighted model counting (WMC), one of the most widely-used approaches to probabilistic logical inference [22]. As computing WMC involves summing over all models of a propositional formula, it lies in the #P complexity class of counting problems [91]. Knowledge Compilation (KC) [29] is a common approach to solve WMC problems. It involves transforming a logical formula to a tractable representation, on which WMC queries can be cast in linear time. Importantly, once a formula has been compiled to a tractable representation, WMC cannot only be computed in linear time but also differentiably. The computational complexity of the problem is effectively shifted to an initial compilation phase but can be amortized, since multiple queries can be cast on the compiled representation. Consider for example the formula $\phi = \neg\text{fast} \wedge (\text{tired} \vee \text{blocked})$, i.e. the transition $q_1 \to q_1$ in Figure 2. Its compiled form as a d-DNNF circuit [27], one of the

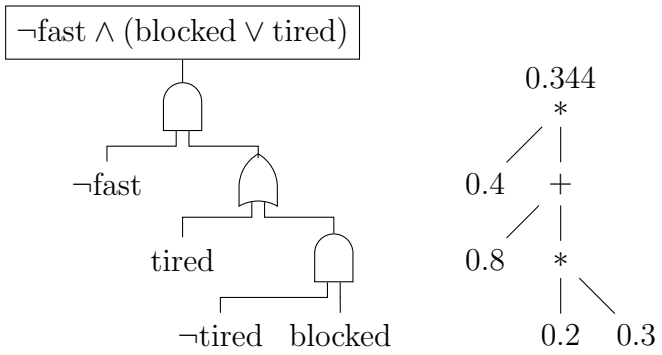


Figure 3: A d-DNNF circuit for the formula $\phi = \neg\text{fast} \wedge (\text{blocked} \vee \text{tired})$ (left) and an arithmetic circuit produced from the d-DNNF circuit (right). The computation of WMC is shown for the vector $\text{p} = [0.8, 0.3, 0.6]$ for the symbols $\{\text{tired}, \text{blocked}, \text{fast}\}$ respectively.

tractable KC representations, and the computation of WMC can be seen in Figure 3. The logical circuit in Figure 3 (left) is converted to an arithmetic circuit Figure 3 (right) by replacing AND gates with multiplication and OR gates with addition. The weighted model count for the probability vector in Figure 3 can be verified to be correct by:

$$\begin{aligned}
&\text{P}(\neg\text{fast} \wedge (\text{tired} \vee \text{blocked}) \mid \text{p} = [0.8, 0.3, 0.6]) \\
&= \text{P}(\{\text{tired}\}|\text{p}) + \text{P}(\{\text{blocked}\}|\text{p}) \\
&\quad + \text{P}(\{\text{tired}, \text{blocked}\}|\text{p}) \\
&= 0.8 \times 0.7 \times 0.4 + 0.2 \times 0.3 \times 0.4 + 0.8 \times 0.3 \times 0.4 \\
&= 0.344.
\end{aligned}$$

Recall that interpretations are given in shorthand, e.g. $\{\text{tired}\}$ is shorthand for $\{\text{tired}, \neg\text{blocked}, \neg\text{fast}\}$.

4.3 Method

4.3.1 Formulation and Inference

We introduce NeSyA as a NeSy extension of the SFAs introduced earlier. Rather than assuming a trace of propositional interpretations $\pi = (\omega_1, \omega_2, \ldots, \omega_n)$ we assume a sequence of subsymbolic observations $o = (o_1, o_2, \ldots, o_n)$ with $o_i \in \mathbb{R}^m$. NeSyA is defined as a tuple $(\mathcal{A}, f_\theta)$, with $\mathcal{A}$ an SFA over variables V and $f_\theta : \mathbb{R}^m \to [0, 1]^{|V|}$ a neural network, which computes a probability vector $f_\theta(o_t)$ over the variables V from the observation o_t. Therefore $f_\theta(o_t)[i]$ denotes the probability of the i^{th} variable in V being true given the observation o_t. The neural network is used to bridge between the discrete representation of the SFA and the continuous representation of the observations.

The resulting model is depicted in graphical model notation in Figure 4, where q_t denotes a discrete random variable over the states of the SFA and o_t the input observation at time t. Following [70], we define

$$\alpha_t(q) = P(q \mid o_1, ..., o_t)$$

as the probability of being in state q at timestep t after seeing the observations $(o_1, ..., o_t)$. α_t can be computed recursively (using dynamic programming) as follows:

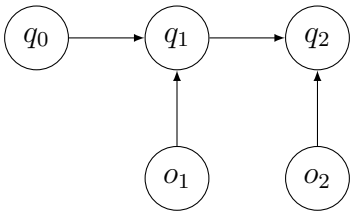


Figure 4: Graphical model for NeSyA. Following the approach used in [70], it resembles a Hidden Markov Model with the arrows between states and observations reversed. The random variables q_t take values from Q, the state space of the SFA, and the random variables o_t take values from high-dimensional continuous spaces.

$$
\begin{aligned}
&\alpha_0(q_0) = 1; \;\; \alpha_0(q) = 0 \;\; \forall q \neq q_0 \\
&\alpha_{t+1}(q) = \sum_{q' \in Q} \mathrm{P}(q \mid q', o_{t+1}) \; \alpha_t(q'), \\
&\text{where } \mathrm{P}(q \mid q', o_{t+1}) = \mathrm{P}(\delta(q', q) \mid f_\theta(o_{t+1})) \\
&\qquad\qquad\qquad = \sum_{\omega \models \delta(q', q)} \mathrm{P}(\omega \mid f_\theta(o_{t+1})).
\end{aligned}
\tag{2}
$$

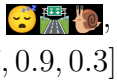
Thus, to update the probability of being in each state, one must first compute the probabilities of the logical formulae in the SFA's transitions given the outputs of the neural network for the current observation. Instead of naively summing over all models of each formula, we use KC to make the computation efficient. The state update, which is similar to the one in Hidden Markov Models, can be captured via matrix operations and is therefore amenable to parallelization and execution on GPUs. It is well-known that α_t can be represented with a vector of size $|Q|$, whose elements are the probabilities of being in each state at timestep t. In what follows we adopt this notation.

Running example computation:

Consider the SFA in Figure 2. Let the first observation be $o_1 =$  and let $f_\theta(o_1) = [0.8, 0.3, 0.6]$ the output of the neural network. We define the transition matrix $T(o_i)$ where $T(o_i)[q', q] = \mathrm{P}(\delta(q', q) \mid f_\theta(o_i))$. We thus have:

$$
T(\text{  }) = \begin{bmatrix} 0.14 & 0.86 & 0 \\ 0.056 & 0.344 & 0.6 \\ 0 & 0 & 1 \end{bmatrix}.
$$

The calculation of the entry $T(\text{  })[q_1, q_1]$ was shown in Section 4.2.3. Similarly, the computation of other entries is performed by propagating an arithmetic circuit for each transition, given the neural network predictions for the current observation. Observe that the sum of each row in the transition matrix, i.e. the total mass out of each state is 1. This is a direct consequence of the deterministic property of the SFA, where exactly one outgoing transition from each state will be true for any possible interpretation. It also ensures that $\sum_{q \in Q} \alpha_t[q] = 1$ for all t.

We start with α_0, where $\alpha_0[q_0] = 1$ and $\alpha_0[q] = 0$ for all $q \in Q, \; q \neq q_0$, We then recursively compute α_t for each subsequent timestep. Let $o = (o_1 =$  , $o_2 =$  $)$. Consider the neural network predictions for o_1 as above and let $f_\theta(o_2) = [0.7, 0.9, 0.3]$. We calculate:

$$\begin{aligned}
\alpha_2 &= \alpha_1 \times T(o_2) \\
&= \alpha_0 \times T(o_1) \times T(o_2) \\
&= \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0.14 & 0.86 & 0 \\ 0.056 & 0.344 & 0.6 \\ 0 & 0 & 1 \end{bmatrix} \times T(o_2) \\
&= \begin{bmatrix} 0.14 & 0.86 & 0 \end{bmatrix} \times \begin{bmatrix} 0.03 & 0.97 & 0 \\ 0.021 & 0.679 & 0.3 \\ 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} 0.023 & 0.7197 & 0.258 \end{bmatrix}.
\end{aligned}$$

Depending on the task, the α-recursion can be used in various ways. For sequence classification, one only cares about the state probabilities in the final timestep and would aggregate over accepting states to get the probability of accepting the sequence. Concretely:

$$\mathrm{P}_{\text{accept}}(o) = \sum_{f \in F} \alpha_n(f).$$

where n is the length of the sequence. In this case $\mathrm{P}_{\text{accept}}(o) = 0.023 + 0.7179 = 0.742$. In other applications, such as sequence tagging, one is interested about the α values in every timestep.

4.3.2 Learning

For ease of exposition we shall consider the sequence classification task, in which NeSyA is given a subsymbolic sequence o and computes $\mathrm{P}_{\text{accept}}(o)$. The computation of $\mathrm{P}_{\text{accept}}(o)$ is differentiable with respect to the neural network outputs as the only operations necessary to compute the acceptance probability are: (a) the computation of WMC which, as shown in Section 4.2.3, reduces to propagating an arithmetic circuit comprised of addition, multiplication and subtraction, (b) the α-recursion which is implemented via standard matrix operations, and (c) a summation over the final α values.

Therefore, given a dataset of pairs (o, L), where $L \in \{0, 1\}$ is a binary label for the sequence o, one can train the neural component of NeSyA by minimizing a standard supervised learning loss, e.g.

$$\mathcal{L}(o, L) = \mathrm{BCE}(\mathrm{P}_{\text{accept}}(o), L),$$

where BCE stands for the standard binary cross entropy. This amounts to training the neural network via weak supervision, where no direct labels are given for the symbol grounding of each observation, but rather for the sequence as a whole. This weak-supervision learning setup is common and can be found in [67, 105, 99, 89]. More concretely, observe that we don't require examples of the form ( , $\{\text{tired}, \neg\text{blocked}, \neg\text{fast}\}$), as we would in a fully supervised multilabel problem, but rather of the form (( ,  , ), 0). Such high-level labels are in general much fewer in number and more easily attained. Given the differentiability of the model, explained at the start of this subsection, the neural component f_θ is trained via standard gradient descent on the distant labels.

4.3.3 Semantics and Discussion

Consider a sequence of probability vectors $(\mathrm{p}_1, \mathrm{p}_2, \ldots, \mathrm{p}_n)$ with each vector p_t assigning a probability to each proposition $v \in V$ at timestep t. In NeSyA $\mathrm{p}_t = f_\theta(o_t)$, i.e. these probability vectors

are computed via a neural network conditioned on the observation at each timestep, but we ignore the neural component at this stage of the analysis. Given $(\mathrm{p}_1, \ldots, \mathrm{p}_\mathrm{n})$, the probability of trace π, a sequence of interpretations, is then:

$$\mathrm{P}(\pi \mid (\mathrm{p}_1, \mathrm{p}_2, \ldots, \mathrm{p}_\mathrm{n})) = \prod_{t=1}^{n} \mathrm{P}(\pi_t \mid \mathrm{p}_t)$$

To elaborate, the probability of a sequence of interpretations is the product of the probability of each interpretation π_t given p_t.

Theorem 4.1 (α-semantics)**.** *It holds that:*

$$\alpha_t[q] = \sum_{\pi \in \mathrm{traces}(q,t)} \mathrm{P}(\pi \mid (\mathrm{p}_1, \mathrm{p}_2, \ldots, \mathrm{p}_\mathrm{t})).$$

where $\mathrm{traces}(q, t)$ *is the set of all traces which cause the* SFA *to end up in state* q *starting from* q_0 *in* t *timesteps. The probability of being in state* q *at timestep* t *is then the sum of all such traces (sequences of interpretations) weighted by the probability of each trace given* $(\mathrm{p}_1, \ldots, \mathrm{p}_\mathrm{t})$.

Proof. The result directly follows from the graphical model but we provide a proof from first principles. We focus on the meaning of $\alpha_t(q)$, from which we can also draw further conclusions. Consider an SFA over propositions V. Let $\mathrm{traces}(q, t)$ denote all traces over V which starting from state q_0 cause the SFA to end up in state q after t timesteps. Further, consider a sequence of probability vectors $(\mathrm{p}_1, \mathrm{p}_2, \ldots, \mathrm{p}_\mathrm{n})$ with each vector p_i assigning a probability to each proposition $v \in V$ at timestep i. Refer to Section 4.2.3 for an example of p_i. The probability of a trace π is then:

$$\mathrm{P}(\pi) = \prod_{t=1}^{n} \mathrm{P}(\pi_t \mid \mathrm{p}_t),$$

Theorem 4.1 states that

$$\alpha_t(q) = \sum_{\pi \in \mathrm{traces}(q,t)} \mathrm{P}(\pi).$$

We shall prove Theorem 4.1 by induction. For $t = 1$ we have:

$$\alpha_1(q) = \sum_{\omega \models \delta(q_0,q)} \mathrm{P}(\omega|\mathrm{p}_1) = \sum_{\pi \in \mathrm{traces}(q,1)} \mathrm{P}(\pi),$$

recalling that $\alpha_0(q) = 1$ if $q = q_0$ and 0 otherwise and from Equation 1 and 2. Assuming the hypothesis holds for t, we can prove it for $t + 1$, as follows:

$$\begin{aligned}
\alpha_{t+1}(q) &= \sum_{q' \in Q} \mathrm{P}(q \mid q', \mathrm{p}_{t+1})\ \alpha_t(q') \\
&= \sum_{q' \in Q} \sum_{\omega \models \delta(q',q)} \mathrm{P}(\omega \mid \mathrm{p}_{t+1}) \sum_{\pi \in \mathrm{traces}(q',t)} \mathrm{P}(\pi) \\
&= \sum_{\pi \in \mathrm{traces}(q,t+1)} \mathrm{P}(\pi).
\end{aligned}$$

Sequence Length	Method	Pattern					
		1		2		3	
		Accuracy	Time	Accuracy	Time	Accuracy	Time
10	NeSyA	**1.0**	**0.8**	**0.98** $\pm$ 0.03	**1.1**	**1.0**	**2.3**
	FuzzyA	0.91 $\pm$ 0.06	10.8	0.70 $\pm$ 0.13	22.5	0.78 $\pm$ 0.04	29.9
20	NeSyA	**1.0**	**1.2**	**0.99** $\pm$ 0.01	**1.7**	**0.99** $\pm$ 0.01	**3**
	FuzzyA	0.77 $\pm$ 0.22	21.4	0.69 $\pm$ 0.14	43.7	0.7 $\pm$ 0.1	57.7
30	NeSyA	**1.0**	**1.7**	**0.94** $\pm$ 0.11	**2.3**	**0.97** $\pm$ 0.03	**3.8**
	FuzzyA	0.98 $\pm$ 0.01	31.7	0.55 $\pm$ 0.1	55.9	0.5	86.6

Table 2: Accuracy results on a test set, and timings (in minutes) for NeSyA against FuzzyA averaged across 5 runs, as well as standard deviation (when over 0.01). Both systems are trained with a learning rate of 0.001 following [89]

.

The last step follows from:

$$\text{traces}(q, t+1) = \bigcup_{q' \in Q} \{\pi.\omega \mid \omega \models \delta(q', q), \pi \in \text{traces}(q', t)\}$$

with the concatenation of an interpretation ω with a trace π, i.e $\pi.\omega = (\pi_1, \ldots, \pi_t, \omega)$. To elaborate, the set of traces which end in state q in $t+1$ timesteps is the union over all traces which ended in state q' in t timesteps concatenated with each interpretation ω causing the SFA to transition from q' to q. □

An immediate consequence of Theorem 4.1 is that:

$$\sum_{\pi \models A} \mathrm{P}(\pi) = \sum_{f \in F} \alpha_T(f)$$

for an SFA A.

A direct consequence is that for an SFA $\mathcal{A}$ and the sequence $(\mathrm{p}_1, \ldots, \mathrm{p}_\mathrm{n})$ of symbol probabilities:

$$\sum_{f \in F} \alpha_n[f] = \sum_{\pi \models \mathcal{A}} \mathrm{P}(\pi \mid (\mathrm{p}_1, \mathrm{p}_2, \ldots, \mathrm{p}_\mathrm{n})). \tag{3}$$

Once the logical transitions of the SFA have been compiled to a tractable form, see Section 4.2.3, this computation is polynomial in the number of nodes of the compiled circuits and the number of states of the SFA. This makes the compiled SFA, a tractable device for performing computations over uncertain symbolic sequences.

This result is important for extending NeSy systems to the temporal domain. Consider the symbolic component of a NeSy system, e.g. DeepProbLog. It eventually reduces to a propositional formula ϕ and relies on the computation of $\sum_{\omega \models \phi} \mathrm{P}(\omega \mid \mathrm{p})$, where p is usually the output of a neural network conditioned on an observation. For temporal NeSy systems if the symbolic component can be captured by an SFA $\mathcal{A}$, then $\sum_{\pi \models \mathcal{A}} \mathrm{P}(\pi | \mathrm{p}_1, \mathrm{p}_2, \ldots, \mathrm{p}_\mathrm{t})$ is computable as shown in Equation 3. To further motivate the potential efficacy of SFAs as promising low-level representations in the context of NeSy, we note that they are known to capture STRIPS domains as well as temporal logics [30] and are

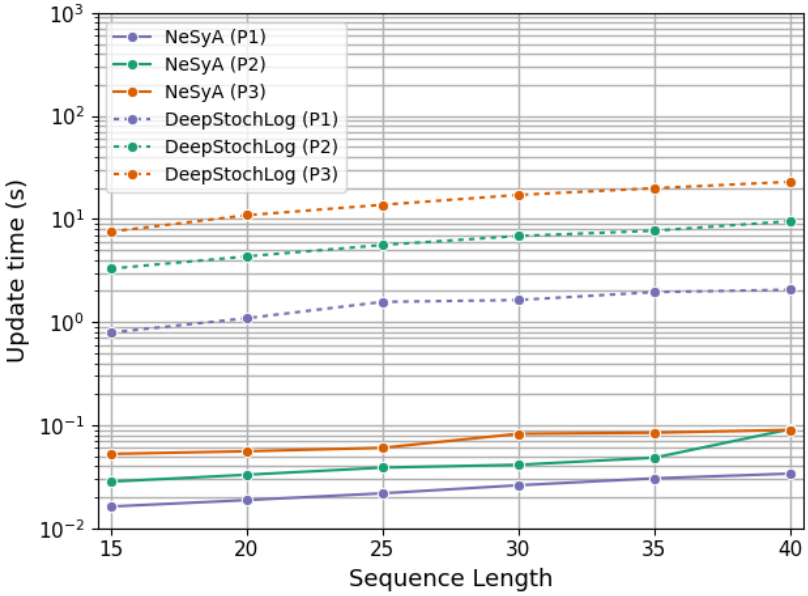


Figure 5: Scalability results for NeSyA (solid) and DeepStochLog (dashed) for each of the three patterns tested. The y-axis represents the update time for a single batch of 16 sequences in logarithmic scale and the x-axis the sequence length. The systems were benchmarked for three different patterns of varying complexity both in terms of symbols, as well as states of the automaton.

thus quite expressive. Hence, it is possible, that SFAs can serve as an efficient compilation target for temporal NeSy systems, much like d-DNNF and similar representations have done for atemporal NeSy systems.

4.4 Results

In this section we provide empirical results for the performance of NeSyA and its comparison to other NeSy systems, as well as to purely neural ones. We aim to answer the following questions:

[Q1] Scalability: How does NeSyA compare to DeepStochLog and FuzzyA in terms of runtime on the same NeSy learning task?

[Q2] Accuracy: How does NeSyA compare to FuzzyA in terms of accuracy on the same NeSy learning task? [1]

[Q3] Generalization: How does NeSyA compare to purely neural solutions in terms of generalization?

All experiments were run on a machine with an AMD Ryzen Threadripper PRO 3955WX 16-Core processor, 128GB of RAM, and 2 NVIDIA RTX A6000 with 50GB of VRAM of which only one was utilized.

All experiments were implemented in Pytorch and Python 3.11. For the experiment in Section 4.4.1 we use the implementation of FuzzyA provided by the authors[2] with minimal changes. Both NeSyA and FuzzyA were trained for a fixed amount of 100 epochs. For the second experiment (Section 4.4.2) we use an LSTM with a single layer and a 128 dimensional hidden state. The Transformer architecture has 3 attention heads per layer, 4 layers and an hidden state dimensionality of 129 (same with the input dimensions). Both architectures utilize the same CNN to extract visual embeddings of the bounding

[1] The accuracy of DeepStochLog is not compared against that of NeSyA, as they generate the same results on the same input.

[2] https://github.com/whitemech/grounding_LTLf_in_image_sequences

boxes.

4.4.1 Synthetic Driving

We first benchmarked NeSy systems on a synthetic task, which allowed us to control the complexity. In particular, we used the domain introduced as a running example, in which a sequence of images must be classified according to a temporal pattern. Each image represents a set of binary symbols. In the example from Figure 2 the symbols were $\{\text{tired}, \text{blocked}, \text{fast}\}$, however we test for sets of up to five symbols. Their truth value is represented via two emojis, one corresponding to the value true and one false. Random Gaussian noise is added to each image to make the mapping between an image and its symbolic interpretation less trivial. We generate three patterns (different SFAs) with 3, 4 and 6 SFA states and 3, 4 and 5 symbols respectively. For each pattern, we generated 100 random trajectories which satisfy the pattern (positive) and 100 negative ones. We use the same setup for generating a training and a testing set.

The neural component of all systems is a CNN. The learning task is as described in Section 4.3.2, where the neural component must perform symbol grounding without direct supervision. Instead supervision is provided at the sequence level and the neural component is trained weakly. We benchmarked against DEEPSTOCHLOG [99] and DEEPPROBLOG [67] in terms of scalability and with FUZZYA [89] both in terms of scalability and accuracy. Figure 5 shows the comparison of NeSyA against the DEEPSTOCHLOG system on temporal patterns of ranging complexity in the synthetic driving benchmark. The DEEPPROBLOG system lagged behind the other two considerably and therefore is omitted from the results for brevity. Accuracy results are also omitted here, since all three systems are equivalent in their computation and learning setup and therefore perform identically in terms of accuracy. In terms of computational performance, NESYA does significantly better than DEEPSTOCHLOG, being on average two orders of magnitude faster. As an indication, for the most complex task and a sequence length of 30, NESYA takes 0.08 seconds for a single batch update and DEEPSTOCHLOG takes about 30 seconds, rendering the latter system of limited practical use. As an indication of the difference against DEEPPROBLOG, for the simplest pattern and a single sequence of length 15, the update time for DEEPPROBLOG is 140 seconds compared to 0.02 seconds for NESYA.

Next, in Table 2 we show accuracy and scalability results of NESYA against the FUZZYA system. NESYA, which interfaces between the SFA and the neural representations using probability, seems to offer a much more robust NeSy solution. FUZZYA delivers significantly lower accuracy compared to NESYA, especially as sequence length grows. Further FUZZYA lags significantly in terms of scalability.

The results on our synthetic benchmark allow us to affirmatively answer **[Q1]** and **[Q2]**. NESYA seems to scale better than both DEEPSTOCHLOG and FUZZYA for even the simplest patterns considered here, often by very large margins. Further, our system is more accurate than FUZZYA, with the difference in performance becoming very large for large sequence lengths and complex patterns.

4.4.2 Event Recognition

In our second experiment we compared NESYA against pure neural solutions on an event recognition task from the CAVIAR benchmark dataset[3]. The task was to recognize events performed by pairs of people in raw video data. We focused on two of the events present in CAVIAR, namely moving and meeting, which appear more frequently in the data, and a third no_event class. We present three

[3] https://homepages.inf.ed.ac.uk/rbf/CAVIARDATA1/

Figure 6: Sample of the CAVIAR data. Models are given the two bounding boxes per timestep instead of the complete image, in order to make the task simpler for the neural component. Along with the pair of bounding boxes, a close(p1, p2) feature is provided, which captures whether the two people are close to each other. The CNN for NeSyA must ground one bounding box to the symbols walking(p1), running(p1), active(p1), inactive(p1) and correspondingly to p2 for the second bounding box. The correct grounding for this image is active(p1) and walking(p2). These are the low-level activities performed by each person. The high-level activities performed by the pair are annotated for each image in one of the three classes no_event, meeting, moving. For the image shown here the annotation is moving.

methods; NeSyA a CNN-LSTM and a CNN-Transformer. The data consists of 8 training and 3 testing sequences. The label distribution for the training data is 1183 frames of no_event, 851 frames of moving and 641 frames of meeting. For the test set, these are 692, 256 and 894 respectively. The mean sequence length is 411 with a minimum length of 82 and a maximum length of 1054. We use the macro F1 score for evaluation of all models.

