Neurosymbolic Markov Models

Lennert De Smet [*1] **Gabriele Venturato** [*1] **Luc De Raedt** [†1 2] **Giuseppe Marra** [†1]

Abstract

Many fields of AI require models that can handle both probabilistic sequential dependencies and logical rules. For example, autonomous vehicles must obey traffic rules in uncertain environments. Deep Markov models excel in managing sequential probabilistic dependencies but fall short in incorporating logical constraints. Conversely, neurosymbolic AI (NeSy) integrates deep learning with logical rules into end-to-end differentiable models, yet struggles to scale in sequential settings. To address these limitations, we introduce neurosymbolic Markov models (NeSy-MM), which merge deep probabilistic Markov models with logic. We propose a scalable strategy for inference and learning in NeSy-MM combining Bayesian statistics, automated reasoning and gradient estimation. Our experimental results demonstrate that this framework not only scales up neurosymbolic inference, but also that incorporating logical knowledge into Markov models improves their performance.

1. Introduction

Markov models are the theoretical foundation for many successful applications of artificial intelligence. For example, Markov processes are at the core of reinforcement learning (Sutton & Barto, 2018) and other temporal tasks, such as speech recognition (Juang & Rabiner, 1991), meteorological predictions (Khiatani & Ghose, 2017), music generation (Austin et al., 2021), sports analytics (Van Roy et al., 2023) and many more (Mor et al., 2020). They are so popular mainly because they naturally factorise a sequential problem into step-wise probability distributions. Such a decomposition leads to better predictions in terms of bias and variance compared to models that do not incorporate the sequential nature of the problem (Bishop, 2006).

[*]Equal first authorship [†]Equal last authorship [1]KU Leuven, Belgium [2]Örebro University, Sweden. Correspondence to: Gabriele Venturato <gabriele.venturato@kuleuven.be>.

Accepted by the Structured Probabilistic Inference & Generative Modeling workshop of ICML 2024, Vienna, Austria.

Neurosymbolic AI (NeSy) has also enjoyed a tremendous increase in attention. Its general goal is to combine the generalisation potential of symbolic, i.e. logical, reasoning with the representational learning prowess of neural networks. Such a combination already exists in many different flavours, using either fuzzy logic (Badreddine et al., 2022) or probabilistic logic (Manhaeve et al., 2021; Yang et al., 2020; Huang et al., 2021; De Smet et al., 2023). The latter case is of special interest, as the probabilistic NeSy systems provide a solid semantics to handle uncertainty, but also to tackle generative tasks. Unfortunately, however, they cannot exploit the sequential decomposition inherent in temporal reasoning tasks, thereby limiting their applicability in complex sequential problems.

To overcome these limitations, we introduce *neurosymbolic Markov models* (NeSy-MMs), the first integration of sequential probabilistic deep models with NeSy. In particular, our contribution is a new differentiable neurosymbolic particle filter that combines Rao-Blackwellised (Liu et al., 2019) inference and state-of-the-art discrete gradient estimation with continuous relaxations (Petersen et al., 2021).

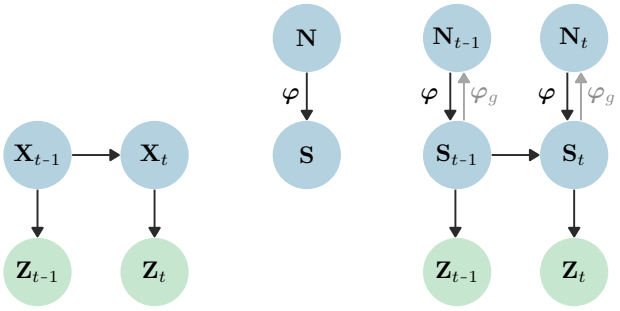
2. Preliminaries

2.1. Markov Models

Hidden Markov models (HMMs) are sequential probabilistic models for discrete-time Markov processes (Baum & Petrie, 1966). Given sequences of *states* $\mathbf{X} = (\mathbf{X}_t)_{t\in\mathbb{N}}$ and *observations* $\mathbf{Z} = (\mathbf{Z}_t)_{t\in\mathbb{N}}$, an HMM factorises the joint probability distribution $p(\mathbf{X}, \mathbf{Z})$ as,

$$p(\mathbf{X}_0)p(\mathbf{Z}_0 \mid \mathbf{X}_0) \prod_{t\in\mathbb{N}} p(\mathbf{X}_{t+1} \mid \mathbf{X}_t)p(\mathbf{Z}_{t+1} \mid \mathbf{X}_{t+1}), \quad (1)$$

