

# NeSyA: Neurosymbolic Automata

Nikolaos Manginas<sup>1,2</sup>, Georgios Paliouras<sup>2</sup> and Luc De Raedt<sup>1,3</sup>

<sup>1</sup>Department of Computer Science and Leuven.AI, KU Leuven, Belgium

<sup>2</sup>Institute of Informatics and Telecommunications, NCSR “Demokritos”, Greece

<sup>3</sup>Centre for Applied Autonomous Sensor Systems (AASS), Örebro University, Sweden

{nikolaos.manginas, luc.deraedt}@kuleuven.be, paliourg@iit.demokritos.gr

## Abstract

Neurosymbolic (NeSy) AI has emerged as a promising direction to integrate neural and symbolic reasoning. Unfortunately, little effort has been given to developing NeSy systems tailored to sequential/temporal problems. We identify symbolic automata (which combine the power of automata for temporal reasoning with that of propositional logic for static reasoning) as a suitable formalism for expressing knowledge in temporal domains. Focusing on the task of sequence classification and tagging we show that symbolic automata can be integrated with neural-based perception, under probabilistic semantics towards an end-to-end differentiable model. Our proposed hybrid model, termed NESYA (Neuro Symbolic Automata) is shown to either scale or perform more accurately than previous NeSy systems in a synthetic benchmark and to provide benefits in terms of generalization compared to purely neural systems in a real-world event recognition task. Code is available at: <https://github.com/nmanginas/nesya>.

## 1 Introduction

Sequence classification/tagging is a ubiquitous task in AI. Purely neural models, including LSTMs [Hochreiter, 1997] and Transformers [Vaswani, 2017], 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 [Garcez and Lamb, 2023] 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 [Winters *et al.*, 2022; De Smet *et al.*, 2024; Umili *et al.*, 2023b], 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

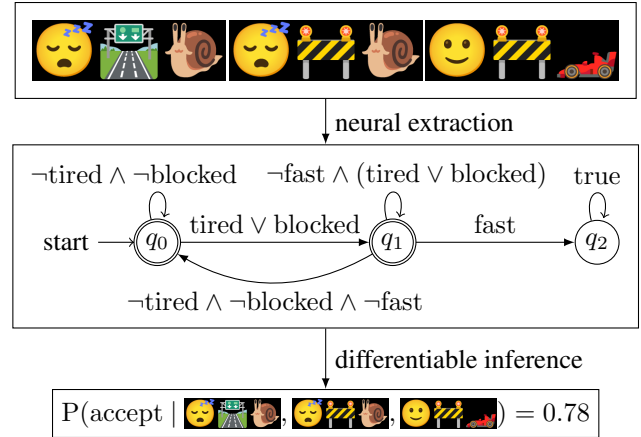
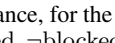


Figure 1: Symbolic automata (middle) are used to reason over sequences of subsymbolic inputs (top) from which information is extracted with the aid of a neural network, performing multilabel classification. For instance, for the image , the correct symbol grounding is {tired, ¬blocked, ¬fast}. The symbolic 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.

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 1 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 [Darwiche and Marquis, 2002].

The closest system to our work is FUZZYA [Umili *et al.*, 2023b], which attempts to address the NeSy integration of LTL<sub>f</sub> 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 [Badredine *et al.*, 2022]. 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) [Xu *et al.*, 2018] and DEEPSTOCHLOG [Manhaeve *et al.*, 2018], 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 [Winters *et al.*, 2022] 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 [Umili *et al.*, 2023a; Umili and Capobianco, 2024] differ from NESYA in terms of [C3], since classical automata lack symbolic transitions and support for atemporal reasoning.

Lastly, [De Smet *et al.*, 2024] 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.
- We introduce an efficient algorithm for inference in NESYA, utilizing matrix-based automata inference and knowledge compilation based approaches for logical inference [Darwiche and Marquis, 2002].
- On a synthetic sequence classification domain, we show that NESYA leads to large performance benefits over FUZZYA [Umili *et al.*, 2023b] and scales orders of magnitude better than DEEPSTOCHLOG [Winters *et al.*, 2022], 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.