The CAVIAR data is annotated at a frame level with bounding boxes of the people in the scene, as well as with low-level activities they perform, such as walking and running. From the raw data, we extract sequences of two bounding boxes per timestep, as well as a Boolean feature of whether the distance between the bounding boxes is smaller than some threshold. Refer to Figure 6 for an overview. The symbolic component of NeSyA in this case is a three-state automaton, capturing a variant of the Event Calculus [59] programs for CAVIAR found in [13] and can be seen in Figure 7. We use the SFA to label the sequence with the current high-level event in each frame given the ground truth labels

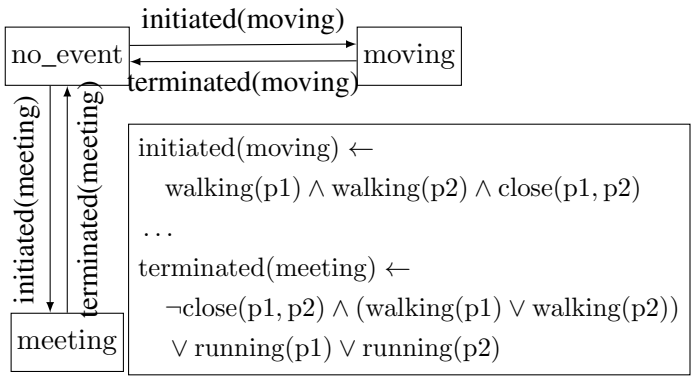


Figure 7: The SFA used for the CAVIAR experiments. It defines transitions between two classes meeting and moving and a third no-event class. Only a subset of the transition logic is shown for brevity. In the case that no outgoing transition from a state is satisfied the SFA loops in its current state.

		Learning rate					
		10^{-3}		10^{-4}		10^{-5}	
	#Params	Train	Test	Train	Test	Train	Test
NeSyA	$26K$	$\mathbf{0.81} \pm 0.13$	$\mathbf{0.60} \pm 0.18$	0.87 ± 0.03	$\underline{\mathbf{0.85} \pm 0.20}$	0.86 ± 0.10	$\mathbf{0.81} \pm 0.18$
CNN-LSTM	$400K$	0.70 ± 0.23	$\underline{0.56 \pm 0.21}$	0.84 ± 0.08	0.35 ± 0.19	0.17 ± 0.05	0.15 ± 0.06
CNN-Transformer	$2.5M$	0.72 ± 0.26	0.40 ± 0.10	$\mathbf{1.00} \pm 0.00$	0.68 ± 0.16	$\mathbf{0.97} \pm 0.02$	$\underline{0.78 \pm 0.14}$

Table 3: Results for the CAVIAR dataset. Performance is averaged over 10 random seeds. The metric reported is macro F1 score. We present results for 3 different learning rates as the dataset is small and constructing a validation split to tune for the learning rate would further reduce the size of the training data. For all systems training is stopped by monitoring the training loss with a patience of 10 epochs. Best test results for each method are underlined.

for the low level activities. The true high-level events are also given in the CAVIAR data, but the labels are noisy, i.e. there is some disagreement between the start and end points of the high-level events generated by the logic and those provided by human annotators. Using the labels generated by the SFA, we assume perfect knowledge, i.e. that the symbolic component of NeSyA can perfectly retrieve the high-level events, given the low-level activities. Learning with a label noise is beyond the scope of this work.

The task in CAVIAR is therefore to tag a sequence of pairs of bounding boxes, along with a Boolean distance feature, with the high-level event being performed in each timestep. For NeSyA each bounding box is processed by a CNN which gives a probability for each of the low-level activities (walking, running, active and inactive). Combining this with the feature $\mathrm{close}(\mathrm{p1}, \mathrm{p2})$, these are then passed through the SFA which outputs the probability of each high-level event per timestep. As a baseline, we drop the final linear projection of the CNN used for NeSyA. The resulting CNN computes a 64-dimensional embedding for each bounding box. We concatenate the embeddings of the bounding boxes along with the distance feature finally producing a 129-dimensional embedding per frame. This embedding is then given to either an LSTM or a Transformer, whose hidden state is projected to the three high-level event classes. All systems are trained by computing a cross entropy loss on the high-level event predictions in every timestep of each sequence. The supervision is therefore in the frame level contrary to the experiment in Section 4.4.1 where supervision was on the sequence level. For NeSyA the loss in the CAVIAR dataset is:

$$\mathcal{L}(o, L) = \sum_{t \in \{1,\dots,n\}} \mathrm{CE}(\alpha_t, L_t),$$

where n denotes the sequence length, α_t the probabilities of being in each state of SFA at timestep t (and therefore of emitting each label) and L_t denotes the true high-level event label for that timestep, e.g. $\mathrm{meeting}$. For the pure neural solutions α_t is replaced with the output of a linear projection on the LSTM/Transformer hidden state at timestep t. The performance of the three systems can be seen in Table 3.

The results in Table 3 allow us to also answer **[Q3]** affirmatively. The inclusion of knowledge about the structure of the high-level events based on the low-level activities aids in generalization and the discrepancy between train and test performance is generally small for NeSyA and larger for purely neural solutions in this low data regime. The Transformer baseline is able to compete with NeSyA, albeit with an order of magnitude more parameters. It is interesting that 2.5 million parameters (the

difference between NeSyA and the CNN-Transformer) are necessary to find a solution that delivers comparable performance with the three state SFA and simple transition logic used by NeSyA as background knowledge. The results in CAVIAR are to be taken with a grain of salt as standard deviations are high due to the small size of the data which causes outlier runs for all methods.

4.5 Future Work

Recently, [104] used the DeepProbLog system to integrate logical constraints in the training of Reinforcement Learning (RL) agents in subsymbolic environments. We believe NeSyA can aid in this direction, by allowing for the specification of more complicated temporal constraints, which require memory, i.e. some notion of state to be remembered from the execution of the environment so far, while being more scalable. A large class of systems is based on constraints for RL agents [6, 52] often using LTL. This seems a promising avenue for NeSyA which can extend such methods to subsymbolic RL domains. Further, NeSyA can be used to extend systems where automata are used to specify tasks and reward structures for RL agents [51] and their NeSy extension [87] to incorporate logical transitions.

Of significant interest is also the work of [2], who define a pseudo-semantic loss for autoregressive models with constraints and [108], who similarly address the problem of incorporating constraints is LLMs. Both approaches assume a flat vocabulary. We believe NeSyA can be beneficial for constrained autoregressive models when the output structure includes many features, i.e. the model generates structured traces, instead of natural language.

Perhaps the most natural avenue for future work is the definition of a high-level NeSy language for the specification of temporal programs which utilizes NeSyA as a compilation target. Automata are generally low-level devices, cumbersome to define by hand, motivating the creation of a human-centric interface.

5 Neuro-symbolic Complex Event Recognition in Autonomous Driving

5.1 Introduction

In this section we apply the Neurosymbolic Automata framework (NeSy), presented in Section 4 to the task of complex event recognition for autonomous driving and evaluate our approach on a challenging, real world dataset from that domain.

Many applications require processing of continuously streaming data from geographically dispersed sources. Complex event recognition (CER) involves identifying events within these streams, enabling the implementation of both reactive and proactive actions [40]. Beyond their time efficiency, CER systems are valued for their emphasis on trustworthy decision-making. This is achieved through well-defined theoretical frameworks, such as logic specifications and automata, and machine learning methods like Inductive Logic Programming and structure learning, which provide symbolic pattern definitions, sound pattern learning and efficient inference.

However, in applications involving sub-symbolic input, such as video data, there is a need to integrate these symbolic methods with sub-symbolic models to maintain performance. This necessity motivates the introduction of Neuro-Symbolic Artificial Intelligence (NeSy) into the CER domain. NeSy systems integrate neural-based learning with logic-based reasoning, combining sub-symbolic data processing with symbolic knowledge representation. This integration aims to enhance interpretability, robustness, and generalization of sub-symbolic methods, particularly improving their capacity to handle out-of-distribution data.

A relevant domain for the integration of NeSy methods and CER is autonomous driving, since –given the mission-critical nature of this domain– event recognition must be both efficient and reliable. In this context, vehicles must interpret data from cameras and sensors to quickly identify events that may require action. Many events in this domain can be formally described using rules and enriched with background knowledge, which can be effectively defined and leveraged through CER methods.

In this setting, simple event predictors can be modeled using sub-symbolic structures, while complex event recognition is addressed through established symbolic CER frameworks. Several NeSy works have been proposed that handle temporal dynamics present in data sequences [102, 10, 97, 11, 9], but they are application specific and do not offer a generalized framework that learns over a formalization of simple events. On the other hand, generalizable NeSy frameworks such as DeepStochLog [100], DeepProbLog [68], and NeurASP [105] are not inherently designed to model temporal events and need to be enforced with time-aware reasoning (e.g. timestamps, sequential neural models, stochastic processes etc.). One model that addresses both limitations is NeSyA (Neuro-Symbolic Automata) [66], which combines symbolic automata with neural-based perception under probabilistic semantics in an end-to-end differentiable framework. NeSyA supports temporal reasoning while enabling the learning of common symbolic structures used in CER.

This work represents an initial effort to address complex events in autonomous driving with NeSy, and specifically NeSyA, with the incentive to yield better results than purely neural approaches, focusing on the recognition of overtake incidents between agents in the ROAD dataset [81]. The remainder of the paper is structured as follows. Section 5.2 presents the necessary theoretical background, focusing on CER and its relation to symbolic automata and autonomous driving. Section 5.3 outlines our neuro-symbolic approach, explaining the integration of the sub-symbolic models and symbolic

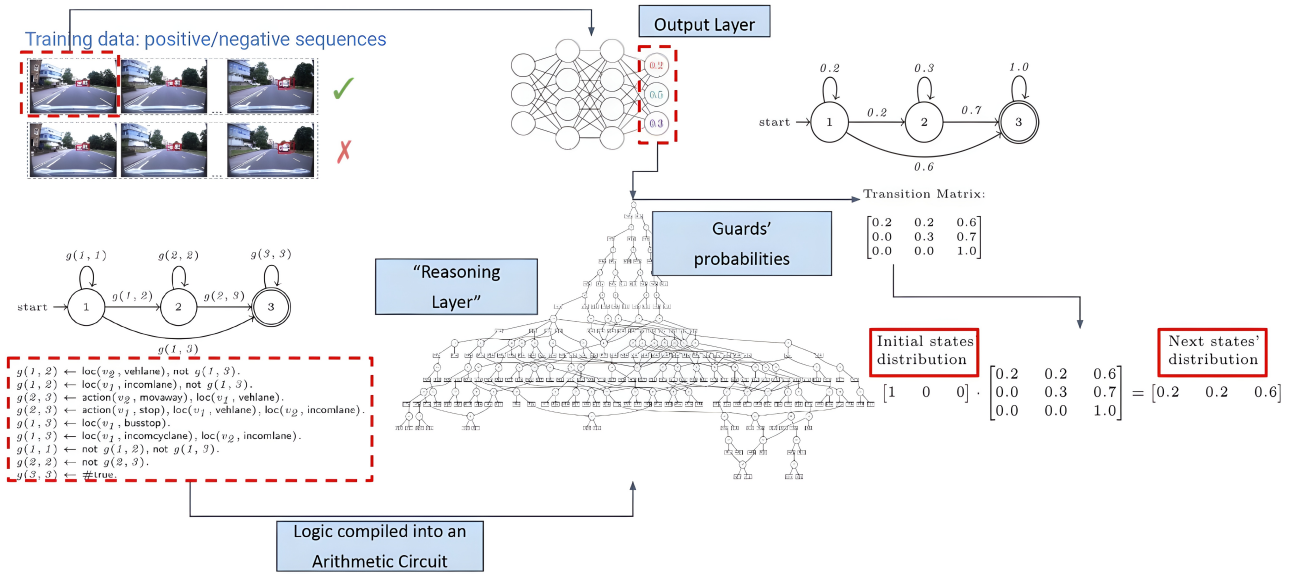


Figure 8: Illustration of the inference and training procedure in NeSy-SFA. First, videos are processed through a neural network that outputs simple event probability distributions (*Output Layer*). These probabilities help answer the probabilistic query of whether the sequence is in a certain state at a given frame (*Guards' Probabilities*), utilizing a compiled Boolean circuit (*Logic compiled into an Arithmetic Circuit*). Over time, each state accumulates probabilities, by multiplying the probability distribution with the *Transition Matrix*, resulting in a final state probability distribution at the end of the sequence. We use this distribution to compute the loss for the 'overtake' incident prediction and backpropagate the loss to train the network, repeating the process until we achieve the minimum loss value.

automata for training and inference. The complete dataset, the experimental setup, results and analysis are provided in Section 5.4, for the challenging task of recognizing the complex event where a road agent overtakes another. Finally, Section 5.6 concludes the paper and outlines directions for future work.

5.2 Background

5.2.1 Complex Event Recognition

Complex Event Recognition (CER), also known as complex event pattern matching, refers to the detection of complex events in streaming data by identifying temporal patterns composed of simple events, i.e. low-level occurrences, or even other complex events [14]. Typically, CER systems operate on streams of event tuples [40, 42], which are time-stamped collections of attribute-value pairs. Conceptually, CER input can be seen as a multivariate sequence, with one sub-sequence per event attribute. For example, an attribute might represent the output of a specific sensor, and its values correspond to the sensor's readings over time, whether numerical, categorical, and/or sub-symbolic. Each event tuple serves as an observation of the joint evolution of all relevant attributes at a specific time point. Complex event patterns define both a temporal structure over these event tuples and a set of constraints on their attributes. A pattern is matched when a sequence of event tuples satisfies both the required temporal ordering and the attribute constraints.

These patterns are typically specified by domain experts using event specification languages [42]. Such languages must support a core set of event-processing operators [5, 40, 107], including: (a) sequence, indicating that specific events must occur in temporal succession; (b) iteration (Kleene Closure), requiring one or more repeated occurrences of an event type; (c) filtering, which restricts

Table 4: Labels and locations for agents, along with available actions for both agents and the autonomous vehicle.

Agent Labels	AV Actions	Agent Actions	Agent Locations
Pedestrian	Stop	Move away, towards	AV lane
Motorbike	Move	Move right, left	Outgoing lane
Bus	Turn right,left	Move	Outgoing cycle lane
Car	Move right, left	Brake	Incoming lane
Medium vehicle	Overtake	Stop	Incoming cycle lane
Large vehicle		Indicating left, right	Pavement
Cyclist		Hazard lights on	Left pavement
AV traffic light		Turn left, right	Right pavement
Other traffic light		Push object	Junction
Emergency vehicle		Reversing	Crossing location
		Overtake	Parking
		Red, Green, Amber light	Bus stop
		Wait to cross / Crossing / Cross from left, right	

matches to events satisfying predefined predicates.

These operators naturally align with a computational model based on Symbolic Finite Automata (SFAs) [26]. Unlike classical automata, which assume finite alphabets, SFAs generalize transitions to be governed by logical predicates over potentially infinite domains, represented using effective Boolean algebras [98, 95]. This enables expressive and compact representations of complex event structures. As a result, most existing CER systems rely on SFA-based pattern representations [1, 107, 44, 4, 25, 8, 21]. In these systems, patterns are typically written in declarative languages (e.g., SQL-like syntax) and compiled into symbolic (often non-deterministic) automata.

5.2.2 CER in Autonomous Driving

Existing work in the autonomous driving domain typically describes activities as driving events, i.e., events occurring during driving [71, 62, 106]. The connection to the CER theory is evident: autonomous vehicles must process numerical and/or sub-symbolic sensor data to recognize driving events. For example, sudden braking may follow a sequence in which a car stops at a red light, accelerates when it turns green, and then brakes abruptly as a deer crosses the road.

Framing these problems as CER tasks is motivated by the fact that many target patterns are either known or can be explicitly defined. When such patterns are not predefined, learning-based methods can be used to discover patterns compatible with CER systems. A relevant example is the ROAD dataset [81, 82], a richly annotated autonomous driving dataset based on the RobotCar dataset [64]. ROAD provides frame-level annotations for agents, including their identity (e.g., vehicle, pedestrian), action(s) (e.g., overtaking, turning left), and semantic location(s) (e.g., left pavement, incoming lane).

From a CER perspective, certain actions, such as ‘overtake’, constitute complex events, while others, such as ‘green traffic light’, represent states. Among these, ‘overtake’ is particularly notable due to its temporal extent, involvement of multiple (simple) sub-events, and significant impact on the scene, making it a compelling CER task. However, the ‘overtake’ pattern is not predefined. Given the complexity of scenes-multiple agents, dynamic locations, and concurrent actions, and the lack of domain experts, manual specification is infeasible. Section 5.4 details the learning approach used to extract such patterns.

5.3 Neuro-Symbolic Approach

To perform CER on the video input, we combine ideas from (sequential) NeSy frameworks and standard CER pipelines: neural networks process sub-symbolic input to detect simple events (actions and semantic locations), while symbolic automata handle pattern matching to recognize 'overtake' incidents. In this Section we will describe in detail the NeSy integration in our work, by outlining the NeSyA framework and its theoretical basis and connecting it to our decisions, driven by the task at hand.

5.3.1 SFAs and Markov Models

In sequence modeling, it is often reasonable to assume that recent observations are more predictive than distant ones. This motivates the use of Markov models, where future states depend only on a limited history, typically just the current or previous state [17]. In these models, transitions between states are governed by probabilities. A model is considered non-stationary if these transition probabilities change over time.

Markov models represent sequences using a state space and a transition function in the form of a matrix that defines the likelihood of moving from one state to another. At each time step, the distribution over states is updated based on the previous distribution and the current transition probabilities. This formulation allows for efficient modeling of temporal dynamics in data.

A seemingly different approach comes from SFAs. Rather than using probabilities, SFAs define transitions using logical conditions over structured inputs. Specifically, inputs are interpreted as truth assignments over a set of propositional variables and transitions occur when the current input satisfies a logical formula attached to an edge in the automaton.

Both frameworks process sequences by transitioning through states in response to observed inputs, whether those inputs are numeric symbols or logical interpretations and when SFAs are applied to data streams (where input patterns or variable co-occurrences can be estimated) transitions can be interpreted probabilistically, much like in a non-stationary Markov chain. So, SFAs can subsume Markov models by encoding structured dependencies while remaining amenable to probabilistic analysis.

5.3.2 Differentiable Probabilistic Inference via SFAs

Probabilistic reasoning over structured domains typically involves modeling uncertainty using joint probability distributions over finite sets of variables [38, 77]. While expressive, these distributions grow exponentially with the number of variables, rendering exact inference intractable. A widely used approach to address this is Weighted Model Counting (WMC), which encodes the probabilistic model as a weighted logical theory, consisting of a propositional formula and a function assigning weights (probabilities) to literals [80]. The probability of a query is then computed by summing the weights of all satisfying assignments, generalizing the classical model counting problem.

This process underlies probabilistic logical inference, where one computes the probability that a logical formula holds under uncertain inputs. Since WMC is a $\#P$-complete problem, practical inference relies on Knowledge Compilation, which transforms formulas into tractable representations, such as deterministic decomposable negation normal form (d-DNNF) circuits [28]. Once compiled, inference becomes linear in the size of the circuit and differentiable.

Symbolic automata define transitions between states using propositional formulas over input variables. When inputs are uncertain or noisy, each transition can be evaluated probabilistically by

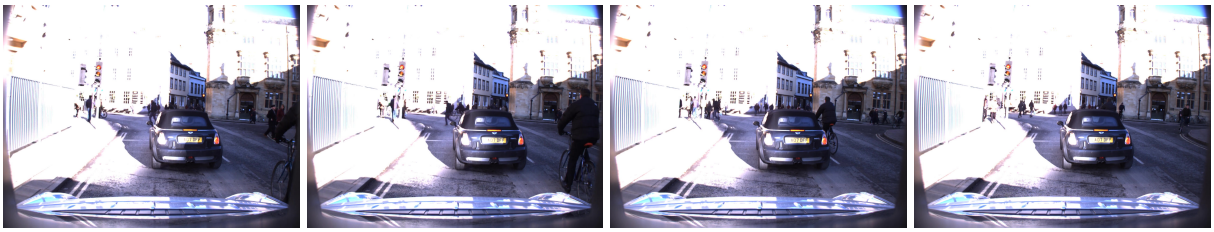


Figure 9: A bicycle approaches from behind the AV, overtakes it while the AV moves forward, and stops at a red light. It then continues to overtake a car that is stopped ahead of the AV at the traffic light.

applying WMC to the corresponding formula. If the automaton is constructed using compiled circuits for each transition, the entire system becomes a differentiable probabilistic model, enabling integration with gradient-based learning methods.

5.3.3 End-to-end training

Let us present in this section the NeSy pipeline in both inference and learning scenarios. Note that the process of probabilistic inference and learning is embedded in NeSyA, but we will not distinguish it here so that the pipeline is more coherent. We begin by outlining the inference process –a single feed-forward pass from input to prediction.

A video is processed by simple event recognition networks, which output probability distributions over simple events, specifically each two agents' actions and semantic locations for every frame. These distributions are then used to classify (ground) the agents' discrete actions and locations for the evaluation of the symbolic automaton. Next, a smooth d-DNNF circuit is compiled from the ASP representation of the automaton. The circuit includes one variable for each possible action and location value, and supports probabilistic queries corresponding to the automaton's transitions. These queries form the transition matrix by computing weighted model counts that accumulate probabilities in the states of the automaton.

For each video, a row vector representing the probability distribution over automaton states at each time step is maintained. It is initialized such that the start state has probability mass 1, with all others set to 0. As each frame is processed, the state vector is updated by multiplying it with the current transition matrix. Each column of the transition matrix represents the probability of transitioning into a particular state at a given frame. Because the transition matrix is computed from neural network outputs, which vary at every timestep, we consider our symbolic automata non-stationary. The final output is the state distribution after processing the last frame.

We now turn to the learning procedure. After each forward pass, the computed state probability distribution can be used to evaluate the prediction loss over the complex event. This loss can be defined over the entire distribution or based solely on the acceptance probability –that is, the probability mass assigned to the automaton's final (accepting) state. Since the compiled symbolic automaton is differentiable, the loss can be backpropagated through the symbolic layer. This enables end-to-end training of the simple event recognition networks via gradient descent. As a result, the model learns to adjust its predictions of simple events in a way that improves recognition of complex events, which in our task is the 'overtake' event through distant supervision. A visualization of the proposed pipeline is presented in Figure 8.

5.4 Experiments

5.4.1 Sequential datasets

ROAD dataset consists of 22 real-world 8-minute videos recorded between November 2014 and December 2015 in central Oxford, covering a range of routes and seasonal conditions. Of these, 20 videos are currently available for training and evaluation.

Road events are defined as a series of bounding boxes linked in time (frames), annotated with the agent's label, action(s), and semantic location(s) (cf. Table 4). Regarding the autonomous vehicle (AV), we only know its unique ego-action (Table 4). Each agent has a unique identifier per video. The dataset includes approximately 122K annotated frames (12 fps) at 1280×960 resolution with a multitude of agents per frame.

Regarding the complex 'overtake' actions, the dataset contains 30 unique overtakes, performed either by the AV or other agents. Durations range from 2 to 164 frames (mean: 49.83; std: 41.87), all occurring within 9 videos. Figure 9 illustrates an overtaking instance from the ROAD dataset.

To enable neurosymbolic integration and construct a pipeline that extracts sub-symbolic information from video and feeds it into a symbolic reasoning module for overtake recognition, we extract two aligned sequential datasets from the complete dataset: one symbolic and one sub-symbolic, in one-to-one correspondence. We differentiate between overtakes involving the AV and those involving two external agents. This distinction is necessary, as each type exhibits different visual and symbolic patterns. When the AV is involved, its position is fixed, and its visual representation is not relevant, unlike scenarios where the AV is not part of the overtake. The dataset consists of sequences ranging from 6 to 10 frames (approximately 0.5 to 1 second), a duration sufficient for humans to recognize overtakes in both symbolic and sub-symbolic modalities.

We define three classes: 0 for negative examples (no overtake), 1 when the first agent overtakes the second, and 2 when the second agent overtakes the first. This labeling explicitly captures the directionality of the overtake. Positive instances were generated by selecting video segments with a maximum length of 10 frames, using non-overlapping chunks to prevent overfitting during NeSy training. A sliding window approach was avoided due to the limited number of positive examples, which would result in highly similar instances. This process yielded 92 positive instances, each concluding with and containing an overtake event.

Selecting negative instances is inherently more challenging, as any sequence not classified as an overtake could theoretically serve as a negative. To ensure informative training, we focused on close negatives: sequences that initially resemble overtakes but do not culminate in one vehicle passing another. To construct these, we identified the action pairs performed by agents prior to overtakes, along with their frequency, and stochastically searched the dataset for similar sequences that do not result in overtakes. Only one instance per agent pair was included, and both agents were required to appear for at least 6 frames. This process yielded approximately 2,000 negative instances. While downsampling negative examples could balance the dataset, we deliberately avoided this approach. Overtake events are inherently sparse, and artificially balancing the dataset would introduce unrealistic conditions. Also training on simplified, artificially balanced data would lead to poor performance, given the sub-symbolic complexity of the task.