where $\mathbf{X}_t$ is a fully latent state (Figure 1a). If $\mathbf{X}_t$ has a known factorisation in the form of a Bayesian network (BN) (Pearl, 1988), then the process and its observations encode a *Markovian dynamic Bayesian network* (DBN) (Dean & Kanazawa, 1989). Note that $\mathbf{X}_t$ and $\mathbf{Z}_t$ are random vectors that can have both discrete and continuous components. In all that follows, a specific assignment of a random variable or vector will be written in lowercase. For example, $\boldsymbol{x}_t = (x_{t,1}, \ldots, x_{t,D})$ is an assignment of the random vector $\mathbf{X}_t = (X_{t,1}, \ldots, X_{t,D})$ of dimension D.



(a) HMM (b) NeSy (c) NeSy-MM

Figure 1: Probabilistic graphical model representations of the different systems considered in this work. Blue represents the states, green the observations.

2.2. Probabilistic Neurosymbolic AI

Probabilistic NeSy methods originate from the field of statistical relational AI (StarAI) that integrates statistical AI with logic (De Raedt et al., 2016; Marra et al., 2024). This integration leads to systems capable of performing inference and learning with uncertainty over *symbolic*, i.e. logical, knowledge. For example, the logical relations `player_at(player1, location1)` and `monster_at(monster1, location1)` can be used to apply the rule `hit(P,M) :- player_at(P,L), monster_at(M,L)` and deduce that `player1` is hit by `monster1` because they are in the same location. Moreover, this knowledge is often uncertain in practice, resulting in uncertainty on whether the deduced logical relations hold. For example, consider the case of a sneaky monster. If we are unsure whether `monster_at(monster1, location1)` is true or not, we will also be uncertain whether the player is hit or not. Notice that uncertain logical relations can be modelled as binary random variables, which justifies the integration with statistics.

While StarAI assumes knowledge to be neatly represented as a symbolic state $\mathbf{S}$, such an assumption does not always hold. Images, sound waves or natural language are usually represented as *subsymbolic* data, i.e. tensors, that are not directly usable by relational AI. Therefore, probabilistic NeSy methods use neural predicates φ to map subsymbolic data to a probability distribution over symbolic representations that can be used by StarAI. Figure 1b depicts a probabilistic graphical model (PGM) (Koller & Friedman, 2009) representation of a NeSy system. More formally, given a boolean variable Y from $\mathbf{S}$ with domain $\{y, \neg y\}$ and a set of rules $\mathcal{R}$ on the symbols in $\mathbf{S}$, inference in NeSy computes the probability that the query Y is true via weighted model integration (WMI) (Morettin et al., 2021)

$$p_{\varphi}(Y{=}y \mid \mathbf{N}{=}\mathbf{n}) = \int \mathbb{1}_{\mathbf{s}\models_{\mathcal{R}} y}\, p_{\varphi}(\mathbf{S}{=}\mathbf{s} \mid \mathbf{N}{=}\mathbf{n})\, \mathrm{d}\mathbf{s}, \quad (2)$$

$$= \int_{\mathbf{s}\models_{\mathcal{R}} y} p_{\varphi}(\mathbf{S}{=}\mathbf{s} \mid \mathbf{N}{=}\mathbf{n})\, \mathrm{d}\mathbf{s}, \quad (3)$$

where the distribution of $\mathbf{S}$ is parametrised by a neural network φ from the subsymbolic state $\mathbf{N}$.

A prominent way of representing neurosymbolic models is via probabilistic logic programming (PLP) (De Raedt et al., 2007). A running example will illustrate the main concepts, while a more technical exposition is given in Appendix A.

```
player(Im, P) ~ normal(noisy_player(Im)).
monster(Im, M) ~ normal(noisy_monster(Im)).
clumsy ~ bernoulli(0.75).

hits(M, P) :-
   distance(M, P, D), D < 2, not clumsy.
game_over(Im) :-
   player(Im, P), monster(Im, M), hits(M, P).
query(game_over(image.png)).
```

Figure 2: DeepSeaProbLog encoding of Example 2.1.