## 2 Background

### 2.1 Propositional Logic and Traces

We shall use lowercase to denote propositional variables, e.g. blocked. A propositional formula  $\phi$  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  $\omega$  satisfies a formula  $\phi$  we write  $\omega \models \phi$  and  $\omega$  is called a model of  $\phi$ .

The semantics of propositional logic are given in terms of interpretations. Traces generalize interpretations for temporal

domains. A trace over variables  $V$ ,  $\pi = (\omega_1, \omega_2, \dots, \omega_n)$  is a sequence of interpretations, with  $\omega_i \subseteq V$ . We use  $\pi_t$  to denote the interpretation  $\omega_t$  at timestep  $t$ .

### 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,  $\delta : Q \times Q \rightarrow B(V)$  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  $\delta$  is given in a factored form, e.g.  $\delta(q_0, q_1) = \text{tired} \vee \text{blocked}$  in Figure 1. 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 [Guestrin *et al.*, 2003] with the goal being to exploit the symbolic nature of the transitions without “propositionalizing”.

A symbolic automaton reads traces, i.e. sequences of interpretations  $(\omega_1, \dots, \omega_n)$  ( $\omega_i \subseteq V$ ) 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  $\omega$ . For the SFA in Figure 1 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  $\pi$  we write  $\pi \models \mathcal{A}$ .

### 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  $\phi$  over variables  $V$ , let  $p$  denote a probability vector over the same variables. Each element  $p[i]$  therefore denotes the probability of the  $i^{\text{th}}$  symbol in  $V$  being true. The probability of the formula given  $p$  is then defined as:

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

This task is reducible to weighted model counting (WMC), one of the most widely-used approaches to probabilistic logical inference [Chavira and Darwiche, 2008]. As computing WMC involves summing over all models of a propositional formula, it lies in the #P complexity class of counting problems [Valiant, 1979]. Knowledge Compilation (KC) [Darwiche and Marquis, 2002] is a common approach to solve WMC problems. It involves transforming a logical formula

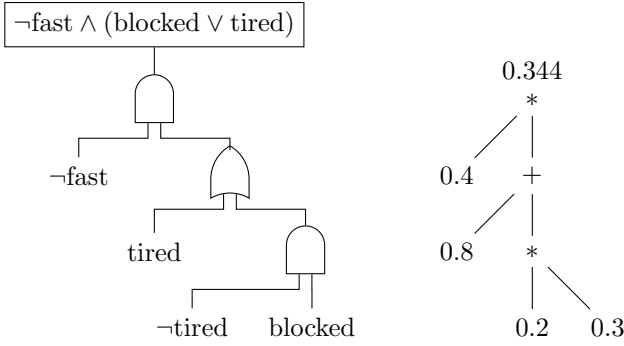


Figure 2: A d-DNNF circuit for the formula  $\phi = \neg \text{fast} \wedge (\text{blocked} \vee \text{tired})$  (left) and an arithmetic circuit produced from the d-DNNF circuit (right). The computation of WMC is shown for the vector  $p = [0.8, 0.3, 0.6]$  for the symbols  $\{\text{tired}, \text{blocked}, \text{fast}\}$  respectively.

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  $q_1 \rightarrow q_1$  in Figure 1. Its compiled form as a d-DNNF circuit [Darwiche, 2001], one of the tractable KC representations, and the computation of WMC can be seen in Figure 2. The logical circuit in Figure 2 (left) is converted to an arithmetic circuit Figure 2 (right) by replacing AND gates with multiplication and OR gates with addition. The weighted model count for the probability vector in Figure 2 can be verified to be correct by:

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

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

### 3 Method

#### 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  $\pi = (\omega_1, \omega_2, \dots, \omega_n)$  we assume a sequence of subsymbolic observations  $o = (o_1, o_2, \dots, o_n)$  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  $f_\theta : \mathbb{R}^m \rightarrow [0, 1]^{|V|}$  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  $i^{\text{th}}$  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.