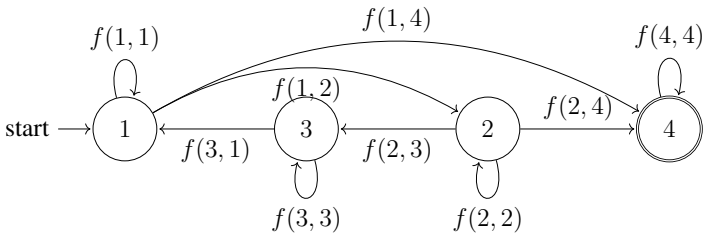
The *symbolic* dataset provides a structured, logic-based representation of events occurring within each frame. Each instance encodes facts describing the two agents involved, including their identity (e.g., AV, large vehicle), actions, semantic locations, and normalized bounding box coordinates at

each timepoint (frame). The instance's class label is also included. This representation enables the grounding of the complex overtake event in terms of simple events, defined by combinations of agent actions and locations within the symbolic framework. The *sub-symbolic* includes the corresponding images of the frames that consist the symbolic dataset.

To ensure unbiased evaluation, we enforced a strict separation between training and testing sets, preventing overlap of augmented positives or negatives from the same video segments. We performed an 80/20 train/test split, analogous to k-fold cross-validation, using disjoint sets of videos for positive samples. This resulted in up to 36 splits, allowing testing on out-of-distribution data. While we initially applied the same strategy to negative samples, we observed a drawback: videos vary significantly in visual characteristics (e.g., snow-covered vs. leafy junctions), and training solely on one type reduces generalization. To mitigate this, we allowed negatives from all videos but enforced a minimum temporal distance of 100 frames between any two selected instances, avoiding redundancy while maintaining visual diversity.

To simplify the task, we focused only on one positive class and overtakes not involving the AV. As a result, not all data splits remained suitable, since some lacked relevant positives or exhibited more positives in the testing set. We randomly selected four viable splits for training and evaluation. Across these splits, the number of positive sequences in the training set ranges from 46 to 75, and from 17 to 46 in the test set. The corresponding number of negative sequences is approximately 550 for training and 250 for testing.

5.4.2 Extracting Background Knowledge




```
% State 1 -> 2: if agent 2 is moving towards the AV and not transitioning to State 4.
f(1,2) :- action_2(movtow), not f(1,4).
% State 1 -> 4: if agents are in the same lane and the agent 2 is moving away from the AV.
f(1,4) :- same_lane(l1, l2), action_2(movaway).
% Stay in State 1: if not moving to State 2 or 4.
f(1,1) :- not f(1,2), not f(1,4).

% State 2 -> 3: if agent 2 is moving towards the AV and not transitioning to State 4.
f(2,3) :- action_2(movtow), not f(2,4).
% State 2 -> 4: if agent 1 is stopped and agent 2 is in the incoming lane.
f(2,4) :- action_1(stop), location_2(incomlane).
% Stay in State 2: if not moving to State 3 or 4.
f(2,2) :- not f(2,3), not f(2,4).

% State 3 -> 1: if agent 1 is in the incoming lane.
f(3,1) :- location_1(incomlane).
% Stay in State 3: if not moving to State 1.
f(3,3) :- not f(3,1).

% Stay in State 4: always; absorbing state.
f(4,4) :- #true.
```

Figure 10: Learned automaton from symbolic dataset. `l1` and `l2` denote the agents' (with the respective index) locations. It achieves an *F1-score* of approximately 0.87 on the test set. The actual ASP syntax has been simplified for clarity of the illustration.

Since 'overtake' patterns were not predefined, we employed the ASAL framework [56] to learn the patterns from the symbolic sequential dataset. ASAL learns Answer Set Automata, an extension of SFAs tailored for CER over multivariate event streams, where transition predicates are defined via ASP rules. Through declarative learning with symbolic reasoning it produces compact models with strong generalization performance.

We used ASAL with the objective of maximizing generalization on the test set. We learned a general automaton from the different symbolic splits. This led to the selection of a subset of simple events most relevant for complex event recognition. The selected actions were: *moving away*, *moving towards*, *stop*, and *other* (none of the above). The selected semantic locations were: *incoming lane*, *vehicle lane*, *junction*, and *other*. Intuitively, this aligns with human reasoning: recognizing an 'overtake' primarily requires understanding the orientation and motion direction of the vehicle.

The above process resulted in the automaton shown in Figure 10. This learned symbolic automaton accepts multiple patterns as valid instances of overtakes, represented by different paths leading to the accepting state. Examples of such paths include: `f(1,1)` → `f(1,1)` → `f(1,2)` → `f(2,4)` or `f(1,1)` → `f(1,4)`. Let us give an intuitive overtaking pattern that is validated by the shortest accepting path `f(1,4)`:

- AV detects two vehicles in the same lane as itself (vehicle lane)
- Both vehicles are visible in front of the AV, meaning they are positioned side by side without overlapping in the AV's field of view
- If one of these vehicles is detected as moving, while the other is static or moving slower, the moving vehicle is classified as overtaking the other

5.4.3 Experimental Setup

In a higher level of abstraction, the task is framed as a binary sequence classification problem: determining whether a given sequence of frames constitutes an 'overtake'. Experiments were conducted on the four (sub-symbolic) data splits described in Section 5.4.1. We trained NeSy models and compared their performance against purely neural baselines.

For simple event recognition, we employed two architectures: a 2D-CNN for semantic location prediction and a 3D-CNN for action recognition, both with multiple convolutional layers. The temporal modeling capability of the 3D-CNN is particularly important for recognizing motion-based actions. Each module outputs eight predictions per frame: probability distributions over the actions and locations of each agent. Although the annotations are multi-label (e.g., an agent may simultaneously move toward the AV and signal a left turn), the task is cast as multi-class due to the requirement in the NeSy pipeline for probability distributions over mutually exclusive classes. Both networks receive the same input: a 10-frame video segment and bounding boxes of the two agents of interest per frame.

To evaluate the temporal reasoning capabilities of our NeSy model, we compare it against a standard spatio-temporal neural architecture: a Long Short-Term Memory (LSTM) network [50]. In this baseline, the outputs of the simple event recognition modules are passed to an LSTM (hidden size 10), whose output is used to predict the final classification probability.

For training, we used the Adam optimizer [58] with a batch size of 8. Due to the differing temporal context –80 frames for the semantic location network versus 8 for the action recognition network– we set distinct learning rates for each. Empirically, we found that the semantic location module required a

lower learning rate, so we used 10^{-5} for the 3D-CNN action recognizer and halved it for the location module.

All CER models were trained for a fixed 40 epochs. The neural baseline took approximately 20 seconds per epoch, whereas for the NeSy approach took 30 seconds. Given the scarcity of positive examples in the training set, we did not employ a validation set. Instead, model selection was based on training loss dynamics: we normalized losses to the $[0, 1]$ range using the first epoch's loss as the maximum and 0 as the minimum, then selected the model at the earliest epoch where the loss plateaued, defined as a change of less than 0.05 across a window of two consecutive epochs.

Since 'overtake' instances are sparse, comprising only 10% of the dataset, the task becomes a highly imbalanced binary classification problem. To address this, we evaluated two loss functions for NeSy and baseline training: *weighted binary cross-entropy* (weighted BCE) and *focal loss*. While weighted BCE increases the contribution of the minority class by reweighting class loss terms, focal loss down-weights easy examples, focusing learning on harder, misclassified ones.

In the neural baseline, outputting a complex event probability is straightforward. In contrast, the NeSy model produces a state probability distribution over the automaton. The first and last entries in this vector correspond to the start and accepting states, respectively. We experimented with two approaches for mapping this distribution to a classification probability: (a) using only the acceptance probability, and (b) comparing the full state distribution to the target distribution $(0, 0, 0, 1)$ using the Kolmogorov-Smirnov (KS) distance. The KS distance provides a bounded $[0, 1]$ similarity score between cumulative distributions, offering a principled, interpretable metric to evaluate whether the final state is reached.

5.5 Results and Discussion

5.5.1 End-to-end NeSy

Our primary objective is to evaluate complex event recognition, i.e., the recognition of the 'overtake' event, across the four sub-symbolic data splits. To ensure a fair comparison across splits with imbalanced class distributions, we adopt the micro-averaged F1 score as our evaluation metric across all data splits. Table 5 presents the comparative results on complex events for the NeSy and baseline for all loss configurations.

Metric	Baseline		NeSy			
	Focal	Weighted BCE	Focal		Weighted BCE	
			States	*Final*	*States*	*Final*
Micro F1	0.15	0.14	**0.55**	0.42	0.31	0.39

Table 5: Micro-F1 scores by model type, loss function, and NeSy probability variant. 'States' uses the full state distribution; 'Final' uses only the acceptance probability. A random 50% predictor yields 0.13 micro-F1.

Overall, the NeSy counterpart outperforms the neural baseline by a large margin across all configurations. Additionally, focal loss yields better performance than weighted BCE in both model types. However, no single acceptance probability computation strategy consistently outperforms the other across all loss types within the NeSy configurations. Specifically, using the full state probability distribution is superior when employing focal loss, whereas relying solely on the acceptance probability

yields better results under weighted BCE.

This discrepancy can be attributed to the characteristics of each loss function. Focal loss is particularly effective at emphasizing hard, misclassified examples, especially from the minority class. In such cases, the richer information provided by the full automaton state distribution enables finer-grained adjustments that help reduce loss more effectively. The KS-derived score, computed from the full distribution, provides a softer, less confident prediction signal that is less biased and better reflects uncertainty across states. Focal loss benefits from this nuance, as it is designed not for probability calibration but for modulating loss based on prediction confidence. In contrast, weighted BCE operates as a weighted maximum likelihood estimator under asymmetric class priors, assuming calibrated, true probabilities as input. Consequently, it performs best when provided with a single, well-defined probability –such as the acceptance probability– rather than a heuristic proxy derived from distributional similarity.

5.5.2 Evaluation on Simple Events

However, as seen in Table 5, the F1 scores on the testing set remain relatively low. Again, as mentioned in Section 5.4.1, the computer vision task itself is difficult, so low scores in the distant supervision task of classifying an 'overtake' is expected. Additionally, for the neural baseline, this outcome is expected due to the high variability among 'overtake' instances, which hinders generalization. In contrast, the reduced performance of the NeSy model suggests deficiencies in simple event recognition, since the symbolic automaton, demonstrates high generalization on the testing set.

To investigate this hypothesis, we overfit a NeSy model on the training set and then evaluate its simple event predictors directly on the training data. As shown in Table 6, although the model achieves perfect recognition of 'overtake' instances, it relies on what can be described as reasoning shortcuts: it learns to exploit superficial cues in the input to satisfy the automaton transitions without truly understanding or modeling the intended semantics of the simple events. Note that in preliminary experiments we also used pre-trained simple event predictors, but the complex event training still managed to find the best training shortcut.

Complex Event (F1-Score) - *Training Set*			0.99		
Action			**Location**		
Class	Micro-F1	Support	**Class**	Micro-F1	Support
Move away	0.301	1335	Vehicle Lane	0.000	1321
Move towards	0.273	6752	Incoming Lane	0.137	7194
Stop	0.301	2350	Junction	0.000	1967
Other	0.000	3963	Other	0.687	3918

Table 6: Trained complex event predictor evaluated on simple events for only one split. The model is overfitted on the training set to isolate the symbolic component's behavior. Evaluation is reported as per-class F1 scores (one-vs-all) for each simple event category.

5.5.3 Loosely coupled NeSy

Given the sub-optimal performance of the NeSy model, one natural consideration is to decouple training and reasoning, i.e., to first train the simple event predictors independently, and then incorporate the symbolic component only at inference time.

Two approaches are possible: (a) utilizing the entire dataset for the simple event prediction task, and (b) utilizing only the sub-symbolic dataset splits defined for the end-to-end NeSy task, as described in

(a) Trained on the whole dataset

Action		Location	
Class	F1 Score	**Class**	F1 Score
Move away	0.980	Vehicle Lane	0.767
Move towards	0.857	Incoming Lane	0.822
Stop	0.889	Junction	0.905
Other	0.861	Other	0.715

(b) Trained on the complex event dataset

Action		Location	
Class	Micro-F1	**Class**	Micro-F1
Move away	0.328	Vehicle Lane	0.504
Move towards	0.686	Incoming Lane	0.408
Stop	0.573	Junction	0.491
Other	0.158	Other	0.532

Table 7: Simple event training and F1 scores for action and semantic location recognition on the testing set using two training configurations: (a) (left) models trained on a randomly selected 60% of the dataset, validated on 20% with early stopping, and tested on the remaining 20%; (b) (right) models trained on the four sub-symbolic splits used for the complex event task. Actions and locations for both agents are evaluated jointly to keep the table simple.

Section 5.4.1. The evaluation results for the simple event predictors trained using these two approaches are presented in Table 7. As expected, leveraging a larger portion of the dataset for training leads to improved performance in simple event recognition. However, since our primary evaluation pertains to the NeSy training and inference process, we proceed with the simple event predictors trained on the dataset used for the end-to-end NeSy component. This configuration serves as the baseline for the current task definition and dataset setup.

If we evaluate 'overtake' recognition using the pre-trained simple event recognizers by appending the symbolic automaton, the results show that relying solely on this sequential setup, without end-to-end training, yields a complex event F1 score of 0.0, indicating that end-to-end training is essential for achieving non-trivial performance.

However, while the overall complex event performance is low, a score of exactly zero suggests further investigation. We therefore conduct an additional experiment in Table 8, where we evaluate complex event recognition while selectively fixing some simple event predictions to their ground-truth labels. This allows us to assess whether the accurate prediction of specific simple events has a disproportionately large influence on complex event recognition and whether certain errors in simple event prediction are particularly detrimental.

Fixed Simple Events	Micro F1
None	0.00
All	0.87
`action_1`, `action_2`	0.43
`location_1`, `location_2`	0.01
`action_1`, `location_2`	0.81

Table 8: Results on the loosely coupled NeSy structure. Simple event predictors trained on the NeSy dataset splits are evaluated on the complex event. Some simple events are given their true labels during evaluation. `action_1` refers to agent 1's action, `location_1` to their location, etc.

If we provide the symbolic automaton with the ground-truth distribution of all simple events, as expected, we recover the automaton's maximum F1 score on the testing set (cf. Figure 10). When providing only the ground truth for the agents' actions, the 'overtake' recognition F1 score increases to 0.43. In contrast, supplying only the ground truth for the agents' semantic locations yields a much

lower score of 0.01. Interestingly, when fixing agent 1's action and agent 2's location to their true values, the F1 score rises to 0.81, very close to the automaton's upper limit.

This observation highlights that not all simple event predictions contribute equally to complex event recognition. Intuitively, one might expect that accurate semantic location predictions would significantly improve performance, as predicates such as `same_lane`, `location_1`, and `location_2` appear in multiple transitions within the automaton, but that is not the case. On the contrary, examining the learned automaton reveals that `action_1` is involved only in the transition `f(2,4)`, where it is conjuncted with `location_2(incomlane)`. Accurately predicting this specific conjunction appears to be critical for achieving high complex event recognition performance. These results indicate that certain transitions in the symbolic automaton are more crucial for temporal reasoning than others, and accurate prediction of the literals involved in these key transitions has a disproportionately large impact on overall complex event recognition.

5.6 Conclusions and Future Work

In this work, we presented a Neuro-Symbolic (NeSy) pipeline for Complex Event Recognition, focusing on the recognition of overtake incidents between two vehicles from video data. Our experiments demonstrate that the NeSy model significantly outperforms its purely neural counterpart across all configurations.

We also evaluated the learned simple events as well as a loosely coupled NeSy setting. Interestingly, our findings show that the end-to-end NeSy model does not rely solely on accurate simple event predictions for correct complex event recognition; instead, it is subject to reasoning shortcuts. In the loosely coupled setting, we observed that the importance of specific simple events depends more on their role in key automaton transitions rather than on their frequency within the automaton structure.

A primary direction for future work is the reformulation and expansion of the dataset. Incorporating more data and a broader range of complex events would address one of the main limitations of our study, namely, the limited training data combined with the inherent complexity of the computer vision tasks involved.

Another promising direction is the systematic study of the relationship between symbolic automaton structure and NeSy training dynamics. It is plausible that certain automaton architectures are more suitable for guiding the neural component. For instance, automata with fewer conjunctive conditions in their transitions may make the simple event training easier, while more complex automata could offer smoother convergence or improved generalization.

Finally, a highly relevant avenue is the joint learning of both the neural and symbolic components. Instead of fixing background knowledge in advance, we could provide a flexible knowledge base and allow the system to learn both the automaton structure and the neural network parameters simultaneously. While this approach poses considerable challenges, it holds the potential for creating more flexible and powerful models that can incorporate symbolic knowledge without introducing domain-specific biases.

6 Event Pattern Learning

6.1 Introduction

In the previous sections we assumed that the symbolic knowledge that is input to a NeSy system is given beforehand. Under this assumption we presented techniques tailored specifically for temporal event-based domains, for training the neural part of the NeSy system alongside the knowledge, so that the two components (neural/symbolic) are aligned. However, the assumption of existing knowledge does not always hold. In many event-based applications the patterns that we are interested to monitor are not known beforehand, or they evolve over time as the underlying processes or data characteristics change. This motivates the need for learning such patterns directly from data, rather than relying on hand-crafted specifications, allowing us to automate part of the pattern engineering process and to seamlessly adapt to new situations. However, although knowledge learning in a purely symbolic setting (e.g., inductive logic programming, automata and grammar induction) is a well-studied problem, it is much less explored in the case where the input data are of perceptual nature, such as videos (image sequences) or high-dimensional time-series data.

A straightforward approach is to train a neural network to map percepts to symbols and then use an off-the-shelf symbolic learner to induce temporal structure from the network's symbolic inferences. This yields a *loosely coupled* NeSy system, where the neural and the symbolic components are trained in isolation, from different data and for different tasks. To illustrate the case, consider the autonomous driving domain from Section 5 and assume that we do not know the automaton that represents the overtake pattern that we wish to monitor. To learn it from the input video feeds via a loosely coupled NeSy approach we would need dense simple event labels to train a neural network to detect simple events. We would then use this network to extract symbolic sequences from the images, annotate the sequences with complex event labels (overtake incidents), and use a symbolic learner, e.g. ASAL, to learn the automaton from these symbolic sequences.

This approach has several shortcomings: (i) acquiring dense, high-quality simple event supervision is costly and often infeasible, given the volume and velocity of real-life event-based data streams; (ii) symbolic learners often rely on combinatorial search and, as a result, their running times and memory requirements scale exponentially with the dimensionality and length of the training sequences and with the size of the symbolic vocabulary (number of symbols or predicates) produced by the perceptual module; (iii) training the neural part of the NeSy system and learning its symbolic component separately, in a pipeline, is sub-optimal: neural training ignores the downstream task that we are primarily interested in and symbolic learning ignores the stochastic, error-prone nature of the neural grounding process that produces its input symbols.

To address these issues, in this section we present advancements on NeSy temporal structure learning from perceptual data. We take first steps towards tackling points (i) and (iii) above by presenting a method for *joint neural/symbolic training* that combines ASAL [57]—also presented previously in D4.1—and NeSyA [66]—see also Section 4. The input consists of training image sequences with downstream (complex event) labels only, together with a small subset of images from these sequences for which simple event labels are also available. ASAL and NeSyA are combined in a co-training framework in which a perceptual neural network is initially partially trained on the small pool of labeled images. The partially trained network is then used to predict simple event labels for all images, thereby inducing noisy symbolic sequences from the raw training sequences. ASAL is

subsequently used to learn an initial symbolic automaton (SFA) from these sequences. To account for the noise introduced by the poorly trained neural network, each sequence is weighted by the average entropy of the neural predictions across time, and ASAL incorporates these weights during induction, alongside a Minimum Description Length (MDL) heuristic, to bias its search towards models that explain low-entropy sequences while discounting high-entropy ones when they incur a large MDL penalty. NeSyA is then used to further train the neural network for a few epochs, using the SFA induced by ASAL as a "teacher". In this step, two losses are combined: the standard NeSyA loss from sequence misclassification and the image-level misclassification loss for the labeled images. The system then employs active learning principles to identify the most informative images to label and requests a batch of new labels under a query budget. ASAL induces an improved SFA using the newly acquired labels (rather than the network's pseudo-labels), and the joint training loop continues until convergence. This approach ensures that the neural and symbolic components are trained jointly and mutually informed, while preliminary experiments indicate that convergence is achieved with only a fraction of the labeled images that would be required without NeSy training or without an active learning heuristic.

We also present a neuro-symbolic framework that addresses primarily point (ii) above by replacing combinatorial symbolic search with fully differentiable SFA learning from perceptual sequences. Instead of repeatedly invoking ASAL on large symbolic datasets, our new approach, ∂SFA, fixes an SFA topology (number of states and candidate transitions) and parameterizes each transition guard as a neural DNF over learned base predicates, implemented as differentiable conjunction and disjunction operators on top of a perceptual network. Given a sequence of inputs, the corresponding guard truth values are converted into a probability distribution over the SFA states' outgoing transition and a classical forward algorithm is used to compute the sequences' acceptance probabilities at the end of each sequence. The fuzzy relaxation and the fully differentiable forward pass through the SFA graph yield an objective that can be optimized end-to-end from sequence-level labels using standard gradient-based methods, without performing explicit symbolic search during training. Post-training, an extraction pipeline prunes small or redundant weights, sweeps thresholds to discretize the neural DNFs into candidate Boolean guards, and uses a lightweight ASP-based meta-encoding to select which guards and literals to retain, subject to structural constraints such as determinism and sparsity. In this way, ∂SFA recovers a compact, human-readable SFA that closely matches the behavior of the trained differentiable model, while training and extraction are significantly faster and more scalable than running ASAL from scratch on the same data.

6.2 Related Work

Most existing work on learning SFA has focused on symbolic, *active learning* algorithms, typically formulated in Angluin's MAT (Minimally Adequate Teacher) setting [7] with membership and equivalence queries (MQ/EQ) [34, 12, 65, 74, 35]. In this setting, a learner repeatedly queries an oracle about whether a word belongs to the target language (membership queries) and whether a conjectured automaton is equivalent to the unknown target, receiving counterexamples in the opposite case (equivalence queries). These assumptions are often impractical in many data-driven applications: MQs requires "probing" the environment/system under all synthetically generated event sequences, and EQs assume a teacher capable of certifying that a learned SFA matches the unknown pattern, or providing informative counterexamples when it does not. In practice, however, what is often available is only a finite log of labeled executions and no access to such oracles.

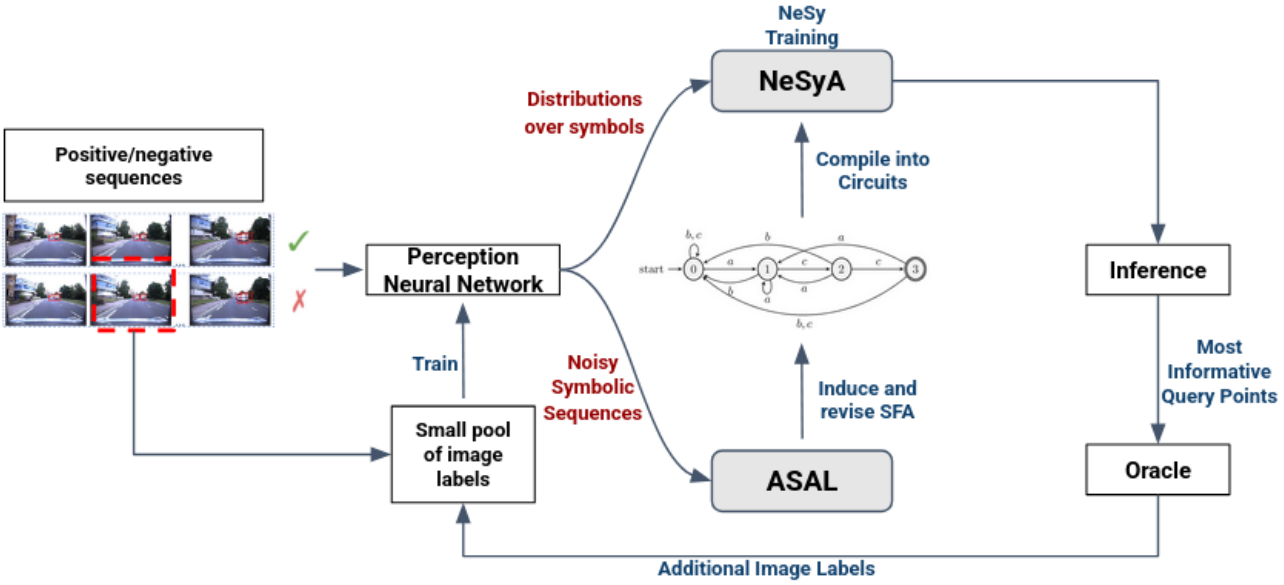


Figure 11: NeurASAL overview.