Example 2.1. *Figure 2 describes a simple game. The first two rows introduce deep random variables representing the normally distributed locations of the player `P` and the monster `M`. These variables are parametrised by the neural predicates `noisy_player` and `noisy_monster`, respectively. Both networks map the tensor representation of the image `Im` into the respective symbols `P` and `M`. The third row introduces a binary random variable `C` indicating that the monster will be clumsy with a 75% chance. Next, two rules determine when a monster can hit (`H`) a player and when the image depicts a lost game (`G`). The final line asks for the probability that `image.png` represents a lost game.*

3. Neurosymbolic Markov Models

A *neurosymbolic Markov model (NeSy-MM)* combines the sequential and partially observable nature of HMMs and DBNs (Figure 1a) with neurally parametrised relational probability distributions (Figure 1b). That is, we consider Markov processes $\mathbf{X} = (\mathbf{X}_t)_{t\in\mathbb{N}}$ with observations $\mathbf{Z} = (\mathbf{Z}_t)_{t\in\mathbb{N}}$ where the state $\mathbf{X}_t$ is now a *neurosymbolic state* $\mathbf{X}_t = (\mathbf{N}_t, \mathbf{S}_t)$. Figure 1c depicts the graphical model of this novel integration. NeSy-MMs represent joint probability distributions $p_{\varphi}(\mathbf{N}, \mathbf{S}, \mathbf{Z})$ that factorise as,

$$p_{\varphi}(\mathbf{S}_0 \mid \mathbf{N}_0) p(\mathbf{N}_0) p(\mathbf{Z}_0 \mid \mathbf{S}_0)$$
$$\prod_{t\in\mathbb{N}} p_{\varphi}(\mathbf{S}_{\mathbf{t+1}} \mid \mathbf{S}_t, \mathbf{N}_{t+1}) p(\mathbf{N}_{t+1}) p(\mathbf{Z}_{t+1} \mid \mathbf{S}_{t+1}). \quad (4)$$

Despite the similarity with Eq. 1, NeSy-MMs are complex

models that enable the definition of a wide variety of distributions, taking into account three main aspects of interest.

Symbolic knowledge. Having a NeSy state means we perform inference in a symbolic state space where *relational logic rules* $\mathcal{R}$ govern the relationship between symbols, both within a single time slice and in the transition between states. This symbolic space allows us to incorporate human knowledge into our reasoning process, giving guarantees on how the sequential process evolves. For example, we can guarantee safety properties throughout the entire sequence (see Example 3.1).

Hybrid domains. Discrete binary random variables can model logical relations under uncertainty. However, we are interested in systems capable of modelling continuous aspects of reality, e.g. for robotics or raw data processing, as well as continuous latent random variables required to model the prior $p(\mathbf{N})$ necessary for generative tasks.

Neural parametrisation. We can parametrise our probabilistic logic theory on given sequential subsymbolic knowledge with neural predicates $\boldsymbol{\varphi}$, allowing us to answer *discriminative queries* $p_{\boldsymbol{\varphi}}(Y{=}y \mid \mathbf{N}{=}\mathbf{n}, \mathbf{Z} = \mathbf{z})$ via

$$\int_{\mathbf{s}\models_{\mathcal{R}} y} p_{\boldsymbol{\varphi}}(\mathbf{s}_0 \mid \mathbf{n_0}, \mathbf{z_0}) \prod_{t\in\mathbb{N}} p_{\boldsymbol{\varphi}}(\mathbf{s_{t+1}} \mid \mathbf{s}_t, \mathbf{n}_{t+1}, \mathbf{z}_{t+1}) \, \mathrm{d}\mathbf{s}. \tag{5}$$

Alternatively, if we assume a generative perspective, i.e. the $\boldsymbol{\varphi}_g$ edges in Figure 1c, we can define the neural parametrisation of our model in terms of a variational auto-encoders (VAE) (Kingma & Welling, 2013). This perspective leads to a new factorisation that allows us to tackle *generative tasks*, where we sample $\mathbf{n} \sim p_{\boldsymbol{\varphi}}(Y{=}y, \mathbf{N} \mid \mathbf{Z} = \mathbf{z})$, which is:

$$\int_{\mathbf{s}\models_{\mathcal{R}} y} p_{\boldsymbol{\varphi}}(\mathbf{s}_0, \mathbf{N}_0 \mid \mathbf{z}_0) \prod_{t\in\mathbb{N}} p_{\boldsymbol{\varphi}}(\mathbf{s_{t+1}}, \mathbf{N}_{t+1} \mid \mathbf{s}_t, \mathbf{z}_{t+1}) \, \mathrm{d}\mathbf{s} \tag{6}$$

While some attempts have been made to cover part of these aspects (see Section 4), we are the first to cover all of them and provide a uniform strategy for inference and learning.