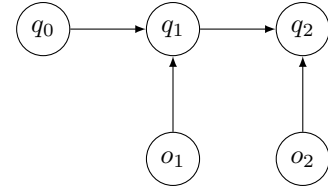


Figure 3: Graphical model for NESYA. Following the approach used in [McCallum *et al.*, 2000], it resembles a Hidden Markov Model with the arrows between states and observations reversed. The random variables  $q_t$  take values from  $Q$ , the state space of the SFA, and the random variables  $o_t$  take values from high-dimensional continuous spaces.

The resulting model is depicted in graphical model notation in Figure 3, where  $q_t$  denotes a discrete random variable over the states of the SFA and  $o_t$  the input observation at time  $t$ . Following [McCallum *et al.*, 2000], we define

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

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

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

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  $\alpha_t$  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 1. Let the first observation be  $o_1 = \text{tired, fast, blocked}$  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  $T(o_i)[q', q] = P(\delta(q', q) \mid f_\theta(o_i))$ . We thus have:

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

The calculation of the entry  $T(\text{tired, fast, blocked})[q_1, q_1]$  was shown in Section 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  $\alpha_0$ , where  $\alpha_0[q_0] = 1$  and  $\alpha_0[q] = 0$  for all  $q \in Q, q \neq q_0$ . We then recursively compute  $\alpha_t$  for each subsequent timestep. Let  $o = (o_1 = \text{tired, blocked, fast}, o_2 = \text{tired, blocked, fast})$ . Consider the neural network predictions for  $o_1$  as above and let  $f_\theta(o_2) = [0.7, 0.9, 0.3]$ . We is is

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

Depending on the task, the  $\alpha$ -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:

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

where  $n$  is the length of the sequence. In this case  $P_{\text{accept}}(o) = 0.023 + 0.7197 = 0.742$ . In other applications, such as sequence tagging, one is interested about the  $\alpha$  values in every timestep.

### 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  $P_{\text{accept}}(o)$ . The computation of  $P_{\text{accept}}(o)$  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 2.3, reduces to propagating an arithmetic circuit comprised of addition, multiplication and subtraction, (b) the  $\alpha$ -recursion which is implemented via standard matrix operations, and (c) a summation over the final  $\alpha$  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.

$$\mathcal{L}(o, L) = \text{BCE}(P_{\text{accept}}(o), L),$$

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 [Manhaeve *et al.*, 2018; Yang *et al.*, 2020; Winters *et al.*, 2022; Umili *et al.*, 2023b]. More concretely, observe that we don't require examples of the form  $(\text{tired, blocked, fast}, \{\text{tired, blocked, fast}\})$ , as we would in a fully supervised multilabel problem, but rather of the form  $((\text{tired, blocked, fast}, \text{tired, blocked, fast}), 0)$ . 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  $f_\theta$  is trained via standard gradient descent on the distant labels.

### 3.3 Semantics and Discussion

Consider a sequence of probability vectors  $(p_1, p_2, \dots, p_n)$  with each vector  $p_t$  assigning a probability to each proposition  $v \in V$  at timestep  $t$ . In NESYA  $p_t = f_\theta(o_t)$ , 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  $(p_1, \dots, p_n)$ , the probability of trace  $\pi$ , a sequence of interpretations, is then:

$$P(\pi \mid (p_1, p_2, \dots, p_n)) = \prod_{t=1}^n P(\pi_t \mid p_t)$$

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

**Theorem 1** ( $\alpha$ -semantics). *It holds that:*

$$\alpha_t[q] = \sum_{\pi \in \text{traces}(q, t)} P(\pi \mid (p_1, p_2, \dots, p_t)).$$

where  $\text{traces}(q, t)$  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  $(p_1, \dots, p_t)$ .