In contrast, in the *passive learning* setting, the learner receives a sample of positive and negative sequences and must infer an automaton consistent with the sample, or, when learning in the presence of noise, an automaton that approximates the data as close as possible. This setting is often more natural in data-driven applications [35, 16]. For classical automata, state-merging algorithms such as RPNI and its variants are widely used in this setting, while also several SAT-based approaches have been proposed. For SFA, most existing techniques rely on Inductive Logic Programming [24] and logic-based learning [57, 55, 72, 36]. These approaches suffer from scalability issues, since they rely on combinatorial search in a space that grows exponentially in the number of states in the target SFA, the number of predicates in their guards and the length of the input training sequences. Besides, scalability is a well-known issue of all purely symbolic SFA induction techniques, either in the passive, or in the active learning setting: efficient induction algorithms in either setting exist only for certain classes of SFA, specifically those defined over *monotone* Boolean algebras – e.g. the interval algebra over the reals – but *not* for in the general case – e.g. the propositional algebra where the SFA guards are Boolean combinations of arbitrary predicates [35].

Aiming to sidestep the hardness of exact automata identification, differentiable approaches have recently emerged, where neural architectures approximate deterministic finite automata (DFAs) from symbolic sequences and can be trained using gradient-based optimization [45, 88, 33]. Such methods demonstrate that continuous relaxations of automata can be learned efficiently and that discrete automata can be extracted post-hoc from trained models. However, these approaches operate over discrete symbol sequences and assume a fixed finite alphabet; they do not handle multivariate, sub-symbolic input (e.g., sensor streams, images, or high-dimensional feature vectors) directly, and they do not provide a systematic mechanism for learning *symbolic* transition guards over relational or numerical predicates, as required in CER/F.

6.3 Active Neuro-Symbolic Learning of Complex Event Patterns

In this section we introduce NeurASAL, a novel approach that combines ASAL for symbolic automata (SFA) induction [57], see also Deliverable D4.1, with NeSyA [66], Section 4, for joint neuro-symbolic training of perception networks with such SFA. NeurASAL – see Figure 11 for an overview. jointly

learns an SFA while training the perception network and targets temporal learning problems in which supervision is cheap at the sequence level, but expensive at the latent concept level – a typical setting in complex event recognition and forecasting tasks. The input consists of sequences of perceptual observations (e.g., images) labeled only as *positive* or *negative* with respect to some unknown complex event pattern, together with a small, actively acquired pool of image-level labels for a small number of sequences. The goal is to learn simultaneously: (i) a *symbolic finite automaton* (SFA) describing the temporal structure of the pattern; and (ii) a perception model that maps each observation into probabilities for a set of base predicates (latent concepts/simple events), such that the learned SFA accurately predicts the sequence labels and the perception model is aligned with this structure.

The perception component is a convolutional neural network (CNN) that, given an image x_t, outputs a probability distribution over a set of simple predicates. These predicate probabilities form the interface between the neural and symbolic parts. Starting from a small seed set of fully labeled sequences, where every image in the sequence has a latent concept label, the CNN is first trained with standard cross-entropy on these image labels. The partially trained CNN is then applied to all images in the training set, producing *noisy symbolic sequences*: for each time step, the most probable predicate configuration is treated as a pseudo-label, and each sequence is associated with an overall confidence score, typically the average entropy of the CNN predictions across the sequence.

ASAL is then used in a purely symbolic fashion on these noisy sequences. It takes as input: (i) the pseudo-labeled sequences; (ii) their sequence-level labels; and (iii) a weight for each sequence derived from the CNN's confidence. ASAL searches in the space of SFAs up to a fixed number of states, guided by a weighted Minimum Description Length (MDL) criterion. Fully labeled sequences and highly confident pseudo-labeled sequences act as strong constraints, while uncertain sequences are discounted by their lower weights and can be ignored if they incur a large MDL penalty. This results in an initial SFA whose guards are Boolean combinations of the base predicates and which approximates the unknown target pattern despite the noise in the symbolic input.

To align the perception model with this symbolic structure, NeurASAL then enters a neuro-symbolic training phase using NeSyA. The SFA produced by ASAL is compiled into an arithmetic circuit – or more generally a tractable probabilistic circuit – that computes, for any sequence of predicate probability vectors produced by the CNN, the acceptance probability of the SFA. This circuit implements differentiable dynamic programming over the automaton, allowing gradients to flow from a sequence-level loss back to the CNN parameters. During this phase, two losses are combined: a *sequence loss*, such as binary cross-entropy between the SFA acceptance probability and the sequence label, and an *image loss*, i.e., cross-entropy on the available image-level labels. Optimizing the sum of these losses refines the CNN so that its predicate predictions not only fit the few image labels but also make sequences symbolically consistent with the current SFA.

NeurASAL wraps the ASAL and NeSyA phases inside an active learning loop. After each round of neuro-symbolic training, the system evaluates the current model on the pool of unlabeled sequences and selects the sequence for which the sequence-level prediction is most uncertain or most erroneous (e.g., highest binary cross-entropy or largest margin). For that sequence, an oracle (typically, a human annotator) is asked to provide latent labels for all images in the sequence. These new labels are added to the labeled pool, and the loop continues. ASAL re-induces an SFA using the updated symbolic dataset, where newly labeled sequences are treated as highly reliable, NeSyA retrains the CNN under the new automaton, and the active learner selects the next query.

In this *co-training approach* the symbolic learner continually reshapes the structure that supervises

the CNN, while the CNN provides increasingly accurate pseudo-labels that improve the quality of the SFA. NeSyA allows the SFA to act as a differentiable teacher for the CNN, propagating sequence-level supervision to the latent level without requiring dense labels. Active learning further reduces labeling costs by focusing annotation effort on the most informative sequences. Crucially, unlike a loosely coupled pipeline where a CNN is trained first and a symbolic learner is applied post hoc to its outputs, NeurASAL performs *joint* learning: the SFA directly influences the representation learned by the CNN, and the evolution of the CNN feeds back into the SFA through ASAL. The result is a neuro-symbolic system in which perception and temporal reasoning are tightly aligned and can be trained from very few latent labels.

6.4 Indicative Results

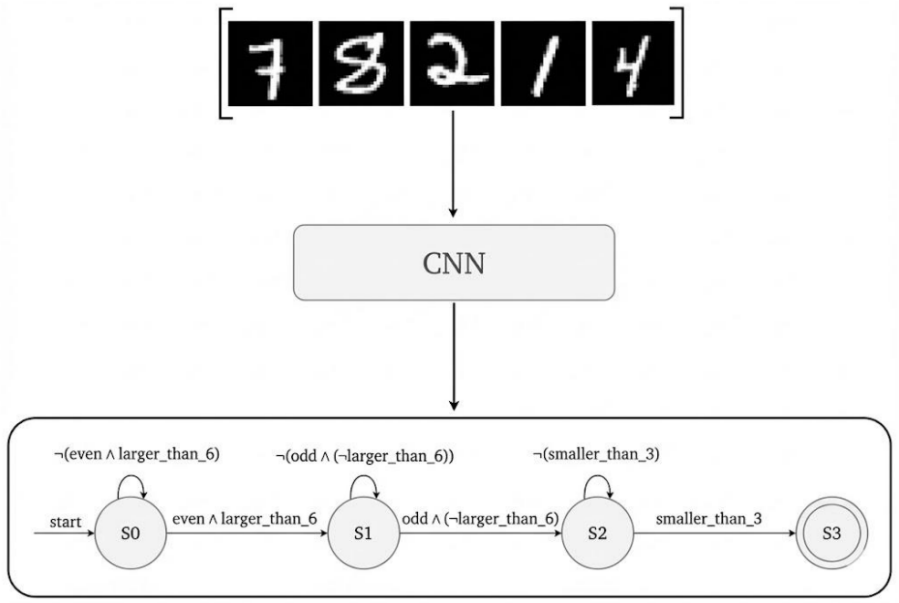


Figure 12: A simple MNIST-based SFA used to generate data in the active learning experiment.

We evaluate NeurASAL on a simple temporal MNIST benchmark tailored to complex event recognition. Each example is a sequence of ten MNIST digits, and labels are generated by a hand-crafted SFA over predicates such as *even*, *odd*, *larger_than_6* and *smaller_than_3* – see Figure 12. A representative pattern starts in state S_0, waits until it sees a digit that is even and larger than 6, moves to S_1 and then requires a digit that is odd and not larger than 6, finally transitioning to S_2 and accepting once it sees a digit smaller than 3. Self-loops on each state ensure that irrelevant digits are ignored. Positive sequences are those that realize the pattern somewhere in the ten positions; all other sequences are negative. The CNN receives the raw digit images; sequence labels are available for all training sequences, whereas only a small number of sequences come with latent concept labels (true values of the predicates per time step).

We compare NeurASAL against DeepDFA [88], a fully differentiable Deterministic Finite Automaton (DFA) learning method, the literature approach that is most closely related to ours. The method learns a probabilistic automaton with gradient-based optimization, properly regularized so that the outcome closely approximates a crisp DFA, which can then be easily extracted from the neural model. DeepDFA learns *classical* DFA over a fixed alphabet, as opposed to symbolic automata over a pool of predicates that can account for infinite alphabets. Typically, DeepDFA can be used in multi-variate

settings via *propositionalization*. However, this was not necessary in this simple, univariate MNIST-based domain: DeepDFA can learn over the digits alphabet, at the potential cost of inducing more complex automata, as opposed to symbolic automata learning methods, which can compress alphabet symbols (and transitions) into logical formulas. Moreover, DeepDFA was originally designed for learning from symbolic strings/words. We thus extended the method so that it can handle perceptual input (images), process by a CNN head. The purpose of the comparison is to show that thanks to ASAL's crisp structure induction, as opposed to DeepSFA's purely neural approach, NeurASAL is able to converge to high-quality models, faster than the state-of-the-art DeepDFA method.

For a fair comparison we equip both methods with the same CNN architecture as a perception module and let them share the same pool of labeled sequences. The experimental training protocol proceeds in rounds. Initially, we provide fully labeled latent concepts for only four sequences (40 images). The CNN is trained on these images, and then in parallel: DeepDFA learns a DFA from the corresponding symbolic sequences, while NeurASAL runs the ASAL+NeSyA loop to induce an SFA and train the CNN under its supervision. After this initial phase, the active learner selects the sequence with the highest sequence-level binary cross-entropy and queries latent labels for that sequence. The labeled pool grows by one sequence per round, and both systems are retrained for a fixed number of epochs at each step. We repeat this procedure for ten rounds, reaching at most fourteen fully labeled sequences (140 labeled images) in total.

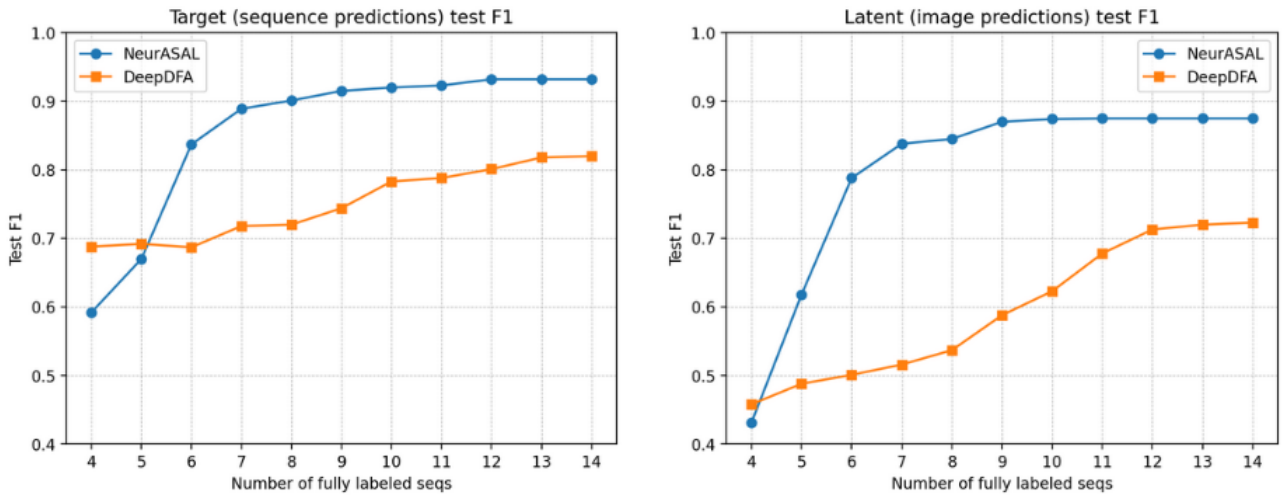


Figure 13: Sequence classification (left) and image classification F_1-scores for NeurASAL and DeepDFA on temporal MNIST.

Figure 13 (left) shows the test F1-score on sequence classification (target task) as a function of the number of fully labeled sequences (averages over five runs). DeepDFA starts around $F_1 \approx 0.68$ with four sequences and improves steadily as more latent labels are acquired, reaching roughly 0.82 with fourteen labeled sequences. NeurASAL exhibits much faster improvement: with only six fully labeled sequences it already surpasses DeepDFA, and by eight to ten fully labeled sequences it reaches an F_1-score in the range 0.9–0.93. Therefore, NeurASAL needs roughly an order of magnitude fewer latent labels to achieve the same sequence-level performance that DeepDFA only attains when trained with about one hundred fully labeled sequences.

The advantage is even more evident on the latent concept (simple event) prediction task. Figure 13 (right) reports F1 on image-level label prediction. For DeepDFA, latent concepts are learned only indirectly through the CNN trained on the limited labeled pool, yielding modest performance (around

Model	Train F1 (T=10)	Test F1 (T=10)	Test F1 (T=50, OOD)
NeurASAL	0.968	0.899	**0.823**
CNN + LSTM	1.000	0.916	0.554
CNN + Transformer	1.000	0.967	0.542

Table 9: Out-of-distribution generalisation from length-10 to length-50 image sequences. All models are trained on length-10 sequences.

0.72 when extrapolating to many labeled sequences). In contrast, NeurASAL uses the SFA learned by ASAL as a teacher that provides indirect supervision for every time step in every sequence, via the NeSyA circuit. As a result, the CNN learns latent concepts much more effectively, reaching latent F_1-scores of approximately 0.88 with the same small set of labeled sequences. Additional ablation experiments confirm that both components are necessary: training the CNN without any neuro-symbolic loss but with the same set of latent labels yields image-level F1 around 0.72, while using the SFA for indirect supervision only (no extra latent labels) achieves about 0.65; combining NeSy training with extra latent labels pushes performance to approximately 0.875.

Out of Distribution Generalization In our second experiment we investigate NeurASAL's Out of Distribution Generalization (OOD) performance, thanks to its ability to learn invariant symbolic patterns and use them to train the perception module. Our OOD setting is based on temporal length: All models under comparison are trained on sequences of length 10 and tested on sequences of length 50 generated by the same underlying SFA pattern. We compare NeurASAL against two purely neural baselines: a CNN followed by an LSTM, and a CNN followed by a Transformer. Table 9 presents that results. All three models achieve high training performance and similar test F_1-scores on length-10 sequences. However, when evaluated on length-50 sequences, NeurASAL suffers a moderate drop in F_1-score, while the purely neural models drop to near random guessing. The hybrid models learned by NeurASAL can handle longer horizons because the automaton in the symbolic component is length-invariant. In contrast, the neural baselines, have learned to exploit superficial regularities in sequences of length 10 and fail to extrapolate to longer temporal contexts.

Overall, these experiments demonstrate three key benefits of NeurASAL. First, by combining ASAL, NeSyA and active learning, it can recover high-quality SFAs and achieve strong sequence-level performance from very few latent labels, substantially outperforming the state-of-the art DeepDFA method in the low-supervision regime. Second, the learned SFAs serve as effective teachers for the perception network, leading to significant gains in latent concept prediction accuracy compared to training the CNN alone or using DeepDFA. Third, the explicit temporal structure encoded in the SFA yields robust OOD generalization, whereas purely neural sequence models' performance degrades sharply. These results highlight the value of tightly integrated neuro-symbolic learning in temporal domains where labels are scarce and extrapolation beyond the training distribution is essential.

6.5 Differentiable Learning of Symbolic Automata

We next proceed to the presentation of our differentiable approach to learning SFA from perceptual sequences, aiming to alleviate the combinatorial complexity of symbolic induction techniques, such as ASAL.

An input sequence is denoted $x_{1:T} = (x_1, \ldots, x_T)$, with each x_t an element of a domain $\mathcal{X}$. The

domain may consist of multivariate time-series vectors ($\mathcal{X} \subseteq \mathbb{R}^d$), images or video frames ($\mathcal{X}$ a space of tensors), or already-symbolic tokens. Our goal is to learn interpretable symbolic automata that recognize complex event patterns over such sequences.

Let Π be a set of base, predicates, i.e. the predicates that will be used as building blocks for synthesizing the SFA guards. Each of these base predicates is either a primitive condition over the input (e.g., `attribute ><= value`) or a higher-level predicate that can be used to reason over the neural output – e.g. consider the `even/1` predicate over MNIST digit predictions by a CNN. In the case where the base predicates are numerical feature/value comparisons over the neural predictions their probabilities are directly given from the neural network. Otherwise, for more complex predicates we use knowledge compilation into probabilistic circuits, which are used to calculate predicates probabilities from the symbol probabilities predicted by the neural network.

Our goal is to use such sequences, as training data, and the predicates as a language bias, in order to learn, in a fully differentiable fashion, symbolic automata that express complex event patterns that are present in the data. Training sequences are labeled as positive or negative. Positive (resp. negative) sequences are those that satisfy (no not satisfy) a target complex event pattern and should be accepted (resp. rejected) by the learnt SFA. We give a brief formal definition of SFA below and refer the reader to the previous sections for further details regarding SFA and their usage in CER applications.

6.5.1 Preliminaries

Symbolic finite automata. Let Π be as above and let $\mathbb{B}(\Pi)$ denote the Boolean algebra generated from Π by conjunction, disjunction, and negation. A *symbolic finite automaton* (SFA) over Π is a tuple

$$\mathcal{A} = (Q, q_0, F, \Pi, \Delta),$$

where $Q = \{1, \ldots, N\}$ is a finite set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of accepting states, and $\Delta \subseteq Q \times \mathbb{B}(\Pi) \times Q$ is a finite set of transitions. Each transition is of the form (u, φ, v), where $\varphi \in \mathbb{B}(\Pi)$ is a *guard*; we write $u \xrightarrow{\varphi} v$ for such a transition. Given a sequence $x_{1:T}$, a run of $\mathcal{A}$ is a state sequence $q_{0:T} = (q_0, \ldots, q_T)$ such that for each $t \in \{1, \ldots, T\}$ there is a guard φ_t with $(q_{t-1}, \varphi_t, q_t) \in \Delta$ and φ_t true on x_t under the Boolean semantics of Π. The run is accepting if $q_T \in F$, and the language $L(\mathcal{A})$ consists of all sequences that admit at least one accepting run.

DNF guards. We fix an upper bound N on the number of states for the target SFA. For simplicity, in what follows we assume that target SFA have a single accepting state, which is also absorbing (i.e. there are no outgoing transitions from the accepting state). Therefore, in principle we want to learn $N^2 - 1$ guards, each represented as a logical formula over the base predicates in Disjunctive Normal Form (DNF), i.e. a disjunction of conjunctions. We assume a maximum number M of clauses (disjuncts) per DNF/guard, which is a hyperparameter. We index guards by $g \in \{1, \ldots, G\}$ with $G = N^2 - 1$, each guard corresponding to a specific ordered pair of states (u_g, v_g).

Guard parametrization. The i-th clause of guard g is a conjunction of literals indexed by j, where a literal is a base predicate π, or its negation $\neg\pi$. We denote by $\pi_g^{i,j}$ the j-th literal in the i-th clause of guard g, and by $p_g^{i,j}(t) \in (0, 1)$ the probability that $\pi_g^{i,j}$ holds at time t. If $\pi_g^{i,j}$ is a positive predicate, then $p_g^{i,j}(t) = p(\pi_g^{i,j} \mid x_t)$, otherwise $p_g^{i,j}(t) = 1 - p(\bar{\pi}_g^{i,j} \mid x_t)$. We therefore maintain $|\Pi| \cdot M \cdot (N^2 - 1)$ real-valued weights specifying the importance of each literal in each clause of each guard – recall that

Π is the base predicate set, M is the maximum number of disjuncts per DNF and N is the maximum number of states in the target SFA.

6.5.2 Conjunction Layer

We split a literal weight $w_g^{i,j} \in \mathbb{R}$ into a positive and a negative part via rectified linear units:

$$w_{g,+}^{i,j} = \max\{w_g^{i,j}, 0\}, \qquad w_{g,-}^{i,j} = \max\{-w_g^{i,j}, 0\}.$$

$w_{g,+}^{i,j}$ measures how strongly the i-th positive literal should be enforced in clause c_j^i i (i-th clause of guard g), whereas $w_{g,-}^{i,j}$ measures how strongly its negation should be enforced; when both are close to zero, the corresponding predicate is effectively irrelevant for that clause.

Given the literal probabilities and split weights, we define the log-product score of clause i of guard g at time t as

$$s_g^i(t) = \sum_j \Big(w_{g,+}^{i,j} \log p_g^{i,j}(t) + w_{g,-}^{i,j} \log\big(1 - p_g^{i,j}(t)\big) \Big), \tag{4}$$

which after exponentiation yields

$$\exp\big(s_g^i(t)\big) = \prod_j \big(p_g^{i,j}(t)\big)^{w_{g,+}^{i,j}} \big(1 - p_g^{i,j}(t)\big)^{w_{g,-}^{i,j}}.$$

This is a *Weighted Product Model* [86] over the positive and negative literal probabilities. When the weights are constrained to $\{0, 1\}$ and at most one of $w_{g,+}^{i,j}, w_{g,-}^{i,j}$ is 1 for each j, this product reduces to the probability that all literals with weight 1 are simultaneously true (under a standard independence assumption), i.e., the probability that a crisp conjunction is satisfied. Allowing real-valued weights turns this into a *soft* conjunction.

To pass clause scores to the disjunction layer (which, as discussed below, is a noisy-OR layer, expecting values in $[0, 1]$), we map $s_g^i(t) \in (-\infty, 0]$ to a truth value in $[0, 1]$ via a sigmoid σ:

$$p_g^i(t) = \sigma\big(s_g^i(t)\big) = \frac{1}{1 + \exp\big(-s_g^i(t)\big)}. \tag{5}$$

$p_g^i(t) \in (0, 1)$ serves as a fuzzy truth value for the fact that the i-th clause of guard g is satisfied at time t: if all high-weight positive literals have high truth values $p_g^{i,j}(t)$ and all high-weight negative literals have low truth values then $p_g^i(t)$ from (5) is close to 1. On the other hand, $p_g^i(t)$ is close to 0 if the clause contains "inconsistent" literals, i.e. at least one high-weight positive literal with small truth value $p_g^{i,j}(t)$, or a high-weight negative literal with a large $p_g^{i,j}(t)$. In such cases, these "inconsistent" literals push the clause's log-product score to a large negative number, so that the sigmoid is close to zero. Note also that a clause's after-sigmoid truth value is close to $1/2$ when all its literals' weights are close to zero, so that it log-product is close to zero. This represents a "neutral", degenerate clause that is effectively independent of the input will most likely be pruned post-training (see below for weight pruning).

6.5.3 Disjunction Layer

At the guard level we introduce an additional parameter $\gamma_g^i \in \mathbb{R}$ per clause, and define a clause weight as $\alpha_g^i = \sigma(\gamma_g^i) \in (0, 1)$. Intuitively, α_g^i expresses how much clause i is trusted, or how "active" it is allowed to be inside the guard: values of α_g^i close to zero effectively deactivate the clause, while values close to one allow the clause to contribute to the DNF. The guard probability at time t is then obtained

by a weighted noisy-OR over the clause values:

$$g_g(t) = 1 \; - \; \prod_{i=1}^{R} \big(1 - \alpha_g^i \, q_g^i(t)\big). \tag{6}$$

If we interpret $q_g^i(t)$ as the probability that clause i is satisfied at time t by input $f(x_t)$ (where by f we denote the perception network that maps percepts to symbols) and α_g^i as the probability that clause i is actually *used* by the guard, then $\alpha_g^i q_g^i(t)$ is the probability that clause i both fires and is active. Under an independence approximation, the probability that no active clause fires is $\prod_i (1 - \alpha_g^i q_g^i(t))$, and the complement in (6) is the probability that at least one active clause is satisfied. Noisy-or is a classical soft OR approximation from fuzzy logic and the dual t-conorm of the product t-norm [93], whose log-product variant we use here to model soft conjunctions.

Although it is in principle possible to omit the clause weights α_g^i from the architecture, they provide a second level of sparsity on top of the literal weights and, empirically, training has been proven to be much more effective when using clause-level weights, as compared to relying on literal weights only. During training we regularize the α_g^i (e.g., via L_1 or L_0-style [63] penalties) so that redundant clauses are driven towards $\alpha_g^i \approx 0$. Such clauses become effectively inactive in the noisy-or layer and can later be pruned when extracting a discrete SFA.