Example 3.1. *Figure 3 shows a new version of the game from Example 2.1. The player can now move in the environment with a Markovian transition function* `player_move` *based on the player's previous location and the static monster's position. The observation rule* `safe` *guarantees the player's safety at every time step within the horizon* `0:T`*. Finally, we can query* `game_lost(image.png)`$_t$ *for any* `t` $\in \{$`0, ..., T`$\}$*. Notice that this NeSy-MM depends only on the first image at time* `t=0` *and that the projection in the future is done via the logical transition rules.*

4. Desiderata of NeSy-MM Inference

Inference in a NeSy-MM reduces to dealing with either the discriminative distribution $p_{\boldsymbol{\varphi}}(\mathbf{S} \mid \mathbf{Z}, \mathbf{N})$ or the generative distribution $p_{\boldsymbol{\varphi}}(\mathbf{S}, \mathbf{N} \mid \mathbf{Z})$. These distributions are at the basis of more specific probabilistic tasks, like expectation computation or maximum likelihood optimisation. NeSy-MMs are related to both neurosymbolic AI and sequential probabilistic models, yet inference methods from these fields can not handle the probability distribution of a NeSy-MM. To be compatible with a NeSy-MM, a suitable inference method has to satisfy three desiderata. **(D.I)** It has to be able to deal with both discrete and continuous random variables. Optionally, it should exploit the symbolic nature of a binary logic variable. **(D.II)** It has to exploit the sequential nature of a NeSy-MM. In particular, this property includes the support of transition probabilities $p_{\boldsymbol{\varphi}}(\mathbf{S}_{t+1} \mid \mathbf{S}_t, \mathbf{N}_{t+1})$ that are logical, neural or purely probabilistic in nature. **(D.III)** It has to support differentiation, either exact or approximate, in order to allow for the optimisation of any neural components of the NeSy-MM. This property is especially crucial for the neurosymbolic applications of NeSy-MMs.

Both neurosymbolic AI and existing probabilistic techniques are insufficient for NeSy-MM inference. On the neurosymbolic side, scalability remains the biggest problem. Purely exact techniques (Manhaeve et al., 2021; Yang et al., 2020) do not scale to non-trivial time horizons, while approximate techniques (Huang et al., 2021; van Krieken et al., 2024) are still limited and do not support continuous variables. A hybrid approach exist for discrete and continuous domains (De Smet et al., 2023), but only in the static setting. On the side of probabilistic models, non-parametric techniques can infer any generic hidden Markov model (Koller & Friedman, 2009) and have been applied to the statistical relational setting (Nitti et al., 2016). However, their integration with the neural paradigm is often paired with strong distributional assumptions (Krishnan et al., 2017). In particular, gradient-based optimisation is often difficult (Ścibior et al., 2021; Corenflos et al., 2021; Younis & Sudderth, 2023).

5. Inference and Learning

To bridge the gap between NeSy and sequential probabilistic models, we propose a new, differentiable inference technique that combines non-parametric Bayesian inference with exact NeSy inference.

5.1. Differentiable NeSy-MM Particle Filtering

Traditional particle filters are not differentiable because they perform resampling (Appendix B). To solve this differentiability problem, we propose a novel solution that takes advantage of the neurosymbolic nature of a NeSy-MM. While

```
player(Im, P)₀ ~ normal(noisy_player(Im)).
player(Im, P)ₜ ~ normal(Next) :-
    player(Im, P)ₜ₋₁, monster(Im, M),
    player_move(P, M, Next).
monster(Im, M) ~ normal(noisy_monster(Im)).
clumsy ~ bernoulli(0.75).

hits(M, P)ₜ :-
    distance(M, P, D)ₜ, D < 2, not clumsy.

game_overₜ(Im) :-
    player(Im, P)ₜ, monster(Im, M),
    hits(M, P)ₜ.

safeₜ(Im, P) :-
    player(Im, P)ₜ, monster(Im, M),
    distance(M, P, D)ₜ, D > 2.

observe(safe₀:T, true).
query(game_over(image1)T).
```

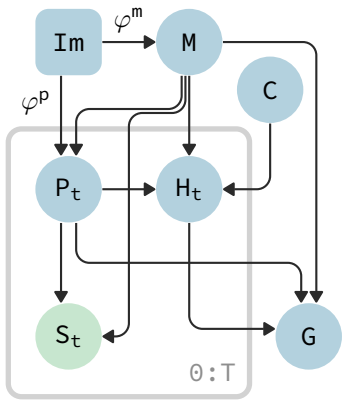


Figure 3: On the left, a logic programming description of the game Example 3.1 in the discrete-continuous probabilistic NeSy language DeepSeaProbLog. On the right, the corresponding graphical model. We use plate notation to indicate a Markov transition. A rolled-out version is available in the appendix in Figure 8.