This directly follows from the graphical model but refer to Appendix A for a proof from first principles. A direct consequence is that for an SFA  $\mathcal{A}$  and the sequence  $(p_1, \dots, p_n)$  of symbol probabilities:

$$\sum_{f \in F} \alpha_n[f] = \sum_{\pi \models \mathcal{A}} P(\pi \mid (p_1, p_2, \dots, p_n)). \quad (3)$$

Once the logical transitions of the SFA have been compiled to a tractable form, see Section 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  $\phi$  and relies on the computation of  $\sum_{\omega \models \phi} P(\omega \mid p)$ , 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  $\sum_{\pi \models \mathcal{A}} P(\pi \mid p_1, p_2, \dots, p_t)$  is computable as shown

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

Table 1: 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 [Umili *et al.*, 2023b]

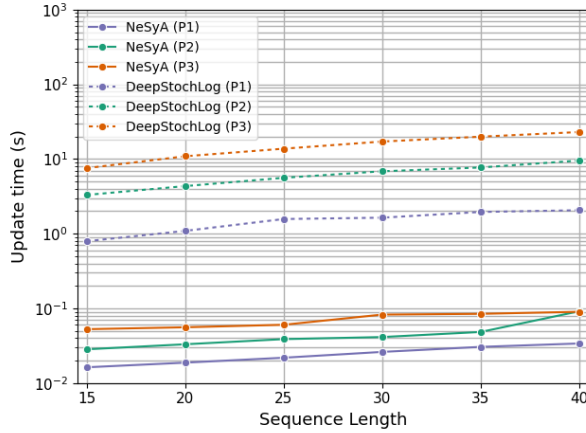


Figure 4: Scalability results for NESYA (solid) and DEEPSTOCHLOG (dashed) for each of the three patterns tested. The y-axis represents the update time for a single batch of 16 sequences in logarithmic scale and the x-axis the sequence length. The systems were benchmarked for three different patterns of varying complexity both in terms of symbols, as well as states of the automaton.

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 [De Giacomo *et al.*, 2013] and are 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 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? <sup>1</sup>

**[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.

### 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 1 the symbols were {tired, blocked, 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 3.2, where the neural component must perform symbol grouping without direct supervision. Instead supervision is provided at the sequence level and the neural component is trained weakly. We benchmarked against DEEPSTOCHLOG [Winters *et al.*, 2022] and DEEPPROBLOG [Manhaeve *et al.*, 2018] in terms of scalability and with FUZZYA [Umili *et al.*, 2023b] both in terms of scalability and accuracy. Figure 4 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

<sup>1</sup>The accuracy of DEEPSTOCHLOG is not compared against that of NESYA, as they generate the same results on the same input.





Figure 5: 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  $\text{close}(p1, p2)$  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  $\text{walking}(p1)$ ,  $\text{running}(p1)$ ,  $\text{active}(p1)$ ,  $\text{inactive}(p1)$  and correspondingly to  $p2$  for the second bounding box. The correct grounding for this image is  $\text{active}(p1)$  and  $\text{walking}(p2)$ . 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  $\text{no\_event}$ ,  $\text{meeting}$ ,  $\text{moving}$ . For the image shown here the annotation is  $\text{moving}$ .

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 DEEP-PROBLOG, for the simplest pattern and a single sequence of length 15, the update time for DEEP-PROBLOG is 140 seconds compared to 0.02 seconds for NESYA.

Next, in Table 1 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.

## 4.2 Event Recognition

In our second experiment we compared NESYA against pure neural solutions on an event recognition task from the

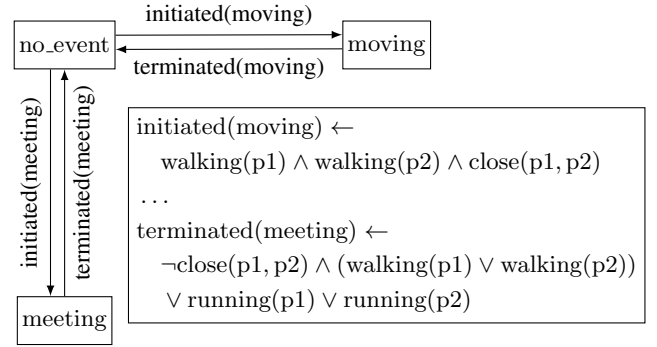


Figure 6: The SFA used for the CAVIAR experiments. It defines transitions between two classes  $\text{meeting}$  and  $\text{moving}$  and a third  $\text{no\_event}$  class. Only a subset of the transition logic is shown for brevity. In the case that no outgoing transition from a state is satisfied the SFA loops in its current state.