6.5.4 Transition Layer and the SFA Forward Pass

Each guard g corresponds to a unique ordered pair of states (u_g, v_g). At time t, we use the guard truth values to construct an *unnormalised* transition matrix. For each source state u and destination state v we define a score

$$r_{u,v}(t) = \frac{g_{g(u,v)}(t)}{\tau},$$

where $g(u, v)$ is the index of the guard for transition $u \to v$, and $\tau > 0$ is a temperature parameter. We then obtain a row-stochastic transition matrix from u at time t via a softmax:

$$p_t(v \mid u) = \frac{\exp\big(r_{u,v}(t)\big)}{\sum_{v'} \exp\big(r_{u,v'}(t)\big)}. \tag{7}$$

For each fixed u and t, the mapping $v \mapsto p_t(v \mid u)$ is a categorical distribution over states. The temperature τ controls how deterministic the transitions are: small τ makes the distribution concentrate on the transition with the largest guard probability, while larger τ yields smoother, more exploratory dynamics. In practice, during training, we typically start with a high temperature and gradually anneal it, encouraging the model to converge towards near-deterministic automaton behavior.

Given the time-varying transition probabilities $p_t(\cdot \mid \cdot)$, the sequence-level semantics is defined by the classical dynamic programming-based recursion from temporal graphical models (e.g. HMMs) – se also Section 4.3.1. Let μ be an initial distribution over states and define $\alpha_0(u) = \mu(u)$. For $t \geq 1$ we propagate state probabilities according to

$$\alpha_t(v) = \sum_u \alpha_{t-1}(u) \, p_t(v \mid u). \tag{8}$$

This is the usual forward recursion for a non-homogeneous Markov chain whose transition matrix at

time t is $[p_t(v \mid u)]_{u,v}$[4]. Let $F \subseteq Q$ be the set of accepting states. The acceptance probability of $x_{1:T}$ under the relaxed δSFA semantics is

$$p_\Theta(\text{acc} \mid x_{1:T}) = \sum_{v \in F} \alpha_T(v), \tag{9}$$

where Θ collects all parameters of the model (literal and clause weights).

Additional inductive biases. As already mentioned, for simplicity we constrain the SFA hypothesis space to the space of symbolic automata of up to N states, with a single, absorbing accepting state and a single start state, as well as up to M disjunct rules per DNF guard. The accepting state q_{acc} being absorbing means that it has no outgoing transitions, but only a deterministic self-loop (i.e. once the SFA reaches q_{acc} it self loops on it with probability 1.0, regardless of the input). Additionally, we refrain from learning explicit guards for self-loop transitions and instead treat them as probabilistic complements of any state's $q \in Q \setminus q_{acc}$ outgoing transitions, thus treating self-loops as realizing the "skip-till-next-match" event consumption policy [57]. Specifically, during the construction of the transition matrix at each step, we set the truth value of any non-accepting self-loop entry q to $1-\vee_q$, where $\vee_q$ is the noisy-OR truth value of q's outgoing guards, and then normalize the rows of the transition matrix via a tempered softmax. This way we encode the requirement that any piece of input that does not satisfy one of the outgoing transition guards is effectively "skipped", without having to learn explicit formulas for the guards' logical complements.

6.5.5 Weight Pruning and Post-Training SFA Extraction

After training converges we extract a crisp SFA via weight pruning and thresholding on the neural model. First, we set to zero the weights of all SFA guards whose removal does not affect the training loss more than a given threshold ϵ. We then repeat this process for the literals of the remaining guards, zeroing out the weights of all literals whose removal from the model does not significantly affect the performance on the training set. Next, we select a weight threshold for surviving literals by sweeping over a range of possible weight values, using a similar training performance reduction criterion and we remove the literals whose weight is below is below that threshold. Finally We construct a crisp SFA from the guards and literals that survive this multi-step pruning process.

Although the weight pruning and thresholding process often yields a high-quality crisp SFA, there is no guarantee for that. It might be the case that the extracted crisp SFA contains redundant literals, which were deemed useful for the neural SFA during the pruning process, but are useless, or even hurtful for the crisp SFA's performance, resulting in over-specific guards, which affect the symbolic model's generalization abilities. Moreover, although the training process pushes towards deterministic SFA via the tempered softmax that is used to convert the unnormalized transitions matrices to row-stochastic ones during training, there is no guarantee that the crisp SFA that is extracted from the neural model is indeed deterministic: it can happen in practice that the DNFs that guard the outgoing transitions from some state q in the crisp SFA are not mutually exclusive, resulting in a non-deterministic SFA. This hurts interpretability of the resulting model, as well as the ability to combine it with NeSy reasoning frameworks that expect deterministic SFA, such as NeSyA. Although a non-deterministic SFA can be made deterministic via standard determinization algorithms, this requires propositionalizing the

[4]For numerical stability on long sequences, the recursion can be implemented in log-space by propagating $\log \alpha_t(v)$ and using log-sum-exp in place of the sums, while keeping the transition logits in the log domain as well.

SFA, which hurts interpretability even further, while it can also result in an exponential blow up in the automaton's size (number of states and transitions).

To address these issues we utilize an ASP-based approach for selecting the optimal combination of guards and literals from those that survive the pruning process, while enforcing structural constraints that ensure determinism, or other properties that might be desirable for simplifying the model (e.g. explicit dead/reject states in the SFA, inclusion of backward transitions, or "jumps" in its graph structure and so on).

Specifically, we treat the surviving guards and literals as a hypothesis space for ASAL and utilize its support for easily definable, declarative constraints and its abductive learning methodology for identifying the optimal crisp structure that best fits the training data. Crucially, the heavily constrained hypothesis space that results from neural training makes the complexity of the SFA induction process manageable. This is in contrast to plain ASAL, where an SFA needs to be learned from scratch, from an unconstrained hypothesis space that grows exponentially with the dimensionality and length of the input.

We omit the description of the post-training SFA-induction meta-encoding and instead refer to [57] and D4.1, which contain detailed descriptions of such encodings and of declarative specifications of structural constraints that are enforced during the induction process.

Example 1. Consider the following rules that were learned from the ≈ 1500 length-50, multivariate gene sequences in EVENFLOW's Personalized Medicine use case – we refer to D3.3 for further details on the use case:

0.589: $g(1,2) \leftarrow -0.233$: $\texttt{hus1b}(0), -0.018$: $\texttt{slc22a1}(0), -0.017$: $\texttt{oasl}(1), -0.016$: $\texttt{oasl}(0)$,
0.616: $g(2,1) \leftarrow -0.043$: $\texttt{hus1b}(2), +0.040$: $\texttt{hus1b}(0), -0.033$: $\texttt{oasl}(2), -0.029$: $\texttt{oasl}(1), -0.016$: $\texttt{slc22a1}(2)$.
0.506: $g(2,3) \leftarrow -0.175$: $\texttt{hus1b}(0), -0.030$: $\texttt{slc22a1}(0), -0.020$: $\texttt{slc22a1}(2), -0.018$: $\texttt{slc22a1}(1), -0.016$: $\texttt{oasl}(0)$.
0.580: $g(3,1) \leftarrow -0.115$: $\texttt{hus1b}(2), -0.110$: $\texttt{hus1b}(1), -0.024$: $\texttt{slc22a1}(1), -0.014$: $\texttt{slc22a1}(0)$,
0.127: $g(3,2) \leftarrow -0.044$: $\texttt{hus1b}(1), -0.035$: $\texttt{hus1b}(2), -0.031$: $\texttt{oasl}(2), -0.027$: $\texttt{slc22a1}(1), -0.020$: $\texttt{slc22a1}(2)$.
0.500: $g(3,4) \leftarrow -0.185$: $\texttt{hus1b}(0), -0.033$: $\texttt{oasl}(0), -0.030$: $\texttt{slc22a1}(0), -0.028$: $\texttt{hus1b}(1), -0.015$: $\texttt{oasl}(2)$.
0.537: $g(4,1) \leftarrow -0.150$: $\texttt{hus1b}(1), -0.126$: $\texttt{hus1b}(2), -0.024$: $\texttt{oasl}(1)$.
0.161: $g(4,2) \leftarrow -0.081$: $\texttt{hus1b}(1), -0.031$: $\texttt{oasl}(1), -0.022$: $\texttt{slc22a1}(0), -0.021$: $\texttt{oasl}(0), -0.017$: $\texttt{hus1b}(2)$.
0.140: $g(4,3) \leftarrow -0.067$: $\texttt{hus1b}(1), -0.033$: $\texttt{hus1b}(0), -0.026$: $\texttt{hus1b}(2), -0.018$: $\texttt{oasl}(0), -0.016$: $\texttt{slc22a1}(2)$.
0.456: $g(4,5) \leftarrow -0.178$: $\texttt{hus1b}(0), -0.037$: $\texttt{hus1b}(1), -0.017$: $\texttt{oasl}(0), -0.017$: $\texttt{slc22a1}(0)$.

These rules represent the post-weight pruning surviving guards of a 5-state SFA learned from the gene sequences dataset, using the top-ten genes from the panel of fifty genes that were identified as most relevant to cancer progression in this use case. Therefore, we are learning from 10-variate sequences in this case. We omit self-loops since, as mentioned above, we are not explicitly learning DNF formulas for them. The head of each rule refers to the transition that the rule guards in the learned SFA. For instance the first rule refers to the transition $(1 \rightarrow 2)$. The learned weight of each rule precedes the head, as in 0.589: $g(1,2)$. The bodies of the rules represent conjunctions of literals. The input training sequences were discretized to three bins for this experiment, therefore, each gene can take three values, 0, 1 and 2. The predicates in the bodies of the rules encode the values that each gene takes, for instance, $\texttt{hus1b}(2)$ means that gene $\texttt{hus1b}$ takes the value of 2. The sign encodes the "polarity" of the predicate, i.e. whether the predicate is used positively or negatively in a rule. The learned weights of each predicate are also presented before each predicate. Recall that these are weights that survive the weight

pruning process.

Given the above, the weighted rules correspond to the following crisp rules obtained by discarding the weights and properly interpreting the literals' signs:

$$
\begin{aligned}
g(1,2) &\leftarrow \mathit{not}\ \texttt{hus1b}(0), \mathit{not}\ \texttt{slc22a1}(0), \mathit{not}\ \texttt{oasl}(1), \mathit{not}\ \texttt{oasl}(0), \mathit{not}\ \texttt{hus1b}(1).\\
g(2,1) &\leftarrow \mathit{not}\ \texttt{hus1b}(2), \texttt{hus1b}(0), \mathit{not}\ \texttt{oasl}(2), \mathit{not}\ \texttt{oasl}(1), \mathit{not}\ \texttt{slc22a1}(2).\\
g(2,3) &\leftarrow \mathit{not}\ \texttt{hus1b}(0), \mathit{not}\ \texttt{slc22a1}(0), \mathit{not}\ \texttt{slc22a1}(2), \mathit{not}\ \texttt{slc22a1}(1), \mathit{not}\ \texttt{oasl}(0).\\
g(3,1) &\leftarrow \mathit{not}\ \texttt{hus1b}(2), \mathit{not}\ \texttt{hus1b}(1), \mathit{not}\ \texttt{slc22a1}(1), \mathit{not}\ \texttt{slc22a1}(0).\\
g(3,2) &\leftarrow \mathit{not}\ \texttt{hus1b}(1), \mathit{not}\ \texttt{hus1b}(2), \mathit{not}\ \texttt{oasl}(2), \mathit{not}\ \texttt{slc22a1}(1), \mathit{not}\ \texttt{slc22a1}(2).\\
g(3,4) &\leftarrow \mathit{not}\ \texttt{hus1b}(0), \mathit{not}\ \texttt{oasl}(0), \mathit{not}\ \texttt{slc22a1}(0), \mathit{not}\ \texttt{hus1b}(1), \mathit{not}\ \texttt{oasl}(2).\\
g(4,1) &\leftarrow \mathit{not}\ \texttt{hus1b}(1), \mathit{not}\ \texttt{hus1b}(2), \mathit{not}\ \texttt{oasl}(1).\\
g(4,2) &\leftarrow \mathit{not}\ \texttt{hus1b}(1), \mathit{not}\ \texttt{oasl}(1), \mathit{not}\ \texttt{slc22a1}(0), \mathit{not}\ \texttt{oasl}(0), \mathit{not}\ \texttt{hus1b}(2).\\
g(4,3) &\leftarrow \mathit{not}\ \texttt{hus1b}(1), \mathit{not}\ \texttt{hus1b}(0), \mathit{not}\ \texttt{hus1b}(2), \mathit{not}\ \texttt{oasl}(0), \mathit{not}\ \texttt{slc22a1}(2).\\
g(4,5) &\leftarrow \mathit{not}\ \texttt{hus1b}(0), \mathit{not}\ \texttt{hus1b}(1), \mathit{not}\ \texttt{oasl}(0), \mathit{not}\ \texttt{slc22a1}(0).
\end{aligned}
$$

In addition to the fact that a careful inspection of these rules reveals several redundancies, the crisp SFA has a very low test-set performance. This is in contrast to the neural model that uses the weighted rules presented above, which achieves an almost perfect test-set F_1-score. This is indicative of the fact that simply keeping all literals that survive weight pruning in the crisp SFA, may result in over-specific guards that generalize poorly. Attempting to improve the SFA manually, by removing additional, low-weight literals is tedious and error-prone, often yielding non-deterministic outcomes.

These issues are resolved in principled fashion by treating the crisp SFA presented above as the hypothesis space for the induction of a minimal SFA that subsumes the crisp SFA above and best fits the training sequences. The latter means that the crisp guards and their literals are written as a "defeasible" ASP program, which is passed to ASAL alongside the symbolic training sequences and structural constraints regarding the desirable properties that the SFA should have (e.g. determinism). The final, crisp SFA process that is induced from this process is presented below (the absorbing $g(5,5)$ rule refers to the accepting state, which is state 5):

$$
\begin{aligned}
g(1,1) &\leftarrow \mathit{not}\ g(1,2).\\
g(1,2) &\leftarrow \mathit{not}\ \texttt{hus1b}(0).\\
g(2,3) &\leftarrow \mathit{not}\ \texttt{hus1b}(0).\\
g(3,1) &\leftarrow \texttt{hus1b}(0).\\
g(3,4) &\leftarrow \mathit{not}\ \texttt{hus1b}(0).\\
g(4,1) &\leftarrow \texttt{hus1b}(0).\\
g(4,5) &\leftarrow \mathit{not}\ \texttt{hus1b}(0).\\
g(5,5) &\leftarrow \texttt{\#true}.
\end{aligned}
$$

Notably, this SFA achieves a test-set F_1-score of 0.96, comparable to that of the neural model. Moreover, it was induced by ASAL using the constrained search space defined by the learned neural SFA in approximately half a minute from the entire 10-variate training set. Attempting to learn this SFA from scratch with ASAL, from the same training set failed after exhausting 30GB of RAM. For ASAL to be able to learn in this domain we had to reduce the dimensionality of the dataset, keeping only the three most promising genes from the 50 genes panel. moreover, we had to increase the cut-off for the allowed induction time for ASAL to three hours. On the other hand, by using the incremental version of ASAL with batches of 100 sequences, we were able to learn with a reasonable time and

memory budget, albeit models of significantly inferior quality, that did exceed a test-set F_1-score of 0.80. In contrast, the combination of differentiable SFA learning plus ASAL-based rectification learned an almost perfect model in less than 5 minutes in total.

6.6 Experimental Evaluation

In this section we present an experimental evaluation of our new method, to which we henceforth refer as ∂SFA, on both time series and image sequence datasets. The goal was to empirically validate in a range of applications and perceptual input data modalities that ∂SFA is indeed capable of scaling much better than purely symbolic techniques (ASAL) without compromizing the predictive performance of the learned models.

Although in principle ∂SFA can be used for end-to-end training that allows to learn an SFA while simultaneously training a perception network to map percepts to symbols, we leave this challenging setting for future work and focus here on the simpler setting where the perception network is pre-trained beforehand and we use its predictions on the percepts to infer the target SFA with ∂SFA. For the time series datasets we discretized the time series to a number of symbols using the SAX algorithm and during training we pass to ∂SFA near one-hot encodings of the symbols, simulating a near-perfectly trained network (e.g. an MLP) that predicts these symbols from the time series data. We next present in detail the datasets used, experimental settings and the obtained results for the various domains in our experimental study.

6.6.1 Temporal MNIST

The purpose of this experiment was to stress-test our structure learning methods to perceptual input sequences of different lengths and to target SFA of different complexity, in a principled and fully controlled fashion. To that end, we used a significantly more complex version of the simple MNIST-based domain, introduced in 6.4. The training data in the Temporal MNIST dataset consist of multivariate sequences of MNIST digit images. Each point in a sequence consists of three digit images (therefore, the dataset is 3-variate), each of which is mapped to a probability distribution over the digits $0-9$ via a convolutional neural network (CNN). In particular, given the three input images $\langle x_1^t, x_2^t, x_3^t \rangle$ at point t in an input sequence, the CNN predicts three probability vectors $v_i^t = \langle p(d_i = 0 \mid x_i^t), \ldots, p(d_i = 9 \mid x_i^t) \rangle$ for $i = 1, 2, 3$ and with d_i referring to the i-th digit.

The sequences in these datasets are labeled as positive or negative, based on whether they satisfy a target SFA or not. The target SFA patterns were defined in terms of the $even/odd$ predicates and express sequential conditions that correspond to complex events that occur, or do not occur in the image sequences. For instance, the following guard definition rules:

$$
\begin{aligned}
g(1,1) &\leftarrow not\ g(1,2).\\
g(1,2) &\leftarrow even(d_1), odd(d_2), odd(d_3).\\
g(1,2) &\leftarrow even(d_3).\\
g(2,2) &\leftarrow not\ g(2,3).\\
g(2,3) &\leftarrow odd(d_1), even(d_2).\\
g(2,3) &\leftarrow even(d_1), even(d_3).\\
g(3,3) &\leftarrow not\ g(3,4).\\
g(3,4) &\leftarrow odd(d_3).\\
g(4,4) &\leftarrow \texttt{\#true}.
\end{aligned}
$$

define a 4-state SFA pattern dictating that a positive sequence should satisfy the following sequential conditions: a point in the sequence where either $even(d_1) \wedge odd(d_2) \wedge odd(d_3)$, or $even(d_3)$ are true should be observed, followed by a point where either $odd(d_1) \wedge even(d_2)$, or $odd(d_1) \wedge even(d_2)$ are true, eventually followed by a point where $odd(d_3)$ is true. Self-loop guards dictate that irrelevant sequence points in-between the transition points should be skipped. Negative sequences are sequences that do not satisfy the pattern.

We defined the patterns as Answer Set programs and we used ASAL to abductively generate multivariate digit sequences that satisfy (resp. do not satisfy) the patterns, using acceptance constraints to guide the abductive search. We also used a dedicated ASP meta-encoding of Hamming distance-based diversity between the sequence-representing stable models returned by the solver in ASAL, in order to ensure that the generated sequences are adequately diverse and to avoid repetitions.

Once a symbolic sequence dataset was generated, we paired it with MNIST images, selecting an image with the same label as the corresponding digit in the symbolic sequences. The images required for this pairing well exceed the size of the original MNIST dataset. Therefore, to avoid data leakages due to repeated images in the training and testing sets, once the original MNIST dataset was exhausted during the generation process, we applied simple transformations to each image selected from that point onward, to ensure that each image in the generated sequences is unique.

We used two target SFA patterns with five and ten states respectively. Each guard in these patterns consists of up to 3 disjuncts, with each disjunct consisting of up to four conjunctions. For each of the two patterns we generated datasets with sequence lengths of 10, 30 and 50, for a total of six multivariate image sequence datasets.

We compare ∂SFA to ASAL, in both its batch and incremental versions. In its batch version, ASAL needs to induce an SFA from the entirety of the training sequences. Given enough resources (time and memory) this version is guaranteed to learn an optimal SFA with the best trade-off between predictive performance and model complexity. However, it is often infeasible to scale this version to input data of increased length and dimensionality, since the induction process scales exponentially in those quantities. To account for that that, ASAL supports an incremental version, which learns partial SFA from data mini-batches and iteratively refines them over the data in a Monte Carlo Tree Search (MCTS)-based approach, trying to approximate a global optimum from partial data views. Given moderate mini batch sizes, ASAL's incremental version scales much better than the batch version, but it can often yield models of inferior quality, especially in complex, or noisy domains, where partial views are insufficient for a good global optimum approximation.

In all experiments we used a standard-for-MNIST CNN architecture, pre-trained for 100 epochs on 1K images sampled randomly from the image sequences, achieving a test-set F_1-score for digit recognition that varied between 0.89 and 0.92 across the various runs. In our experiment ∂SFA learned directly from the CNN's predictions, i.e. we define an integrated neural architecture where each input point in a training sequence is first input to the CNN, its predicted probability distributions are passed to ∂SFA and the entire sequence is processed via ∂SFA's forward recursion algorithm that returns an acceptance probability at the end of the sequence. For ASAL the training setting was different, resembling the one outlined in Section 6.3: the CNN was used to make argmax predictions from the image sequences; each inferred symbolic sequence was weighted by its average entropy across the sequences and ASAL used these weights as an extra bias during SFA induction, encoding a preference towards low-entropy sequences. Note that although this setting allows ASAL to simulate learning from neural outputs, it is extremely challenging in terms of scalability, since it introduces and additional

			∂SFA		∂SFA+ASP		ASAL		ASAL$_{inc}$	
Pattern	$\|Q\|$	T	F_1	Time	F_1	Time	F_1	Time	F_1	Time
Pattern A	5	10	1.0	0.7	1.0	0.9	1.0	3.8	0.955	4.7
		30	1.0	1.2	1.0	2.3	1.0	7.3	0.702	4.9
		50	1.0	1.8	1.0	3.5	0.948	10 (time-out)	0.723	5.3
Pattern B	10	10	1.0	1.4	1.0	2.2	0.913	10 (time-out)	0.718	8.23
		30	1.0	2.6	1.0	3.3	0.887	10 (time-out)	0.725	8.82
		50	1.0	3.8	1.0	5.2	0.687	10 (time-out)	0.620	12.4

Table 10: Test F_1-score and training time (**in minutes**) for the two target SFAs with different numbers of states $|Q|$ and datasets with different sequence lengths T.

WeightedMaxSAT optimization objective. It also makes incremental ASAL less appropriate, since global access to the weighted sequences is typically required in order to learn a good approximation to a global optimum.

We used 70/30 train/test splits on 2K positive and negative sequences in each one of the six datasets and the presented results are averages from a 3-fold cross-validation. ∂SFA was trained for 100 epochs in each run. For ASAL we used a cut-off induction time of 10 minutes. We run ASAL$_{inc}$ on mini-batches of 100 sequences for 20 MCTS iterations and a cut-off induction time per iteration of 1 minute.

Table 10 presents the results from the MNIST experiment in terms of F_1-scores and training times for the methods compared. As can be seen from Table 10, both ∂SFA and ∂SFA+ASP are able to correctly recover the target SFA in all runs, achieving a perfect test-set F_1-score. Moreover, the extra overhead in induction time that stems from searching for an optimal crisp SFA using a neural SFA as a starting point (∂SFA+ASP) is tolerable in general and does not exceed an additional 2 minutes on top of neural training. In contrast, ASAL exceeds the cut-off induction time of 10 minutes in most runs and the model returned within this time is sub-optimal, as can be seen by the imperfect test-set F_1-scores. Moreover, ASAL's predictive performance deteriorates rapidly in the more challenging settings (larger number of states in the target SFA, larger sequence lengths), indicating that ASAL requires significantly more time to learn a high-quality SFA, or adequately approximate the optimum in such settings. The incremental version of ASAL ends-up performing even worse. This is mainly due to the relatively small batch size that we used for ASAL$_{inc}$ in this experiment, which incurs an increased "myopia", caused by locally optimal solutions from which the system cannot effectively recover. ASAL$_{inc}$ can achieve significantly better results in this experiment with a larger mini-batch size, at the cost of increased learning times that approximate those of the batch version.

In contrast to the above ∂SFA can learn a perfect neural model in significantly less time and, when coupled with ASP-based search, a perfect, fully interpretable crisp model at the cost of a small increase in total training times. It is worth mentioning that ASAL has a very large memory footprint, requiring ≈13GB of RAM in this experiment, in contrast to the neural method that has no significant memory requirements. These results demonstrate empirically the efficacy of our new approach for differentiable SFA learning.

6.6.2 ROAD-R

Our second experiment is again from a vision-based domain and its purpose is to evaluate the efficacy of our approach on a real-world event recognition application. To that end, we used the ROAD-R

dataset, which was discussed in Section 5, with the purpose of learning SFA-based definitions for *overtake* incidents. The difference from the work in Section 5 is that there we had a crisp SFA as background knowledge, which was learned separately by ASAL from purely symbolic sequences. The goal was to train a CNN to detect simple driving events with indirect supervision using NeSyA (Section 4), with the supervision provided in terms of *overtake* ground truth only, at the sequence level. Here, we instead use a pre-trained CNN, trained on the entirety of the ROAD-R dataset, and use its predictions to train ∂SFA, so that it learns a high-quality SFA for the overtake complex event.