existing methods focus on repairing the differentiability of resampling (Ścibior et al., 2021; Younis & Sudderth, 2023), we circumvent this problem by using a Rao-Blackwellised particle filter (RBPF) (Murphy & Russell, 2001). A RBPF avoids the need for resampling caused by the separation of the observations $\mathbf{Z}_t$ from the transitions. Instead, it recursively computes $p_{\boldsymbol{\varphi}}(\mathbf{X}_{t+1} \mid \mathbf{Z}_{0:t+1})$ as

$$\int p_{\boldsymbol{\varphi}}(\mathbf{X}_{t+1} \mid \mathbf{x}_t, \mathbf{Z}_{t+1}) p_{\boldsymbol{\varphi}}(\mathbf{x}_t \mid \mathbf{Z}_{0:t}) \, \mathrm{d}\mathbf{x}_t, \tag{7}$$

by immediately transitioning a set of recursive samples to the next time step using the exact transition probabilities $p_{\boldsymbol{\varphi}}(\mathbf{X}_{t+1} \mid \mathbf{x}_t, \mathbf{Z}_{t+1})$. In the case $\mathbf{X}_t$ is purely discrete, computing these probabilities can leverage the advances in exact inference from both neurosymbolic AI (Kisa et al., 2014; Tsamoura et al., 2021) and probabilistic AI (Darwiche, 2020; Holtzen et al., 2020).

By removing resampling and having access to the exact transition probabilities, we can exploit an up-until-now unexplored synergy with gradient estimation. State-of-the-art discrete gradient estimation algorithms use samples and the gradients of the probability of those samples to approximate the gradients of finite distributions. In other words, they need the exact probabilities of these distributions to function. Hence, gradient estimation can be used to restore the differentiability of the particle filter by removing resampling and having access to the exact transition probabilities. In our implementation, we opted for the state-of-the-art performance of RLOO (Kool et al., 2019) for gradient estimation.

5.2. NeSy inference via Markov block factorization

Unfortunately, computing $p_{\boldsymbol{\varphi}}(\mathbf{X}_{t+1} \mid \mathbf{x}_t, \mathbf{Z}_{t+1})$ exactly when $\mathbf{X}_t$ also contains continuous variables is generally only possible under strict assumptions such as Gaussian densities. We mitigate this problem by factorising the NeSy-MM further through the novel notion of *Markov blocks* to separate the finite from the infinite.

Definition 5.1 (Markov Block)**.** Given a conditional probability distribution $p(\mathbf{X} \mid \mathbf{Z})$, two random variables X_1, X_2 in $\mathbf{X}$ belong to the same *Markov block* if and only if they become dependent when conditioning on $\mathbf{Z}$, i.e. $X_1 \not\perp\!\!\!\perp X_2 \mid \mathbf{Z}$.

Proposition 5.2 (Markov Block Factorisation)**.** *Every conditional probability distribution $p(\mathbf{X} \mid \mathbf{Z})$ can be factorised in terms of its Markov blocks.*

Proof. Markov blocks correctly factorise a distribution $p(\mathbf{X} \mid \mathbf{Z})$ if and only if they define a partition on $\mathbf{X}$. Moreover, every equivalence relation on a set induces a partition on that set. Therefore, it is sufficient to prove that belonging to the same Markov block is an equivalence relation on $\mathbf{X}$. Reflexivity and symmetry are trivially satisfied, so let us prove transitivity. Assume X_1 and X_2 belong to the same Markov block B_1, and also that X_2 and X_3 are in the same block B_2. By definition, we know that there exist variables Z_1 and Z_2 in $\mathbf{Z}$ such that $X_1 \not\perp\!\!\!\perp X_2 \mid Z_1$ and $X_2 \not\perp\!\!\!\perp X_3 \mid Z_2$, i.e. Z_1 depends on X_1 and X_2 while Z_2 depends on X_2 and X_3. If Z_1 and Z_2 are the same variable, then B_1 and B_2 must be the same block. Otherwise, suppose that the blocks B_1 and B_2 are different, i.e. $B_1 \neq B_2$, then it must hold that $X_1 \not\perp\!\!\!\perp X_2 \mid \mathbf{Z}$ but $X_2 \perp\!\!\!\perp X_3 \mid \mathbf{Z}$, or the symmetric case. However, this case is not possible since

both Z_1 and Z_2 belong to $\mathbf{Z}$ and we had that $X_2 \not\perp\!\!\!\perp X_3 \mid Z_2$. As B_1 and B_2 are the same blocks, transitivity follows. □