CAVIAR benchmark dataset<sup>2</sup>. 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  $\text{moving}$  and  $\text{meeting}$ , which appear more frequently in the data, and a third  $\text{no\_event}$  class. We present three 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  $\text{no\_event}$ , 851 frames of  $\text{moving}$  and 641 frames of  $\text{meeting}$ . 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 5 for an overview. The symbolic component of NESYA in this case is a three-state automaton, capturing a variant of the Event Calculus [Kowalski and Sergot, 1986] programs for CAVIAR found in [Artikis *et al.*, 2014] and can be seen in Figure 6. We use the SFA to label the sequence with the current high-level event in each frame given the ground truth labels 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.

<sup>2</sup><https://homepages.inf.ed.ac.uk/rbf/CAVIARDATA1/>

	#Params	Learning rate					
		0.001		0.0001		0.00001	
		Train	Test	Train	Test	Train	Test
NESYA	258884	<b>0.81</b> $\pm$ 0.13	<b>0.60</b> $\pm$ 0.18	0.87 $\pm$ 0.03	<b>0.85</b> $\pm$ 0.20	0.86 $\pm$ 0.10	<b>0.81</b> $\pm$ 0.18
CNN-LSTM	399683	0.70 $\pm$ 0.23	<u>0.56</u> $\pm$ 0.21	0.84 $\pm$ 0.08	0.35 $\pm$ 0.19	0.17 $\pm$ 0.05	0.15 $\pm$ 0.06
CNN-Transformer	2659767	0.72 $\pm$ 0.26	0.40 $\pm$ 0.10	<b>1.00</b> $\pm$ 0.00	0.68 $\pm$ 0.16	<b>0.97</b> $\pm$ 0.02	<u>0.78</u> $\pm$ 0.14

Table 2: 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 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  $\text{close}(p1, p2)$ , 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 4.1 where supervision was on the sequence level. For NESYA the loss in the CAVIAR dataset is:

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

where  $n$  denotes the sequence length,  $\alpha_t$  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. meeting. For the pure neural solutions  $\alpha_t$  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 2.

The results in Table 2 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.

## 5 Future Work

Recently, [Yang *et al.*, 2023] used the DEEPROBLOG 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 [Alshiekh *et al.*, 2018; Jansen *et al.*, 2020] 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 [Icarte *et al.*, 2018] and their NeSy extension [Umili *et al.*, 2023a] to incorporate logical transitions.

Of significant interest is also the work of [Ahmed *et al.*, 2024], who define a pseudo-semantic loss for autoregressive models with constraints and [Zhang *et al.*, 2023], who similarly address the problem of incorporating constraints in 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.

## 6 Conclusion

We presented the NESYA system for integrating neural networks with symbolic automata, under clean probabilistic semantics. We showed that current systems, such as DEEPSTOCHLOG, DEEPROBLOG, and FUZZYA struggle to achieve scalable and accurate solutions in temporal domains. The NESYA system was instead shown to scale considerably better and achieve higher accuracy.

## A Proof of Theorem 1

We focus on the meaning of  $\alpha_t(q)$ , from which we can also draw further conclusions. Consider an SFA over propositions  $V$ . Let  $\text{traces}(q, t)$  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  $(p_1, p_2, \dots, p_n)$  with each vector  $p_i$  assigning a probability to each proposition  $v \in V$  at timestep  $i$ . Refer to Section 2.3 for an example of  $p_i$ . The probability of a trace  $\pi$  is then:

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

Theorem 1 states that

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

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

$$\alpha_1(q) = \sum_{\omega \models \delta(q_0, q)} P(\omega \mid p_1) = \sum_{\pi \in \text{traces}(q, 1)} P(\pi),$$

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:

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

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  $\omega$  with a trace  $\pi$ , i.e.  $\pi.\omega = (\pi_1, \dots, \pi_t, \omega)$ . 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  $\omega$  causing the SFA to transition from  $q'$  to  $q$ .  $\square$

An immediate consequence of Theorem 1 is that:

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

for an SFA  $A$ .