Other than that, the experimental setting is identical to that of Section 4. We limit ourselves to the small simple event library introduce there, namely the events *move_away, move_towards, stop, other* for the vehicles' actions and *vehicle_lane, incoming_lane, junction, other* for their locations. The training data for ∂SFA are the positive and negative *overtake* image sequences that were extracted in Section 5 with sequence lengths that range between 6 and 10. We perform experiments on the four training/test splits described there.

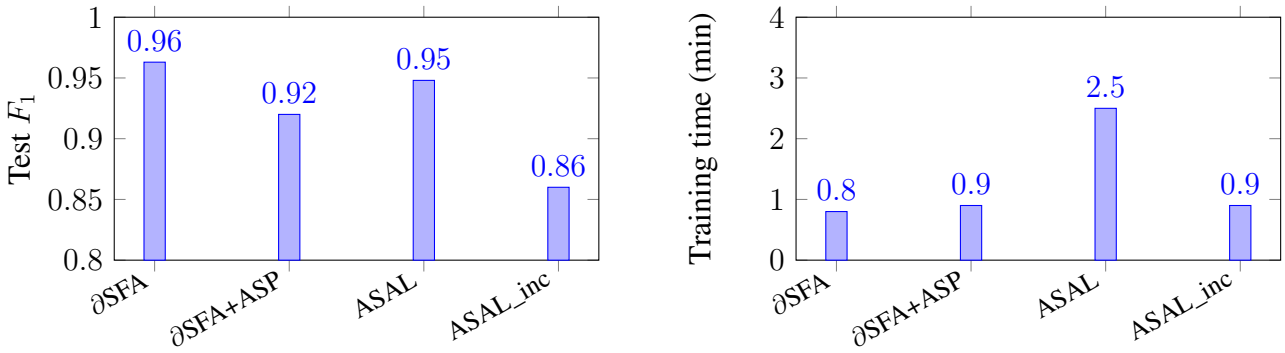


Figure 14: Comparison of test F_1-score (left) and training time (right) for SFA learning on the ROAD-R dataset (learning *overtake* definitions).

The results are presented in Figure 14 in terms of F_1-scores and training times for the ∂FA and ASAL variants compared in the previous experiments. We observe that the neural SFA learned by ∂SFA achieves the best performance in less than one minute (trained for 100 epochs), while discretization via ASP search slightly exacerbates this performance without significant training time overheads. ASAL is able to nearly match ∂SFA's performance, but requires significantly larger training time. Its incremental version is able to discover an SFA of significantly inferior quality in training times that match the neural ones.

6.6.3 Evaluation on EVENFLOW Data.

In Example 1 we already provided an overview of learning from multi-variate time series data with ∂SFA from an EVENFLOW dataset, in particular, the gene expression level dataset for cancer progression (Personalized Medicine use case). In this section we provide additional technical details for this experiment and we also present a similar evaluation on the Industry 4.0 use case.

Personalized Medicine. As mentioned in Example 1, the purpose of this experiment was to learn an SFA that captures cancer progression in terms of the change over time in the expression levels of critical genes. The purpose SFA learning in this use case was to infer a high-quality, logic-based temporal model, i.e. an SFA, that can subsequently be used for interpretable complex event forecasting with Wayeb [3], our baseline interpretable event forecaster in EVENFLOW. The results from this

experimental study are presented in Deliverable D3.2, where ASAL was used to learn the background knowledge SFA. Here, we attempt to apply ∂SFA on the same task, in order to speed-up the SFA induction process.

Deliverable D3.3 provides a detailed overview of the core dataset RNA-sequencing gene expression profiles dataset that was used by BSC to train the Variational Autoencoder that was used in EVEN-FLOW, in order to generate synthetic patients' trajectories, in an effort to compensate for the lack of real-patient trajectory data. Here, a trajectory refers to a multi-variate time series of gene expressions that starts from an early state cancer profile and either progresses to a late stage, or remains at early stage. The former case, which represents a positive sample/sequence in our learning setting, attempts to capture a deterioration in the patient's condition. The latter case, corresponding to a negative sample (no progression), captures a desirable outcome that could be co-related to specific medical treatments in clinical trials. In addition, Deliverable D3.3 provides details on the actual trajectory generation via patient pairing, trajectory extrapolation in the VAE's latent space and the methodology that was followed for generating negative trajectories.

The trajectory generation process resulted in a dataset of 1494 positive and 264 negative trajectories, split into training and testing sets using an 80/20 split ratio and stratified sampling. The high class imbalance, partly due to measures against data leakage in the testing set, which was in turn due to the particularities of patient pairing for trajectory generation, called for weighted learning schemes, both in purely symbolic (ASAL) and NeSy (∂SFA) learning.

SFA learning was based on discretized trajectories. That is, the time series data was mapped into a set of symbols and the input to the structure learning algorithms was either the corresponding symbolic sequences (ASAL), or, near-one-hot probability vectors (∂SFA), simulating the predictions of a nearly perfectly trained neural network that maps time series signals to symbols. We used two methods to discretize the trajectories: (a) SAX, a specialized discretization algorithm, designed specifically for time series data, and (b) a standard k-bins discretization process. The reason for working with k-bins was that the SAX discretization process, while perfectly OK for sequence classification tasks, introduces phenomena that can be interpreted as data leakages in *early sequence classification, or forecasting tasks*. The results presented here are with k-bins discretization.

We discretized the gene expression level values into three bins. This was deemed adequate, since this level of symbol granularity already produced adequate results with baseline sequence classification (e.g. LSTM) and patient classification (e.g. XGBoost + SHAP for feature importance) techniques – see Deliverable D3.3 for further details.

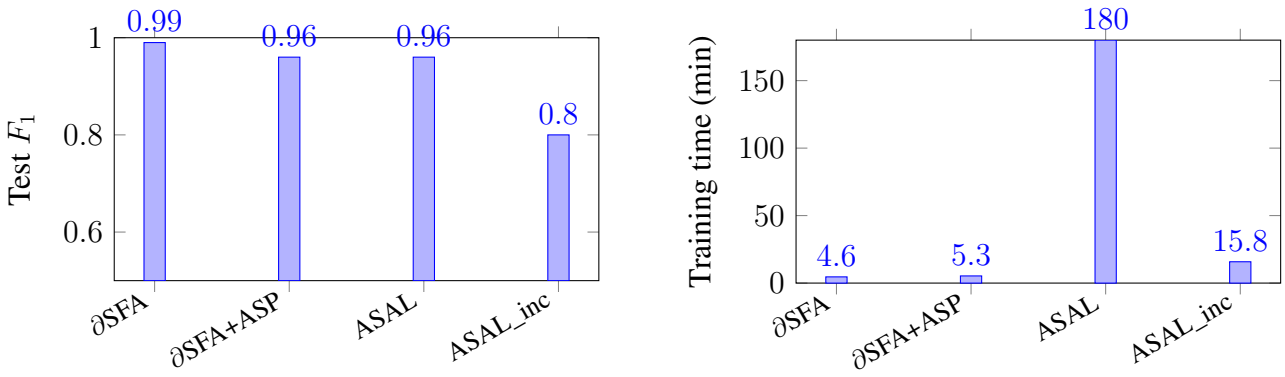
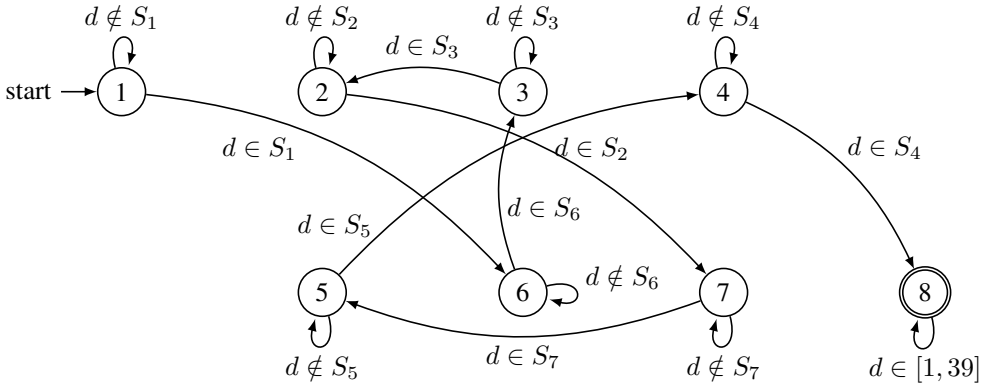


Figure 15: Comparison of test F_1-score (left) and training time (right) for SFA learning on the cancer progression dataset (Personalized Medicine use case).



$S_1 = [4,5] \cup [7,10] \cup \{14\} \cup \{18\} \cup [22,25] \cup [27,28] \cup \{30\} \cup \{34\} \cup [36,38]$
$S_2 = [4,5] \cup [7,8] \cup \{10\} \cup \{12\} \cup [14,17] \cup [19,22] \cup [24,26] \cup [28,31] \cup [33,34] \cup \{36\} \cup \{38\}$
$S_3 = \{4\} \cup \{7\} \cup \{9\} \cup \{11\} \cup \{13\} \cup \{15\} \cup [24,25] \cup \{29\} \cup \{31\} \cup \{34\} \cup \{37\}$
$S_4 = [4,5]$
$S_5 = [4,5] \cup \{9\} \cup \{19\} \cup \{26\} \cup \{30\} \cup [33,35] \cup [37,38]$
$S_6 = [4,5] \cup \{7\} \cup [9,10] \cup \{12\} \cup \{15\} \cup [23,24] \cup \{26\} \cup \{31\} \cup [33,34] \cup \{36\} \cup \{39\}$
$S_7 = \{4\} \cup \{7\} \cup [9,10] \cup [16,19] \cup [22,23] \cup \{29\} \cup [33,35].$

Figure 16: Learned automaton structure for robots' deadlock prediction. d denotes the robots' Euclidean distance, discretized into 40 symbols. Although two features were used (Euclidean distance and robot velocity), the latter was dropped during the induction process, since the formed suffices for a high-quality model.

Figure 15 presents the results. ∂SFA learns an almost perfect SFA in less than 5 minutes, while the crisp SFA extracted with ASP has a slightly inferior predictive performance at a negligible additional training time overhead. ASAL is able to match this performance, but a three-hours induction time cut-off (smaller cut-offs yielded inferior results). Its incremental version, run on mini-batches of 100 sequences, learns a model of significantly inferior quality.

The learned SFA is presented in Example 1. The results from using this SFA for event forecasting with Wayeb in the Personalized Medicine use case are presented in Deliverable D3.3.

Industry 4.0. Similarly to Personalized Medicine, the purpose of automata learning in this use case was to infer logic-based temporal models, in order to use them for interpretable robot deadlock forecasting with Wayeb.

The data that the symbolic model was inferred from consists of robot trajectories in the form of multi-variate time series for a number of attributes, such as robots' (x, y, z) coordinates over time, velocity and orientation vectors and so on. We refer to Deliverables D3.1 and D4.2 for the complete list of attributes and the details of the trajectory generation process via simulations. Our training data consisted of 100 trajectories with an average length of approximately $6.5K$ points each.

In these trajectories the robots move around the smart factory space simulating the execution of a set of pre-defined tasks, where each task requires visiting a number of work stations in a fixed order, to which we refer to as a "plan". The plan-based joint movement of the robots ensures that their motion is not random, allowing for joint mobility patterns to emerge. The robot trajectories are annotated (by DFKI) with deadlock labels. Typically, a deadlock incident occurs when the relative motion of the robots makes them perceive each other as obstacles, which are not easy to avoid via early motion re-planning. Therefore, in almost all deadlock incidents in the robot trajectories datasets a relatively

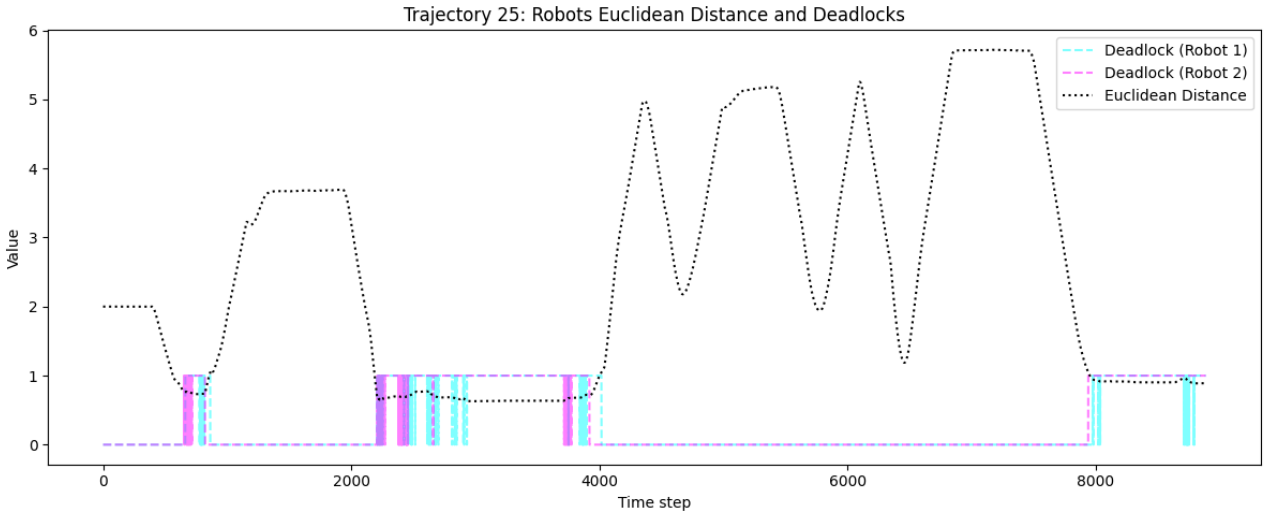


Figure 17: Illustration of the correlation between robot proximity and deadlock incidents in the DFKI dataset.

close Euclidean distance d between the robots is strongly correlated to the occurrence of deadlocks, as illustrated in Figure 17, making d a highly informative feature for deadlock prediction. The second most informative feature is the robots' velocity, since, by definition, in a deadlock incident at least one robot is stopped, or, its non-zero velocity is the rotational one (the robot is spinning, trying to exit the deadlock). In most cases a simple rule that combines the robots' distance and their speed suffices to achieve an adequate performance on deadlock prediction. However, we are not interested in static rules, but in temporal mobility patterns, so that we can forecast their completion from early signs and infer deadlock incidents before they actually occur.

We therefore attempted to learn symbolic automata that correlate the robots' proximity and velocities in their guards. To that end we worked as follows: we first split the trajectories into segments of 500 time points each. We then used SAX to further reduce the dimensionality of these segments and discretize them, resulting in sequences of length 20 time points over an alphabet of 40 symbols (bins) for both features (distance, velocity). We arrived at these numbers via preliminary, exploratory experiments.

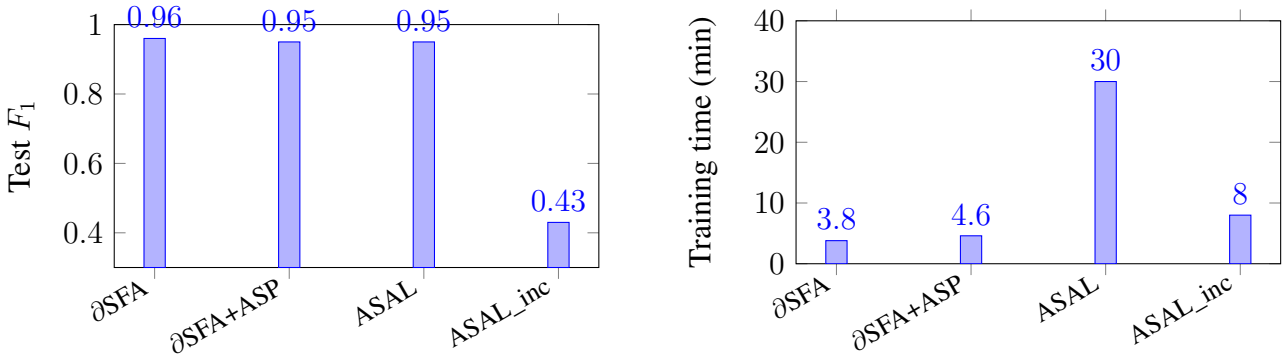


Figure 18: Comparison of test F_1-score (left) and training time (right) for SFA learning on the robot deadlock prediction dataset (Industry 4.0 use case).

The symbolic sequences were used as the training data for ∂SFA and ASAL (one-hot probability vectors in the case of ∂SFA, as before), in an 80/20 train/test split. The latter ensured that no trajectory

was “shared” between the training and the testing set, to avoid data leakages. That is, the symbolic sequences generated from each one of the 100 trajectories were either all added to the training, or to the testing set. Figure 18 presents the results. As previously, ∂SFA achieves the best performance in 4.5 minutes. ASAL is able to match this performance with a cut-off induction time of half an hour. Figure 16 presents the learned automaton.

7 Data Programming for Neuro-Symbolic Training

7.1 Introduction

A central promise of Neuro-Symbolic (NeSy) AI is that we can train perception networks and symbolic reasoning components *jointly*, so that the learned representations are aligned with high-level concepts and temporal structures. In many temporal, event-based applications, however, supervision is naturally given at the level of *complex events* over whole sequences (e.g., whether a trajectory satisfies a pattern), while fine-grained labels for latent concepts (simple events) at each time step are scarce or entirely unavailable. NeSy systems are then trained primarily from *indirect supervision*: neural outputs influence the symbolic model, which in turn yields a sequence-level loss that is backpropagated to the perception network.

While powerful, this regime is prone to *reasoning shortcuts*. The perception network may discover idiosyncratic features of the raw data that correlate with the complex-event labels but have little to do with the intended latent concepts (e.g., artefacts of the data generation process). The symbolic component then adapts to these artefacts rather than to the true temporal structure. Such behaviour is difficult to detect from sequence-level metrics alone and undermines both interpretability (the learned simple events no longer match their intended semantics) and transferability (performance degrades under distribution shift, when shortcuts disappear).

Temporal NeSy methods such as *NeSyA* address part of this challenge by embedding symbolic temporal knowledge—e.g., Symbolic Finite Automata (SFA) encoding complex event patterns—into a probabilistic circuit that consumes neural predictions over time. Given a sequence of perceptual inputs, NeSyA uses a perception network to output probabilities for primitive predicates or symbols at each time step, and compiles the SFA into a tractable arithmetic circuit that computes the acceptance probability of the sequence. The resulting NeSy objective allows us to train the perception network directly against sequence-level complex-event labels and to perform exact probabilistic inference over exponentially many symbolic traces in time linear in the circuit size.

However, even in this setting, the perception network is still driven *only* by sequence-level supervision, and nothing prevents it from converging to shortcut solutions that satisfy the SFA-based loss while distorting latent concepts. This motivates the use of additional, inexpensive signals about latent concepts, beyond a small pool of manually labelled examples. *Data programming* [79] offers such a mechanism: instead of hand-labelling many individual instances, we define multiple noisy labelling sources (labelling functions) and use a generative label model to aggregate their outputs into probabilistic latent labels, which can be used to regularise the NeSy training process. In this section we describe how we instantiate data programming using Snorkel [78] and how we combine *strong* (but indirect) sequence-level supervision with *weak* latent labels in NeSyA, focusing on a temporal MNIST pattern task.

7.2 Data Programming and Snorkel

Data programming is a weak supervision paradigm where, instead of labelling each example by hand, the user specifies a set of *labelling functions* (LFs) that encode arbitrary heuristics, models or rules. Formally, let $\mathcal{X}$ denote the input space and $\mathcal{Y}$ a finite label space (in our case, latent concept labels such as simple events). For each input $x \in \mathcal{X}$, we define m labelling functions

$$\lambda_j : \mathcal{X} \to \mathcal{Y} \cup \{\perp\}, \quad j = 1, \ldots, m,$$

where $\lambda_j(x) = y$ proposes a label $y \in \mathcal{Y}$ and $\lambda_j(x) = \perp$ denotes abstention. Applying the LFs to a dataset $\{x_t\}_{t=1}^n$ yields an *LF output matrix* $\Lambda \in (\mathcal{Y} \cup \{\perp\})^{n \times m}$, where $\Lambda_{t,j} = \lambda_j(x_t)$.

Data programming assumes an unobserved true label $Y_t \in \mathcal{Y}$ for each example x_t and places a parametric generative model over $(Y_t, \Lambda_{t,1}, \ldots, \Lambda_{t,m})$, typically factorised as

$$p_\theta(Y_t, \Lambda_{t,1}, \ldots, \Lambda_{t,m}) \;=\; p(Y_t) \prod_{j=1}^{m} p_\theta(\Lambda_{t,j} \mid Y_t),$$

where the conditional distributions $p_\theta(\Lambda_{t,j} \mid Y_t)$ capture the (unknown) accuracies and propensities of the labelling functions. In the simplest case [79], each LF is modelled as a conditionally independent noisy channel with parameters describing its probability of emitting the correct label versus an incorrect one.

Given only the LF outputs Λ (and optionally a small labelled validation set), the parameters θ can be estimated by maximising the marginal likelihood

$$\max_\theta \sum_{t=1}^{n} \log \sum_{y \in \mathcal{Y}} p_\theta(Y_t = y, \Lambda_{t,1}, \ldots, \Lambda_{t,m}),$$

for example via gradient-based optimisation. Once θ is learned, we obtain for each example a *probabilistic label*

$$\tilde{y}_t(y) \;=\; p_\theta(Y_t = y \mid \Lambda_{t,1}, \ldots, \Lambda_{t,m}), \quad y \in \mathcal{Y},$$

which aggregates the noisy signals from all LFs into a distribution over $\mathcal{Y}$. These probabilistic labels can then supervise a discriminative model (e.g., a neural network) via a standard cross-entropy loss. This two-step process—learning a label model over LFs, then training a discriminative model using its outputs—forms the core of the Snorkel system.

Snorkel extends this basic framework along several dimensions. It supports richer dependency structures between LFs (e.g., modelling correlations and subclass relationships), multi-task settings where the same LFs contribute to several related label spaces, and end-to-end pipelines for defining, debugging and monitoring labelling functions and label models. Importantly, the label model can be trained with no or minimal ground-truth labels, relying instead on structural assumptions (e.g., that LFs are not perfectly anti-correlated) and the diversity of LFs to identify their accuracies.

In our setting, the label space $\mathcal{Y}$ corresponds to a set of latent concepts (simple events) that we wish the NeSyA perception network to predict at each time step. We instantiate each LF λ_j as a CNN trained on the *same small seed set* of manually annotated images, but with a different architecture, initialisation or regularisation scheme (e.g., varying width, depth and dropout). This yields a pool of partially trained CNNs, each with slightly different error patterns. For each image x_t, we record the predicted label $\lambda_j(x_t)$ from each CNN (or $\perp$ if we choose to let an LF abstain under low confidence), and feed the resulting matrix Λ into Snorkel's label model to obtain probabilistic latent labels $\tilde{y}_t$.

Remark 7.1 (Data programming and active learning). Data programming and active learning address complementary aspects of supervision. Data programming uses domain knowledge and weak models to produce *many* noisy labels at low marginal cost, which can be denoised by a label model and used to train a discriminative student. Active learning, in contrast, focuses human effort on *few* but highly informative labels, typically by querying an oracle on instances where the current model is most uncertain or prone to error. Recent work has explored combining the two, for example by using NeSy

models to guide rapid image labelling or to identify where weak supervision sources are unreliable. In our context, data programming provides broad, low-cost coverage of latent concepts, while active learning (Section 6.3) can be used to refine them further on the most critical time points.

7.3 Combining Strong and Weak Supervision in NeSy Training

We now describe how we combine sequence-level supervision from NeSyA with the probabilistic latent labels produced by Snorkel. Let $\mathcal{D}_{\text{seq}}$ denote a set of training sequences, where each sequence consists of perceptual inputs $x_{1:T} = (x_1, \ldots, x_T)$ and a corresponding complex-event label $y^{\text{seq}} \in \{0, 1\}$ indicating whether the sequence satisfies a given pattern. Let $\mathcal{I}_{\text{strong}}$ be an index set of time steps for which we have *strong* simple-event labels $y_t^{\text{strong}} \in \mathcal{Y}$ (one-hot, ground truth), and let $\mathcal{I}_{\text{weak}}$ be an index set of time steps for which we have *weak* probabilistic labels $\tilde{y}_t(\cdot)$ from Snorkel.

The NeSyA perception network f_θ maps each input x_t to a distribution over latent concepts:

$$p_\theta(\cdot \mid x_t) \in \Delta^{|\mathcal{Y}|-1},$$

which is fed into the compiled SFA circuit to obtain the sequence-level acceptance probability $P_\theta(y^{\text{seq}} \mid x_{1:T})$. The standard NeSyA sequence-level loss is

$$L_{\text{seq}}(\theta) \;=\; \frac{1}{|\mathcal{D}_{\text{seq}}|} \sum_{(x_{1:T}, y^{\text{seq}}) \in \mathcal{D}_{\text{seq}}} \text{BCE}\big(P_\theta(y^{\text{seq}} \mid x_{1:T}), y^{\text{seq}}\big),$$

where BCE denotes the binary cross-entropy loss.