Applying the Markov block factorisation to the conditional probability distribution $p_{\boldsymbol{\varphi}}(\mathbf{X}_{t+1} \mid \mathbf{x}_t, \mathbf{Z}_{t+1})$ with Markov blocks $\{\mathbf{X}^i_{t+1}\}^B_{i=1}$ yields

$$p_{\boldsymbol{\varphi}}(\mathbf{X}_{t+1} \mid \mathbf{x}_t, \mathbf{Z}_{t+1}) = \prod_{i=1}^{B} p_{\boldsymbol{\varphi}}(\mathbf{X}^i_{t+1} \mid \mathbf{x}_t, \mathbf{Z}_{t+1}). \quad (8)$$

If we split every Markov block $\mathbf{X}^i_t$ into a finite part $\mathbf{F}^i_t$ and infinite part $\mathbf{I}^i_t$, this factorisation can be further refined into

$$\prod_{i=1}^{B} p_{\boldsymbol{\varphi}}(\mathbf{F}^i_{t+1} \mid \mathbf{I}^i_{t+1}\mathbf{x}_t, \mathbf{Z}_{t+1}) p_{\boldsymbol{\varphi}}(\mathbf{I}^i_{t+1} \mid \mathbf{x}_t, \mathbf{Z}_{t+1}). \quad (9)$$

By first obtaining samples for the infinite random variables $\mathbf{I}^i_t$ in every i^{th} Markov block using a traditional particle filter, we are again left with a purely finite probability distribution $p_{\boldsymbol{\varphi}}(\mathbf{F}^i_{t+1} \mid \mathbf{I}^i_{t+1}\mathbf{x}_t, \mathbf{Z}_{t+1})$ that we compute exactly. *The result is a novel Rao-Blackwellised particle filter for NeSy-MMs (NeSy-PF) that handles hybrid domains and exploits the inner conditional dependency structure of the NeSy states* $\mathbf{X}_t$.

To wrap up this section, we answer the remaining question of differentiability for infinite random variables. As our NeSy-PF reduces to a generic particle filter for infinite random variables, we can exploit the many proven and tailored gradient estimation algorithms (Ścibior et al., 2021; Corenflos et al., 2021; Younis & Sudderth, 2023). In our case, we followed the work of Ścibior et al. (2021) as it provides strong baseline performance. *In summary, we recover gradient-based optimisation of infinite, finite and binary logical variables by joining local exact inference with specialised gradient estimation.*

6. Experiments

We test the capabilities of NeSy-MMs on a generative setting inspired by the Mario dataset of Misino et al. (2022), extended using MiniHack (Samvelyan et al., 2021), a flexible framework to define environments of the open-ended game NetHack (Küttler et al., 2020).

6.1. Experimental Setting

This dataset consists of trajectories, each containing a sequence of images of length T representing an agent moving T steps in a grid world of size $N \times N$, surrounded by walls. The starting position of the agent is randomly initialised, while the goal position is always at the bottom right of the grid. The actions the agent can take are uniformly sampled among the four cardinal directions, i.e. up, down, left, right (Figure 4). The input of the model is then a set of trajectories, like the one in Figure 4. However, we generate two datasets for grid size $N = 5$ and $N = 10$, both of them with trajectories of length 10. The task for this dataset, following the setup of Misino et al. (2022), is to learn to generate images of the environment that follow a list of given actions and satisfy the rules of NetHack. More details for reproducibility are available in Appendix C.

We use VAEL (Misino et al., 2022) as a neurosymbolic baseline and two other fully neural baselines: a variational transformer architecture (VT); and a NeSy-MM without logical rules and with neural networks as transition function (Deep-MM).

We consider two metrics for the evaluation. First, the reconstruction loss (R) is measured by the mean absolute difference in pixel values, which is first averaged over the images, then separately averaged over all images of the sequence. Second, the reconstruction accuracy, which uses a pre-trained classifier for the location of the agent and measures how much the reconstructed trajectory aligns with the ground truth. This is important to understand if the agent is moving according to the actual rules of the game.

6.2. Results

Results are reported in Table 1 and Table 2. We discuss the main findings by highlighting the advantages of NeSy-MMs.

Better in-distribution generalisation. The advantage of integrating knowledge about the environment in the generation process is clear. The neural baselines are able to get a lower reconstruction loss, but that does not translate into good reconstruction accuracy. In our case, accuracy is guaranteed by the logical rules that govern the movements of the agent. The advantage is even clearer in the 10×10 grid, where the neural baselines fail to produce accurate behaviour because of the larger state space. Notice that we compressed the images from 16 to 8 pixels per grid cell for the 10×10 grids because of hardware memory limits.