## B Implementation Details

All experiments were implemented in Pytorch and Python 3.11. For the experiment in Section 4.1 we use the implementation of FUZZYA provided by the authors<sup>3</sup> with minimal changes. Both NESYA and FUZZYA were trained for

<sup>3</sup>[https://github.com/whitemech/grounding\\_LTLf\\_in\\_image\\_sequences](https://github.com/whitemech/grounding_LTLf_in_image_sequences)

a fixed ammount of 100 epochs. For the second experiment (Section 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 boxes.

## Acknowledgments

NM and GP are supported by the project EVENFLOW, Robust Learning and Reasoning for Complex Event Forecasting, which has received funding from the European Union’s Horizon research and innovation programme under grant agreement No 101070430. LDR is supported by the KU Leuven Research Funds (iBOF/21/075, C14/24/092), by the Flemish Government under the “Onderzoeks programma Artificiële Intelligentie (AI) Vlaanderen” programme, the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, and the European Research Council (ERC) under the European Union’s Horizon Europe research and innovation programme (grant agreement n°101142702).

## Contribution Statement

LDR and GP share last authorship.

## References

- [Ahmed *et al.*, 2024] 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.
- [Alshiekh *et al.*, 2018] 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.
- [Artikis *et al.*, 2014] 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.
- [Badreddine *et al.*, 2022] Samy Badreddine, Artur d’Avila Garcez, Luciano Serafini, and Michael Spranger. Logic tensor networks. *Artificial Intelligence*, 303:103649, 2022.
- [Chavira and Darwiche, 2008] Mark Chavira and Adnan Darwiche. On probabilistic inference by weighted model counting. *Artificial Intelligence*, 172(6-7):772–799, 2008.
- [Darwiche and Marquis, 2002] Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264, 2002.
- [Darwiche, 2001] Adnan Darwiche. Decomposable negation normal form. *Journal of the ACM (JACM)*, 48(4):608–647, 2001.
- [De Giacomo *et al.*, 2013] 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.



- [De Smet *et al.*, 2024] 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.
- [Garcez and Lamb, 2023] Artur d’Avila Garcez and Luis C Lamb. Neurosymbolic ai: The 3 rd wave. *Artificial Intelligence Review*, 56(11):12387–12406, 2023.
- [Guestrin *et al.*, 2003] Carlos Guestrin, Daphne Koller, Ronald Parr, and Shobha Venkataraman. Efficient solution algorithms for factored mdps. *Journal of Artificial Intelligence Research*, 19:399–468, 2003.
- [Hochreiter, 1997] S Hochreiter. Long short-term memory. *Neural Computation MIT-Press*, 1997.
- [Icarte *et al.*, 2018] Rodrigo Toro Icarte, Torny 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.
- [Jansen *et al.*, 2020] 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.
- [Kowalski and Sergot, 1986] Robert Kowalski and Marek Sergot. A logic-based calculus of events. *New generation computing*, 4:67–95, 1986.
- [Manhaeve *et al.*, 2018] 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.
- [McCallum *et al.*, 2000] 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.
- [Umili and Capobianco, 2024] Elena Umili and Roberto Capobianco. Deepdfa: Automata learning through neural probabilistic relaxations. In *ECAI 2024*, pages 1051–1058. Ios Press, 2024.
- [Umili *et al.*, 2023a] 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.
- [Umili *et al.*, 2023b] Elena Umili, Roberto Capobianco, and Giuseppe De Giacomo. Grounding ltl specifications in image sequences. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, volume 19, pages 668–678, 2023.
- [Valiant, 1979] Leslie G Valiant. The complexity of enumeration and reliability problems. *siam Journal on Computing*, 8(3):410–421, 1979.
- [Vaswani, 2017] A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.
- [Winters *et al.*, 2022] 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 10091–10100, 2022.
- [Xu *et al.*, 2018] 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.
- [Yang *et al.*, 2020] Zhun Yang, Adam Ishay, and Joohyung Lee. Neurasp: Embracing neural networks into answer set programming. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 1755–1762. International Joint Conferences on Artificial Intelligence Organization, 7 2020. Main track.
- [Yang *et al.*, 2023] 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.
- [Zhang *et al.*, 2023] 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.