If strong simple-event labels are available, we define an image-level loss

$$L_{\text{strong}}(\theta) \;=\; \frac{1}{|\mathcal{I}_{\text{strong}}|} \sum_{t \in \mathcal{I}_{\text{strong}}} \text{CE}\big(p_\theta(\cdot \mid x_t), e_{y_t^{\text{strong}}}\big),$$

where CE is the multi-class cross-entropy and $e_{y_t^{\text{strong}}}$ is the one-hot vector for the true label y_t^{strong}.

To incorporate weak labels from data programming, we define

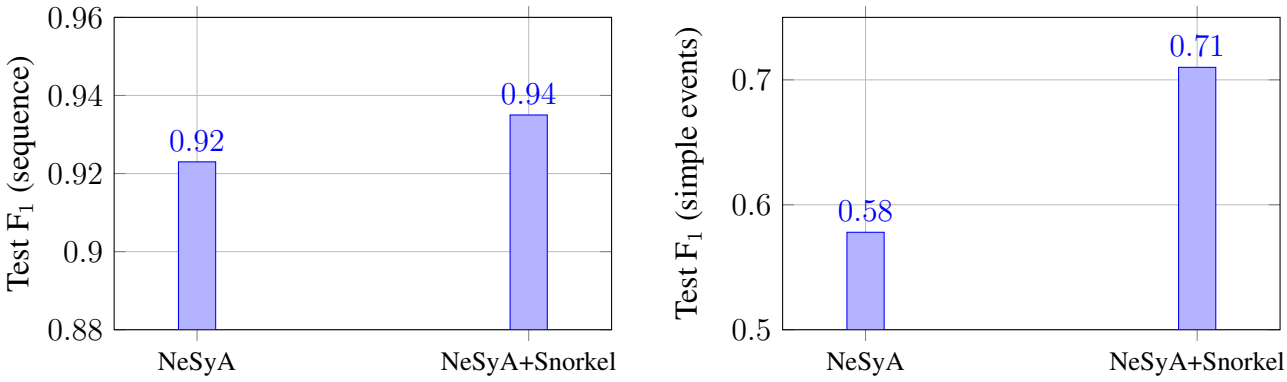
$$L_{\text{weak}}(\theta) \;=\; \frac{1}{|\mathcal{I}_{\text{weak}}|} \sum_{t \in \mathcal{I}_{\text{weak}}} w_t \cdot \text{CE}\big(p_\theta(\cdot \mid x_t), \tilde{y}_t(\cdot)\big),$$

where $\tilde{y}_t(\cdot)$ is the probabilistic label distribution produced by Snorkel's label model for time step t and $w_t \in [0, 1]$ is a confidence weight (e.g., a deterministic function of the entropy of $\tilde{y}_t$, or a global hyperparameter).

The overall training objective is then

$$L(\theta) \;=\; \lambda_{\text{seq}} L_{\text{seq}}(\theta) \;+\; \lambda_{\text{strong}} L_{\text{strong}}(\theta) \;+\; \lambda_{\text{weak}} L_{\text{weak}}(\theta), \tag{10}$$

with non-negative coefficients $\lambda_{\text{seq}}, \lambda_{\text{strong}}, \lambda_{\text{weak}}$ controlling the relative importance of each supervision signal. In the experiments reported below we focus on the setting where no strong simple-event labels are available, that is, $\mathcal{I}_{\text{strong}} = \emptyset$ and $\lambda_{\text{strong}} = 0$, so $L(\theta)$ interpolates between purely indirect supervision via L_{seq} and weak latent supervision via L_{weak}.



(a) Sequence-level F_1-score. **(b)** Image-level F_1-score.

Figure 19: Indicative test F_1-scores on the temporal MNIST pattern task for NeSyA trained with indirect supervision only and NeSyA augmented with Snorkel-based weak supervision.

7.4 Indicative Results on Temporal MNIST

We evaluate this approach on the simple temporal MNIST task introduced in Section 6.4. Each example is a sequence of $T = 10$ MNIST images, and the complex-event label y^{seq} indicates whether the sequence satisfies a fixed SFA pattern defined over latent concepts such as "even digit" and "digit larger than 6". The SFA is fixed and known; only the perception network f_θ is trained. We consider two training regimes:

- **NeSyA (indirect only).** The CNN is trained solely using L_{seq} in (10), with $\lambda_{\text{seq}} = 1$, $\lambda_{\text{strong}} = 0$ and $\lambda_{\text{weak}} = 0$. No simple-event labels (strong or weak) are used.
- **NeSyA+Snorkel.** We construct a small pool of CNN-based labelling functions $\{\lambda_j\}_{j=1}^{m}$, each trained on the same small seed set of manually labelled MNIST images (a subset of the training digits), but with different architectures, initialisations and regularisation schemes (e.g., different widths and dropout probabilities). These CNNs are then frozen and used as LFs in Snorkel's label model, which is trained to produce probabilistic latent labels $\tilde{y}_t(\cdot)$ for a large pool of unlabeled training images. We then train the NeSyA CNN using the combined objective (10) with $\lambda_{\text{seq}} = 1$, $\lambda_{\text{strong}} = 0$ and $\lambda_{\text{weak}} > 0$, so that it learns jointly from the sequence-level NeSyA signal and the Snorkel weak labels.

We evaluate both variants on a held-out test set. For each model, we report: (i) the F_1-score on sequence-level complex-event prediction (target task), and (ii) the F_1-score on image-level latent concept prediction (simple events), using ground-truth digit labels that are never seen during training in this experiment.

In the baseline NeSyA (indirect only) regime, we obtain a sequence-level test F_1 of 0.923 and an image-level test F_1 of 0.578. Adding Snorkel-based weak supervision, without changing the complex-event supervision, yields a NeSyA+Snorkel model with indicative performance of 0.935 sequence-level F_1 and 0.680 image-level F_1. Figure 19 summarises these indicative results.

These results illustrate two main points. First, even in the absence of any simple-event labels, NeSyA's indirect supervision is strong enough to achieve high sequence-level performance on the complex-event task ($F_1 \approx 0.92$), confirming that the SFA-based NeSy objective is effective in this setting. Second, supplementing this indirect signal with weak latent labels from Snorkel significantly

improves the quality of the learned latent concepts (from $F_1 \approx 0.58$ to $F_1 \approx 0.7$ in this indicative run), while also yielding a modest but consistent gain at the sequence level. In practice, this means that data programming can help counteract shortcut behaviour in temporal NeSy training by nudging the perception network towards latent concept representations that are better aligned with the intended simple-event semantics, without requiring dense manual annotation.

8 Discrete Mutual Information Markov Models

Learning discrete representations of raw signals is an important problem in machine learning. In the context of dynamical systems such discovery methods are useful for downstream tasks like forecasting and planning. If the data obeys the Markov property, it is of particular interest to map observations to discrete states, such that a simple Markov chain can be used to analyze the dynamics. The problem of state discovery with complex observations is primarily handled with Hidden Markov Models. In this work we provide a different approach for finding a good mapping from observations to states using a mutual information based objective. We use our method to induce a Markov chain for the input data in an unsupervised fashion. Firstly, we showcase the benefits of our approach in terms of correct discovery in a synthetic domain. Secondly, we present a neurosymbolic system which uses the induced Markov chain along with a probabilistic model checker to answer complex queries about the input data in forecasting scenarios.

8.1 Introduction

Analyzing dynamical systems has multiple applications in AI. Sequential discovery methods, which represent raw input data in an interpretable representation, are therefore of much interest. In the realm of Markov models this leads to representing raw data as, among others, Markov Decision Processes [85], and Markov Chains [18]. The central, question therefore is how to map raw observations to discrete states such that the dynamics of the process are preserved.

Hidden Markov Models and their extensions, “deep" [61, 60] or otherwise [20, 39, 73], are often used for discovery purposes in unsupervised learning. HMMs attempt to model a generative process of the data by considering a) a latent Markov chain which captures the dynamics in a lower-dimensional state space and b) the reconstruction of the observations given the states. The problem with the approach is that in order to accurately reconstruct the data the state must include a number of high frequency and low level characteristics, e.g. local noise, which in a sense disincentivizes the discovery of a high level, slow moving state. This is noted for example in [75].

In order to alleviate this stress we consider learning a mapping between observations and states based on mutual information (MI) of adjacent inputs in a sequence. This approach has been used for learning representations in high dimensions where mutual information is intractable and we must resort to approximations [75, 48, 19, 76] albeit not in a Markov setting. In the simpler case, in which we are interested in mapping observations to a value from a single categorical variable, MI has seen far less use, e.g. in clustering [53].

State discovery in discrete Markov models also bears extreme relevance for the field of neurosymbolic (NeSy) AI [69]. A number of NeSy systems have been introduced both generally [103, 67, 105, 99, 94] and along the temporal dimension [32, 66, 90, 46, 101]. While most principled probabilistic NeSy methods concern themselves mainly with knowledge injection, we instead turn to the use of symbolic solvers as a means to analyze what we have learnt. More specifically, we use probabilistic model checkers [54] for analysis of the induced Markov chain (in the latent space).

8.2 MiMM

Table 11 summarizes the notation that we use in this section. Consider a Markov process over data $x_{1:T}$. The joint can be written as:

Notation	Description
$x \perp y$	Independence between two random variables
$x \perp y \mid z$	Conditional independence of two random variables conditioned on a third
$x \not\perp y$	Dependence between two random variables
$\mathrm{I}(x; y)$	Mutual Information between two random variables
$\mathrm{D}_{\mathrm{KL}}(p(x) \mid\mid q(x))$	KL divergence between two distributions $p(x)$ and $q(x)$
$\mathrm{H}(x)$	Entropy of a random variable
$\mathrm{H}(x \mid y)$	Conditional entropy of x given y

Table 11: Notation used throughout this section.

$$p(x_{1:T}) = p(x_1) \prod_{t=2}^{T} p(x_t \mid x_{t-1})$$

When we are dealing with complex high-dimensional data, e.g. video streams working in the original space of the data, is cumbersome. A popular approach is to turn to a latent variable models.

8.2.1 Hidden Markov Models

For Markov data the standard solution is Hidden Markov Models (HMMs). Along with the observations $x_{1:T}$ the HMM uses a latent sequence of states $z_{1:T}$. The joint is:

$$p(x_{1:T}, z_{1:T}) = p(z_1) \prod_{t=2}^{T} p(z_t \mid z_{t-1}) \prod_{t=1}^{T} p(x_t \mid z_t)$$

This solution avoids modelling the dynamics in the original high-dimensional space of x and instead does so in the lower dimensional space of z. In classical HMMs, z is a single categorical variable; the state. More modern implementations [61, 60] extend the state to be distributed [47] and high dimensional. Naturally while more expressive, this comes at the drawback of the interpretability and tractability of the discovery method. Knowledge injection for “deep" Hidden Markov Models has been studied in [32], e.g. by providing the dynamics, $p(z_1) \prod_{t=2}^{T} p(z_t \mid z_{t-1})$ and learning the observation function $p(x \mid z)$ with a neural network, thus aligning the model with background knowledge.

In HMMs, while the latent state is assumed to be Markov, the data is not, i.e. $x_{t+1} \not\perp x_{t-1} \mid x_t$. In fact, the state z is used to capture possibly long interactions that are necessary for predicting the future. Here we study the implications in HMMs of the data $x_{1:T}$ being Markov. The following must hold:

$$z_t \not\perp z_{t-1} \mid x_t$$

in order for the HMM to generate Markov data. The only way for this to be true in general is that the observation completely determines the state or more formally:

$$p(x \mid z = i) \neq 0 \Rightarrow p(x \mid z = j) = 0$$

for all $j \neq i$. This essentially removes the “hidden" nature of the HMM as the observation uniquely identifies the state. A distribution that satisfies the property above can be described with a sequential process where a) we classify which state z has produced the observation x and b) we quantify the probability that z produces x among all observations it emits. Importantly, the second task is completely unnecessary for inducing the latent Markov chain.

Therefore task a) which is a discriminative task is sufficient for discovery and task b) which is a generative “harder" task does not aid in discovery while making learning significantly more complex.

8.2.2 Mutual Information

Instead of learning a full generative model we opt for a different treatment. We want to find a mapping between original data $\mathcal{X}$ and $\mathcal{Z}$ where $\mathcal{X}$ is the input dimension and $\mathcal{Z}$ is a set of states $\{1, \ldots, N\}$. To this end we use an information theoretic approach. A sufficient representation for Markov data is one which captures maximal information from the dependencies in the original data. Let I be the mutual information (MI). $\mathrm{I}(x_t; x_{t+1})$ measures the reduction of uncertainty of x_{t+1} when conditioning on x_t. Letting $z = f(x)$ we have:

$$I(z_t; z_{t+1}) \leq I(x_t; x_{t+1})$$

by the data processing inequality. A good representation function is one which maximizes the mutual information, i.e. recovers as much of the dependencies which exist between consecutive timesteps in the data x as possible.

As aforementioned this approach, is being used of late for high dimensional representation learning [75, 48, 19, 76]. Further MI has been used in natural language processing for the unsupervised induction of part-of-speech tags in pairs of words (x, y) [83], in a way very similar to the one considered here. Nonetheless, to the best of our knowledge, there is no work in using MI for discrete state discovery in Markov models.

While the representation of data via one of N states is limiting in capacity, it makes the induction of a differentiable function $p_\theta(z \mid x)$ (in practise a softmax activated neural network) via MI closed form. This is not the case when moving to higher dimensions. Formally, let $\{(x_i, \hat{x}_i)\}_{i=1}^{M}$ be a batch of M pairs of data where $(x, \hat{x})$ are adjacent, i.e. are found in consecutive timesteps in the training data. It is then possible to compute the expected mutual information under the stochastic mapping $p_\theta(z \mid x)$ as:

$$\mathrm{I}(z; \hat{z}) = \sum_{z, \hat{z}} p(z, \hat{z}) \log \frac{p(z, \hat{z})}{p(z)p(\hat{z})}$$

$$\text{with} \quad p(z, \hat{z}) = \frac{1}{M} \sum_{i=1}^{M} p_\theta(z \mid x_i) p_\theta(\hat{z} \mid \hat{x}_i)$$

We first compute empirically from the data the joint distribution of z and $\hat{z}$. Note that while locally, i.e. in each batch element z and $\hat{z}$ are considered independent (after they have been conditioned on x and $\hat{x}$), the variables are generally not independent [53]. Using the joint we calculate MI, which since

the sample space of $z \times \hat{z}$ is quadratic in the number of states, can be achieved without approximations.

Relation to Maximum Likelihood. We can relate the MI objective to maximum likelihood. The mutual information $\mathrm{I}(x_t; x_{t+1})$ can be written as $\mathrm{H}(x_{t+1}) - \mathrm{H}(x_{t+1} \mid x_t)$ with H representing the entropy (in the second term the conditional entropy). Consider a model that is trying to predict x_{t+1} given x_t, i.e. $q_\phi(x_{t+1} \mid x_t)$.

Then:

$$\mathrm{I}(x_t; x_{t+1}) \geq \mathrm{H}(x_{t+1}) - \mathcal{L}_{\mathrm{ML}}$$
$$\text{where } \mathcal{L}_{\mathrm{ML}} = -\log q_\theta(x_{t+1} \mid x_t)$$

$\mathcal{L}_{\mathrm{ML}}$ would be the loss of a generative model trying to reconstruct adjacent observations. Therefore, trying to minimize the negative log likelihood is optimizing a lower bound on the MI of adjacent timesteps. If the distribution $q_\theta(x_{t+1} \mid x_t)$ would include a latent state z, as done for example in HMMs, it follows that this z would be optimized for increasing the lower bound on the MI. Our direct objective $\mathrm{I}(z_t; z_{t+1})$ is also optimizing a lower bound on $\mathrm{I}(x_t; x_{t+1})$. It therefore follows that optimizing one also, albeit implicitly, optimizes for the other.

Neural Discrete Representation Learning. Most of the work in inducing discrete representations with neural networks is focused in the more complex, and often intractable, distributed case [47], including examples which use MI [84]. Perhaps the most popular approach is generative, see [92] and its numerous extensions. Our work, which is focused on discovery, i.e. the induction of simple representations, instead focuses on simpler optimization by a) not including intractable objectives and therefore resorting to approximations and b) not reconstructing the data. The latter characteristic, as aforementioned, can potentially be unnecessary for state discovery. Moreover, it is expected to struggle in scenarios which include noise. In such cases, reconstructing the data based on a low capacity z, as is the case for a single categorical variable, is challenging. z must capture high frequency, low level characteristics to reconstruct the data. While in larger dimensions this may be unproblematic, in lower dimensions the necessary capacity for such a task is often not existent.

Knowledge Injection. The objective of maximizing $\mathrm{I}(z_t; z_{t+1})$ attempts to extract enough information from x such that the representation of x_{t+1} can be accurately predicted, while maximally utilizing the state space. This objective is purely data driven and therefore requires significant amounts of data. However, the latent space would ideally capture high level dynamics, for which knowledge may often exist. Using said knowledge is possible for our approach by adding a regularization during training which aids the neural network $p_\theta(z \mid x)$ to discover more accurate and interpretable states. Our knowledge injection routine is based on a declarative specification of the latent states. Details can be found in Appendix 8.5. This includes the specification of $q(z_{t+1} \mid z_t)$, i.e. the transition function. It is expected that aligning $p_\theta(z \mid x)$ with the known dynamics will produce higher quality mappings as the neural network is penalized for not respecting known dynamics.

To achieve this we define the final loss of the model to be:

$$\mathcal{L}(\theta) = -\mathrm{I}(z_t; z_{t+1})$$
$$+ \lambda \cdot \mathrm{D}_{\mathrm{KL}}(p_\theta(z_{t+1} \mid z_t) \,||\, q(z_{t+1} \mid z_t))$$

where the mutual information I should be maximized, i.e. its negation minimized (hence the minus

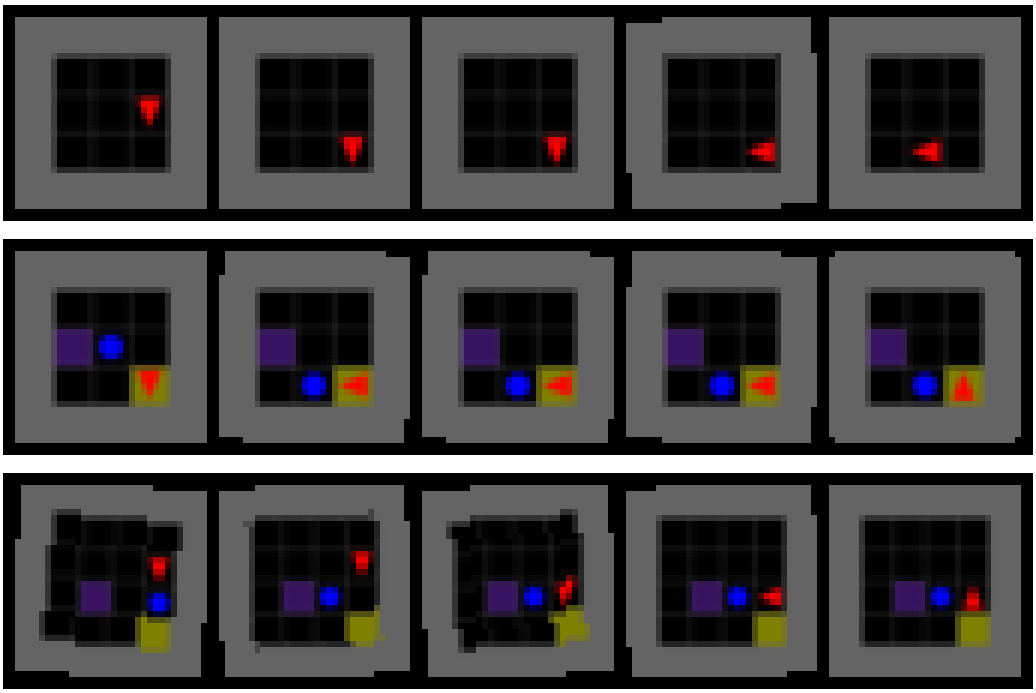


Figure 20: Samples from the three different environments, vertically increasing in complexity. The number of states of each environment are 36, 216 and 384 respectively. The goal of the models is to map the image observations to discrete states in an unsupervised manner.

sign) and the KL divergence should be minimized (hence the plus sign).

When the transition function is unknown λ can be set to zero. The calculation of MI was based on $p_\theta(z_t, z_{t+1})$, the joint distribution, which is a $N \times N$ matrix where N is the number of states. Row normalization of the matrix gives the conditional $p_\theta(z_{t+1} \mid z_t)$ which is used for the calculation of D_{KL}. In comparison, to NeSy-MMs [32] where the knowledge is used in the definition of the probabilistic model and the reconstruction forces the representation to remain informative, in our work the mutual information achieves the same result.

8.3 Experiments

We create a synthetic dataset ranging in complexity from a few states to several hundreds. We benchmark our method against the work of [32] on how accurately the intended states can be recovered. We also consider variants which include noise in the data x which was not previously considered. We compare our method, both with and without knowledge of the dynamics to the baselines.

While [32] use variational Markov models as baselines and perform approximate inference, the implementations here instead use exact inference through an enumerable state space resembling classical HMMs more closely. This is true both for the NeSy-HMM and the Neural-HMM methods. In NeSy-MMs the latent process is given exactly, i.e. the transition matrix and initial distribution of the HMM is fixed according to the environment specification. In Deep-HMM the transition and initial distributions are learnt jointly with gradient descent. Both models differ with classical HMMs, since the observation model $p(x|z)$ is a neural network, replacing simpler emission distributions like, e.g. Gaussian mixtures, as the data modelled are high-dimensional images.

We create a dataset with 200, 100, 150 trajectories of length 10 for train, validation and test respectively. Some sample trajectories can be seen in 20. An observation is generated from a symbolic. For example in the second trajectory of 20 the state used to generate the observation is

Scenario	Ours $\lambda = 1$	Ours 0	NeSy MM	Deep HMM
simple	0.95	0.85	**1.0**	0.94
	0.86	0.83	**1.0**	0.60
medium	**0.96**	0.72	0.86	0.88
	0.78	0.72	0.72	0.55
hard	**0.98**	0.76	0.71	0.87
	0.75	0.63	0.40	0.46

Table 12: Comparison of methods across scenarios and noise conditions, with MiMM split into two subcolumns.

$(\text{x} = 2, \text{y} = 2, \text{dir} = 2, \text{obstacle} = 2, \text{carrying} = 0)$. The possible interpretations in the environment are the state space. All models are trained with a learning rate of 0.0001, a batch size of 32 on a single A100 GPU. For our model we set λ to 1, when considering knowledge injection and to 0 otherwise. The probabilistic models used to generate the symbolic version of the data can be found in Section 8.5. Symbolic sequences are subsequently rendered to image observations using the minigrid library [23].

8.3.1 State Discovery

Table 12 shows the results for our model and how it compares to NeSy-MMs and purely Deep-HMMs (which include no knowledge of the dynamics). We report Adjusted Mutual Information, $\text{AMI}(z; s)$, i.e. the AMI between the predicted states z for each observation x and the actual state s from which x was generated. AMI is the mutual information adjusted for that of a random model. It is high when knowing one random variable significantly reduces the uncertainty of another and is 1 if z and s are identical up to renaming. With AMI we can assess the quality of the predicted latent states beyond exact match. The reason is that a model should not be punished for renaming states and we therefore seek a permutation invariant metric. Lastly, we compare our method with knowledge of the dynamics $\lambda = 1$ and without $\lambda = 0$. The latter is purely statistical and uses no knowledge.

From the results in Table 12 our introduced method is worse that NeSy-MMs only in the simplest setting where the generative approach outperforms our mutual information based one. In more complex scenarios, and particularly when noise is added to the data, we outperform both baselines. Further, we see that the inclusion of knowledge about the dynamics, which both we and NeSy-MMs utilize, greatly improves learning. Finally, we show that even without any knowledge of the environment, our method matches NeSyMMs in the noisy scenarios of medium and outperforms them in hard. Further, the MI based objective, as a purely statistical approach, outperforms Deep-HMMs almost throughout in noisy scenarios.

8.3.2 Forecasting

With knowledge of the dynamics, due to the synthetic nature of the data, we can accurately benchmark the capability of the induced state space for forecasting future events and their probabilities. We use the model with knowledge injection for this task which performs better in terms of aligning with the actual state space.

An example of the application of our method to forecasting can be seen in Figure 21. Two raw datapoints are mapped to their corresponding states, via the mapping $p(z \mid x)$ learnt during training.

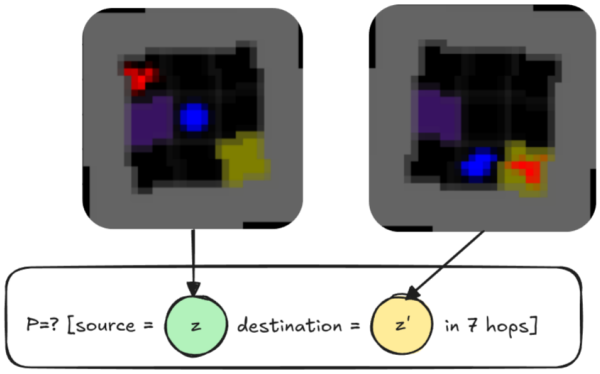


Figure 21: Example of a forecasting query. Two images are passed through the representation function. A query is cast of reaching z' from z is some number of hops (here 7). The query, along with the dynamics (i.e. transition function) learnt through training (which are discrete and low dimensional) are then fed into a probabilistic model checker which computes the probability of reaching z' from z is the given number of steps.