Scaling NeSy to non-trivial time horizons. Neurosymbolic methods are known for their scalability issues. When sequential settings are considered, the situation is even more dramatic. VAEL fails to process a sequence of length 10 on an even smaller grid of size 3×3, with $6h$ timeout. On the contrary, we manage to perform inference and generative learning that do not deteriorate over time, even compared to the neural baselines.

Generation is logically consistent, interpretable and intervenable. One of the biggest advantages of neurosymbolic generation is its ability to induce interpretable and intervenable logical consistency into subsymbolic generation. As an example, consider the generation in Figure 5

Figure 4: MiniHack dataset example trajectory of length 5 in a 5×5 grid, with the corresponding labels.

Table 1: Results for the MiniHack dataset on a 5×5 grid.

Method	R (↓)	Rec. Acc. (↑)
VT	$\mathbf{4.98 \pm 0.17}$	49.38 ± 4.80
Deep-MM	6.85 ± 0.29	38.00 ± 3.42
NeSy-MM	8.64 ± 1.07	$\mathbf{66.43 \pm 1.23}$

Table 2: Results for the MiniHack dataset on a 10×10 grid.

Method	R (↓)	Rec. Acc. (↑)
VT	$\mathbf{2.55 \pm 0.04}$	4.53 ± 1.11
Deep-MM	3.81 ± 0.58	7.74 ± 2.68
NeSy-MM	4.15 ± 0.41	$\mathbf{69.52 \pm 1.72}$

where the generative model was asked to generate a trajectory for the agent starting in the middle and following a given sequence of actions while adhering to the movement rules of the game. Because the symbolic rules of the game are an inherent part of the generative model, NeSy-MMs have no problem generating such trajectories. Other methods lack the necessary semantics or symbolic knowledge to fully guarantee this sort of logical consistency. Moreover, NeSy-MMs are flexible in the sense of allowing for different constraints to hold at test time, allowing for zero-shot adherence to new queries (Figure 6).

7. Conclusion

We introduced neurosymbolic Markov models (NeSy-MM), together with a novel scalable and differentiable particle filtering technique, for inference and learning. The promising preliminary results show that the integration of symbolic knowledge into neural Markov models leads to significant improvements in generative tasks, and provides guarantees that neural models alone cannot achieve. Future work will focus on testing NeSy-MMs in various environments, such as those with probabilistic transition functions and continuous state variables, as well as exploring different tasks like classification and transition learning.

Acknowledgements

This work was supported by the KU Leuven Research Fund (C14/18/062), the Flemish Government under the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen" programme, the EU H2020 ICT48 project "TAILOR" under contract #952215, and the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

References

Austin, J., Johnson, D. D., Ho, J., Tarlow, D., and Van Den Berg, R. Structured denoising diffusion models in discrete state-spaces. *Advances in Neural Information Processing Systems*, 34:17981–17993, 2021.

Badreddine, S., Garcez, A. d., Serafini, L., and Spranger, M. Logic tensor networks. *Artificial Intelligence*, 2022.

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

Bishop, C. M. Pattern recognition and machine learning. *Springer google schola*, 2:645–678, 2006.

Corenflos, A., Thornton, J., Deligiannidis, G., and Doucet, A. Differentiable particle filtering via entropy-regularized optimal transport. In *International Conference on Machine Learning*, pp. 2100–2111. PMLR, 2021.

Darwiche, A. An advance on variable elimination with applications to tensor-based computation. In *ECAI 2020*, pp. 2559–2568. IOS Press, 2020.

De Raedt, L., Kimmig, A., and Toivonen, H. Problog: A probabilistic prolog and its application in link discovery. In *IJCAI*. Hyderabad, 2007.

De Raedt, L., Kersting, K., Natarajan, S., and Poole, D. Statistical relational artificial intelligence: Logic, probability, and computation. *Synthesis lectures on artificial intelligence and machine learning*, 2016.

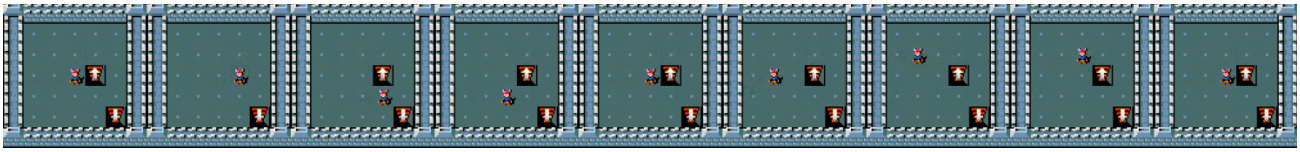


Figure 5: Generated trajectory for actions: right, down, left, up, left, up, right, down.