Along with the dynamics $p(z_{t+1} \mid z_t)$ also learnt during training, a probabilistic model checker is used to compute the probability or reaching the latent state z' from the latent state z in a given number of hops.

Depth	L1 Error	True Prob (mean $\pm$ std)
1	0.195	0.377 ± 0.188
2	0.123	0.179 ± 0.108
3	0.0726	0.0980 ± 0.0721
4	0.0475	0.0612 ± 0.0494
5	0.0320	0.0411 ± 0.0328
6	0.0222	0.0296 ± 0.0232
7	0.0148	0.0208 ± 0.0157
8	0.00927	0.0143 ± 0.00898

Table 13: L1 error between predicted and true probabilities across different depths.

The results for forecasting are shown in Table 13. The errors in probability estimation of various scenarios are reported, based on the depth of the query.

8.4 Conclusion

We presented a data driven statistical method for learning discrete state representation of high dimensional data in the context of Markov models based on mutual information. We introduced a simple way to include prior knowledge in training, by aligning the discovered states with prior dynamics. We showcased that our method can achieve better state discovery, compared to both NeSy and purely statistical baselines, especially in the case of noisy data in a synthetic domain. Further, we applied our model and its discovered (latent) Markov chain $p(z_{t+1} \mid z_t)$ to a forecasting task. The system included both neural components, which mapped raw image queries, to their (symbolic) state representation, and utilized symbolic solvers (a probabilistic model checker) to answer queries.

8.5 Appendix: Probabilistic Models Used in the Experiments

Here we give the formal definitions of the Markov Chains used for experimentation. These are also used when specifying the transition function $p(z_{t+1} \mid z_t)$ for knowledge injection. The specification of the first task is:

```
mdp

formula wall_in_front =
  (direction = 0 & y = 0) |
  (direction = 1 & x = MAX_X) |
  (direction = 2 & y = MAX_Y) |
  (direction = 3 & x = 0);

formula forward_x =
  (direction = 1 & x < MAX_X) ? x + 1 :
  (direction = 3 & x > 0)     ? x - 1 : x;

formula forward_y =
  (direction = 0 & y > 0) ? y - 1 :
  (direction = 2 & y < MAX_Y) ? y + 1 : y;


const int MAX_X = 2;
const int MAX_Y = 2;

module robot

  x         : [0..MAX_X];
  y         : [0..MAX_Y];
  direction : [0..3] init 1;    // 0=N, 1=E, 2=S, 3=W


  [step] (wall_in_front) ->
      0.75 : (direction' = mod((direction + 1), 4)) +
      0.25 : (direction' = direction);

  // If free ahead, move forward or rotate
  [step] (!wall_in_front) ->
      0.80 : (x' = forward_x) & (y' = forward_y) +
      0.20 : (direction' = mod((direction + 1), 4));
endmodule
```

And for the second and third tasks, just different sizes are:

```
mdp

// --- Constants ---
const int MAX_X = 2;
const int MAX_Y = 2;

// Pick-up and drop-off zones
const int PICKUP_X = 0;
const int PICKUP_Y = 1;
```

```
const int DROP_X  = MAX_X;
const int DROP_Y  = MAX_Y;

// --- Robot Module ---
module robot

    // Position and direction
    x         : [0..MAX_X];
    y         : [0..MAX_Y];
    direction : [0..3] init 1;    // 0=N, 1=E, 2=S, 3=W


    // --- Movement Transitions ---

    // If wall or obstacle ahead, rotate
    [step] (wall_in_front | obstacle_forward) ->
        0.75 : (direction' = mod((direction + 1), 4)) +
        0.25 : (direction' = direction);

    // If free ahead, move forward or rotate
    [step] (!wall_in_front & !obstacle_forward) ->
        0.80 : (x' = forward_x) & (y' = forward_y) +
        0.20 : (direction' = mod((direction + 1), 4));


endmodule

// Package Module
module object
    // Carrying a package?
    carrying  : bool;
    // --- Pickup Transition ---
    [step] (at_pickup & !carrying) ->
        0.90 : (carrying' = true) +
        0.10 : (carrying' = false);

    // --- Drop-off Transition ---
    [step] (at_drop & carrying) ->
        0.90 : (carrying' = false) +
        0.10 : (carrying' = true);

    [step] (!(at_pickup & !carrying) & !(at_drop & carrying)) -> (carrying'=carrying);
endmodule

// --- Obstacles Module ---
module obstacles
    obstacle : [0..2]; // Positions: 0=(1,1), 1=(1,2), 2=(2,1)

    [step] (true) ->
        0.70 : (obstacle' = obstacle) +
        0.30 : (obstacle' = mod((obstacle + 1), 3));
endmodule
```

```
// --- Derived Formulas ---

// Wall detection based on current direction
formula wall_in_front =
    (direction = 0 & y = 0) |
    (direction = 1 & x = MAX_X) |
    (direction = 2 & y = MAX_Y) |
    (direction = 3 & x = 0);

// Compute position in front of robot
formula forward_x =
    (direction = 1 & x < MAX_X) ? x + 1 :
    (direction = 3 & x > 0)     ? x - 1 : x;

formula forward_y =
    (direction = 0 & y > 0) ? y - 1 :
    (direction = 2 & y < MAX_Y) ? y + 1 : y;

// Obstacle detection in the cell ahead
formula obstacle_forward =
    (forward_x = 1 & forward_y = 1 & obstacle = 0) |
    (forward_x = 1 & forward_y = 2 & obstacle = 1) |
    (forward_x = 2 & forward_y = 1 & obstacle = 2);

// Pickup and drop location check
formula at_pickup = (x = PICKUP_X & y = PICKUP_Y);
formula at_drop   = (x = DROP_X   & y = DROP_Y);
```

References

[1] Jagrati Agrawal, Yanlei Diao, Daniel Gyllstrom, and Neil Immerman. Efficient pattern matching over event streams. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, pages 147–160, 2008.

[2] Kareem Ahmed, Kai-Wei Chang, and Guy Van den Broeck. A pseudo-semantic loss for autoregressive models with logical constraints. *Advances in Neural Information Processing Systems*, 36, 2024.

[3] Elias Alevizos, Alexander Artikis, and Georgios Paliouras. Complex event forecasting with prediction suffix trees. *The VLDB Journal*, 31(1):157–180, 2022.

[4] Elias Alevizos, Alexander Artikis, and Georgios Paliouras. Complex event forecasting with prediction suffix trees. *VLDB J.*, 31(1):157–180, 2022.

[5] Elias Alevizos, Anastasios Skarlatidis, Alexander Artikis, and Georgios Paliouras. Probabilistic complex event recognition: A survey. 50(5), 2017.

[6] Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe reinforcement learning via shielding. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[7] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and computation*, 75(2):87–106, 1987.

[8] Apache. FlinkCEP. Version 2.2-SNAPSHOT, accessed July 2025.

[9] Gianluca Apriceno, Luca Erculiani, and Andrea Passerini. A Neuro-Symbolic Approach for Non-Intrusive Load Monitoring. In *Volume 372: ECAI 2023*, Frontiers in Artificial Intelligence and Applications, pages 3175–3181, 2023.

[10] Gianluca Apriceno, Andrea Passerini, and Luciano Serafini. A Neuro-Symbolic Approach to Structured Event Recognition. In *28th International Symposium on Temporal Representation and Reasoning (TIME 2021)*, volume 206 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 1–14, 2021.

[11] Gianluca Apriceno, Andrea Passerini, and Luciano Serafini. A Neuro-Symbolic Approach for Real-World Event Recognition from Weak Supervision. In *29th International Symposium on Temporal Representation and Reasoning (TIME 2022)*, volume 247 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 1–19, 2022.

[12] George Argyros and Loris D'Antoni. The learnability of symbolic automata. In *International Conference on Computer Aided Verification*, pages 427–445. Springer, 2018.

[13] Alexander Artikis, Marek Sergot, and Georgios Paliouras. An event calculus for event recognition. *IEEE Transactions on Knowledge and Data Engineering*, 27(4):895–908, 2014.

[14] Alexander Artikis, Marek J. Sergot, and Georgios Paliouras. An event calculus for event recognition. *IEEE Trans. Knowl. Data Eng.*, 27(4):895–908, 2015.

[15] Samy Badreddine, Artur d'Avila Garcez, Luciano Serafini, and Michael Spranger. Logic tensor networks. *Artificial Intelligence*, 303:103649, 2022.

[16] Mrudula Balachander, Emmanuel Filiot, and Raffaella Gentilini. Passive learning of regular data languages in polynomial time and data. In *35th International Conference on Concurrency Theory (CONCUR 2024)*, pages 10–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2024.

[17] David Barber. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, 2012.

[18] Leonard E Baum and Ted Petrie. Statistical inference for probabilistic functions of finite state markov chains. *The annals of mathematical statistics*, 37(6):1554–1563, 1966.

[19] Mohamed Ishmael Belghazi, Aristide Baratin, Sai Rajeshwar, Sherjil Ozair, Yoshua Bengio, Aaron Courville, and Devon Hjelm. Mutual information neural estimation. In *International conference on machine learning*, pages 531–540. PMLR, 2018.

[20] Yoshua Bengio and Paolo Frasconi. Input-output hmms for sequence processing. *IEEE Transactions on Neural Networks*, 7(5):1231–1249, 1996.

[21] Marco Bucchi, Alejandro Grez, Andrés Quintana, Cristian Riveros, and Stijn Vansummeren. CORE: a complex event recognition engine. *Proc. VLDB Endow.*, 15(9):1951–1964, 2022.

[22] Mark Chavira and Adnan Darwiche. On probabilistic inference by weighted model counting. *Artificial Intelligence*, 172(6-7):772–799, 2008.

[23] Maxime Chevalier-Boisvert, Bolun Dai, Mark Towers, Rodrigo Perez-Vicente, Lucas Willems, Salem Lahlou, Suman Pal, Pablo Samuel Castro, and Jordan Terry. Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks. In *Advances in Neural Information Processing Systems 36, New Orleans, LA, USA*, December 2023.

[24] Andrew Cropper, Sebastijan Dumančić, Richard Evans, and Stephen H Muggleton. Inductive logic programming at 30. *Machine Learning*, 111(1):147–172, 2022.

[25] Gianpaolo Cugola and Alessandro Margara. TESLA: a formally defined event specification language. In *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems*, pages 50–61, 2010.

[26] Loris D'Antoni and Margus Veanes. The power of symbolic automata and transducers. In *Computer Aided Verification*, pages 47–67, Cham, 2017. Springer International Publishing.

[27] Adnan Darwiche. Decomposable negation normal form. *Journal of the ACM (JACM)*, 48(4):608–647, 2001.

[28] Adnan Darwiche. Tractable boolean and arithmetic circuits. In Pascal Hitzler and Md. Kamruzzaman Sarker, editors, *Neuro-Symbolic Artificial Intelligence: The State of the Art*, volume 342 of *Frontiers in Artificial Intelligence and Applications*, pages 146–172. IOS Press, 2021.

[29] Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264, 2002.

[30] Giuseppe De Giacomo, Moshe Y Vardi, et al. Linear temporal logic and linear dynamic logic on finite traces. In *Ijcai*, volume 13, pages 854–860, 2013.

[31] Lennert De Smet, Gabriele Venturato, Luc De Raedt, and Giuseppe Marra. Neurosymbolic markov models. In *ICML 2024 Workshop on Structured Probabilistic Inference {\&} Generative Modeling*, 2024.

[32] Lennert De Smet, Gabriele Venturato, Luc De Raedt, and Giuseppe Marra. Relational neurosymbolic markov models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 16181–16189, 2025.

[33] Sahil Rajesh Dhayalkar. Symbolic feedforward networks for probabilistic finite automata: Exact simulation and learnability. *arXiv preprint arXiv:2509.10034*, 2025.

[34] Samuel Drews and Loris D'Antoni. Learning symbolic automata. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 173–189. Springer, 2017.

[35] Dana Fisman, Hadar Frenkel, and Sandra Zilles. Inferring symbolic automata. *Logical Methods in Computer Science*, 19, 2023.

[36] Daniel Furelos-Blanco, Mark Law, Anders Jonsson, Krysia Broda, and Alessandra Russo. Induction and exploitation of subgoal automata for reinforcement learning. *Journal of Artificial Intelligence Research*, 70:1031–1116, 2021.

[37] Artur d'Avila Garcez and Luis C Lamb. Neurosymbolic ai: The 3 rd wave. *Artificial Intelligence Review*, 56(11):12387–12406, 2023.

[38] Lise Getoor and Ben Taskar. *Introduction*. The MIT Press, 2007.

[39] Zoubin Ghahramani and Michael Jordan. Factorial hidden markov models. *Advances in neural information processing systems*, 8, 1995.

[40] Nikos Giatrakos, Elias Alevizos, Alexander Artikis, Antonios Deligiannakis, and Minos Garofalakis. Complex event recognition in the big data era: a survey. *The VLDB Journal*, 29(1):313–352, 2020.

[41] Nikos Giatrakos, Elias Alevizos, Alexander Artikis, Antonios Deligiannakis, and Minos N. Garofalakis. Complex event recognition in the big data era: a survey. *VLDB J.*, 29(1):313–352, 2020.

[42] Alejandro Grez, Cristian Riveros, Martín Ugarte, and Stijn Vansummeren. A formal framework for complex event recognition. 46(4), 2021.

[43] Carlos Guestrin, Daphne Koller, Ronald Parr, and Shobha Venkataraman. Efficient solution algorithms for factored mdps. *Journal of Artificial Intelligence Research*, 19:399–468, 2003.

[44] Daniel Gyllstrom, Eugene Wu, Hee-Jin Chae, Yanlei Diao, Patrick Stahlberg, and Gordon Anderson. SASE: Complex event processing over streams. *CoRR*, abs/cs/0612128, 2006.

[45] Awni Hannun, Vineel Pratap, Jacob Kahn, and Wei-Ning Hsu. Differentiable weighted finite-state transducers. *arXiv preprint arXiv:2010.01003*, 2020.

[46] Rishi Hazra, Pedro Zuidberg Dos Martires, and Luc De Raedt. Saycanpay: Heuristic planning with large language models using learnable domain knowledge. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 20123–20133, 2024.

[47] Geoffrey E Hinton. Distributed representations. 1984.

[48] R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. Learning deep representations by mutual information estimation and maximization. In *International Conference on Learning Representations*, 2019.

[49] S Hochreiter. Long short-term memory. *Neural Computation MIT-Press*, 1997.

[50] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Comput.*, 9(8):1735–1780, 1997.

[51] Rodrigo Toro Icarte, Toryn Klassen, Richard Valenzano, and Sheila McIlraith. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *International Conference on Machine Learning*, pages 2107–2116. PMLR, 2018.

[52] Nils Jansen, Bettina Könighofer, Sebastian Junges, Alex Serban, and Roderick Bloem. Safe Reinforcement Learning Using Probabilistic Shields. In Igor Konnov and Laura Kovács, editors, *31st International Conference on Concurrency Theory (CONCUR 2020)*, volume 171 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 3:1–3:16, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[53] Xu Ji, Joao F Henriques, and Andrea Vedaldi. Invariant information clustering for unsupervised image classification and segmentation. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9865–9874, 2019.

[54] Joost-Pieter Katoen. The probabilistic model checking landscape. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 31–45, 2016.

[55] Nikos Katzouris and Georgios Paliouras. Learning automata-based complex event patterns in answer set programming. In *International Conference on Inductive Logic Programming*, pages 52–68. Springer, 2022.

[56] Nikos Katzouris and Georgios Paliouras. Answer set automata: A learnable pattern specification framework for complex event recognition. *ECAI*, 2023.

[57] Nikos Katzouris and Georgios Paliouras. Answer set automata: A learnable pattern specification framework for complex event recognition. In *30th International Symposium on Temporal Representation and Reasoning (TIME 2023)*, pages 17–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2023.

[58] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[59] Robert Kowalski and Marek Sergot. A logic-based calculus of events. *New generation computing*, 4:67–95, 1986.

[60] Rahul Krishnan, Uri Shalit, and David Sontag. Structured inference networks for nonlinear state space models. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

[61] Rahul G Krishnan, Uri Shalit, and David Sontag. Deep kalman filters. *arXiv preprint arXiv:1511.05121*, 2015.

[62] Eilham Lokman, Vik Tor Goh, Timothy Tzen Vun Yap, and Hu Ng. Driving event recognition using machine learning and smartphones. *F1000Research*, 11, 2022.

[63] Christos Louizos, Max Welling, and Diederik P Kingma. Learning sparse neural networks through l_0 regularization. *arXiv preprint arXiv:1712.01312*, 2017.

[64] Will Maddern, Geoff Pascoe, Chris Linegar, and Paul Newman. 1 year, 1000km: The oxford RobotCar dataset. *The International Journal of Robotics Research (IJRR)*, 36(1):3–15, 2017.

[65] Oded Maler and Irini-Eleftheria Mens. A generic algorithm for learning symbolic automata from membership queries. In *Models, Algorithms, Logics and Tools: Essays Dedicated to Kim Guldstrand Larsen on the Occasion of His 60th Birthday*, pages 146–169. Springer, 2017.

[66] Nikolaos Manginas, George Paliouras, and Luc De Raedt. NeSyA: Neurosymbolic automata. to appeat at IJCAI, 2025. Pre-print doi: 10.48550/arXiv.2412.07331.

[67] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. *Advances in neural information processing systems*, 31, 2018.

[68] Robin Manhaeve, Sebastijan Dumančić, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Neural probabilistic logic programming in DeepProbLog. *Artificial Intelligence*, 298, 2021.

[69] Giuseppe Marra, Sebastijan Dumančić, Robin Manhaeve, and Luc De Raedt. From statistical relational to neurosymbolic artificial intelligence: A survey. *Artificial Intelligence*, page 104062, 2024.

[70] Andrew McCallum, Dayne Freitag, Fernando CN Pereira, et al. Maximum entropy markov models for information extraction and segmentation. In *Icml*, volume 17, pages 591–598, 2000.

[71] Dejan Mitrović. Reliable method for driving events recognition. *IEEE Transactions on Intelligent Transportation Systems*, 6(2):198–205, 2005.

[72] Stephen H Muggleton, Dianhuan Lin, Niels Pahlavi, and Alireza Tamaddoni-Nezhad. Meta-interpretive learning: application to grammatical inference. *Machine learning*, 94(1):25–49, 2014.

[73] Kevin Patrick Murphy. *Dynamic bayesian networks: representation, inference and learning*. University of California, Berkeley, 2002.

[74] Takamasa Okudono, Masaki Waga, Taro Sekiyama, and Ichiro Hasuo. Learning weighted finite automata over the max-plus semiring and its termination. *arXiv preprint arXiv:2407.09775*, 2024.

[75] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[76] Ben Poole, Sherjil Ozair, Aaron Van Den Oord, Alex Alemi, and George Tucker. On variational bounds of mutual information. In *International conference on machine learning*, pages 5171–5180. PMLR, 2019.

[77] Luc De Raedt, Kristian Kersting, Sriraam Natarajan, and David Poole. *Statistical Relational Artificial Intelligence: Logic, Probability, and Computation*. 2016.

[78] Alexander Ratner, Stephen H Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. Snorkel: Rapid training data creation with weak supervision. In *Proceedings of the VLDB endowment. International conference on very large data bases*, volume 11, page 269, 2017.

[79] Alexander J Ratner, Christopher M De Sa, Sen Wu, Daniel Selsam, and Christopher Ré. Data programming: Creating large training sets, quickly. *Advances in neural information processing systems*, 29, 2016.

[80] Joris Renkens, Dimitar Shterionov, Guy Broeck, Jonas Vlasselaer, Daan Fierens, Wannes Meert, Gerda Janssens, and Luc De Raedt. ProbLog2: From probabilistic programming to statistical relational learning. In *Proceedings of the NIPS Probabilistic Programming Workshop,*, 2012.

[81] Gurkirt Singh, Stephen Akrigg, Manuele Di Maio, Valentina Fontana, Reza Javanmard Alitappeh, Suman Saha, Kossar Jeddisaravi, Farzad Yousefi, Jacob Culley, Tom Nicholson, et al. ROAD: The ROad event awareness dataset for autonomous driving. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 2022.

[82] Gurkirt Singh, Suman Saha, Michael Sapienza, Philip HS Torr, and Fabio Cuzzolin. Online real-time multiple spatiotemporal action localisation and prediction. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3637–3646, 2017.

[83] Karl Stratos. Mutual information maximization for simple and accurate part-of-speech induction. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1095–1104, 2019.

[84] Karl Stratos and Sam Wiseman. Learning discrete structured representations by adversarially maximizing mutual information. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 9144–9154. PMLR, 13–18 Jul 2020.

[85] Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[86] Gwo-Hshiung Tzeng and Jih-Jeng Huang. *Multiple attribute decision making: methods and applications*. CRC press, 2011.

[87] Elena Umili, Francesco Argenziano, Aymeric Barbin, Roberto Capobianco, et al. Visual reward machines. In *Neural-Symbolic Learning and Reasoning 2022*, volume 3212, pages 255–267. 2023.

[88] Elena Umili and Roberto Capobianco. Deepdfa: Automata learning through neural probabilistic relaxations. In *ECAI 2024*, pages 1051–1058. Ios Press, 2024.

[89] Elena Umili, Roberto Capobianco, and Giuseppe De Giacomo. Grounding ltlf specifications in image sequences. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, volume 19, pages 668–678, 2023.

[90] Elena Umili, Roberto Capobianco, and Giuseppe De Giacomo. Grounding LTLf Specifications in Image Sequences. In *Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning*, pages 668–678, 8 2023.

[91] Leslie G Valiant. The complexity of enumeration and reliability problems. *siam Journal on Computing*, 8(3):410–421, 1979.

[92] Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

[93] Emile Van Krieken, Erman Acar, and Frank Van Harmelen. Analyzing differentiable fuzzy logic operators. *Artificial Intelligence*, 302:103602, 2022.

[94] Emile van Krieken, Thiviyan Thanapalasingam, Jakub Tomczak, Frank Van Harmelen, and Annette Ten Teije. A-nesi: A scalable approximate method for probabilistic neurosymbolic inference. *Advances in Neural Information Processing Systems*, 36:24586–24609, 2023.

[95] Gertjan van Noord and Dale Gerdemann. Finite state transducers with predicates and identities. *Grammars*, 4(3):263–286, 2001.

[96] A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

[97] Marc Vilamala, Tianwei Xing, Harrison Taylor, Luis Garcia, Mani Srivastava, Lance Kaplan, Alun Preece, Angelika Kimmig, and Federico Cerutti. Using deepproblog to perform complex event processing on an audio stream. 10 2021.

[98] Bruce W. Watson. Implementing and using finite automata toolkits. *Natural Language Engineering*, 2(4):295–302, 1996.

[99] Thomas Winters, Giuseppe Marra, Robin Manhaeve, and Luc De Raedt. Deepstochlog: Neural stochastic logic programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 10090–10100, 2022.

[100] Thomas Winters, Giuseppe Marra, Robin Manhaeve, and Luc De Raedt. DeepStochLog: Neural stochastic logic programming. *AAAI Conference on Artificial Intelligence*, 36(9):10090–10100, 2022.

[101] Kai Xi, Stephen Gould, and Sylvie Thiébaux. Neuro-symbolic learning of lifted action models from visual traces. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, pages 653–662, 2024.

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

[103] Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Broeck. A semantic loss function for deep learning with symbolic knowledge. In *International conference on machine learning*, pages 5502–5511. PMLR, 2018.

[104] Wen-Chi Yang, Giuseppe Marra, Gavin Rens, and Luc De Raedt. Safe reinforcement learning via probabilistic logic shields. In Edith Elkind, editor, *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, pages 5739–5749. International Joint Conferences on Artificial Intelligence Organization, 8 2023. Main Track.

[105] Zhun Yang, Adam Ishay, and Joohyung Lee. NeurASP: Embracing neural networks into answer set programming. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 1755–1762. International Joint Conferences on Artificial Intelligence Organization, 2020.

[106] Mahdi Zarei Yazd, Iman Taheri Sarteshnizi, Amir Samimi, and Majid Sarvi. A robust machine learning structure for driving events recognition using smartphone motion sensors. *Journal of Intelligent Transportation Systems*, 28(1):54–68, 2024.

[107] Haopeng Zhang, Yanlei Diao, and Neil Immerman. On complexity and optimization of expensive queries in complex event processing. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, pages 217–228, 2014.

[108] Honghua Zhang, Meihua Dang, Nanyun Peng, and Guy Van den Broeck. Tractable control for autoregressive language generation. In *International Conference on Machine Learning*, pages 40932–40945. PMLR, 2023.