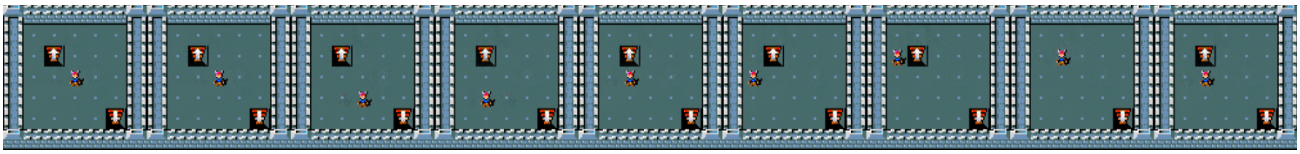


Figure 6: Generated trajectory for actions: right, down, left, up, left, up, right, down but with the test-time constraint that the area to the right of the start position should not be entered. This constraint manifests itself at the first transition, where the agent stays in the safe zone by not moving instead of following the instruction to go right.

De Smet, L., Zuidberg Dos Martires, P., Manhaeve, R., Marra, G., Kimmig, A., and De Readt, L. Neural probabilistic logic programming in discrete-continuous domains. *UAI*, 2023.

Dean, T. and Kanazawa, K. A model for reasoning about persistence and causation. *Computational intelligence*, 5 (2):142–150, 1989.

Holtzen, S., Van den Broeck, G., and Millstein, T. Scaling exact inference for discrete probabilistic programs. *Proceedings of the ACM on Programming Languages*, 4 (OOPSLA):1–31, 2020.

Huang, J., Li, Z., Chen, B., Samel, K., Naik, M., Song, L., and Si, X. Scallop: From probabilistic deductive databases to scalable differentiable reasoning. *NeurIPS*, 2021.

Juang, B. H. and Rabiner, L. R. Hidden markov models for speech recognition. *Technometrics*, 33(3):251–272, 1991.

Khiatani, D. and Ghose, U. Weather forecasting using hidden markov model. In *2017 International Conference on Computing and Communication Technologies for Smart Nation (IC3TSN)*, pp. 220–225. IEEE, 2017.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *ICLR*, 2015.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Kisa, D., Van den Broeck, G., Choi, A., and Darwiche, A. Probabilistic sentential decision diagrams. In *Fourteenth International Conference on the Principles of Knowledge Representation and Reasoning*, 2014.

Koller, D. and Friedman, N. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

Kool, W., van Hoof, H., and Welling, M. Buy 4 reinforce samples, get a baseline for free! *ICLR Deep RL Meets Structured Prediction Workshop*, 2019.

Krishnan, R., Shalit, U., and Sontag, D. Structured inference networks for nonlinear state space models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 31(1), 2017.

Küttler, H., Nardelli, N., Miller, A., Raileanu, R., Selvatici, M., Grefenstette, E., and Rocktäschel, T. The nethack learning environment. *Advances in Neural Information Processing Systems*, 33:7671–7684, 2020.

Liu, R., Regier, J., Tripuraneni, N., Jordan, M., and Mcauliffe, J. Rao-blackwellized stochastic gradients for discrete distributions. *ICML*, 2019.

Manhaeve, R., Dumančić, S., Kimmig, A., Demeester, T., and De Raedt, L. Neural probabilistic logic programming in deepproblog. *Artificial Intelligence*, 2021.

Marra, G., Dumančić, S., Manhaeve, R., and De Raedt, L. From statistical relational to neurosymbolic artificial intelligence: a survey. *Artificial Intelligence*, pp. 104062, 2024.

Misino, E., Marra, G., and Sansone, E. Vael: Bridging variational autoencoders and probabilistic logic programming. In *NeurIPS*, 2022.

Mor, B., Garhwal, S., and Kumar, A. A systematic review of hidden markov models and their applications. *Archives of Computational Methods in Engineering*, 28:1429 – 1448, 2020.

Morettin, P., Zuidberg Dos Martires, P., Kolb, S., and Passerini, A. Hybrid probabilistic inference with logical and algebraic constraints: a survey. In *IJCAI*, 2021.

Murphy, K. and Russell, S. Rao-blackwellised particle filtering for dynamic bayesian networks. In *Sequential Monte Carlo methods in practice*, pp. 499–515. Springer, 2001.

Nitti, D., De Laet, T., and De Raedt, L. Probabilistic logic programming for hybrid relational domains. *Machine Learning*, 2016.

Pearl, J. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan kaufmann, 1988.

Petersen, F., Borgelt, C., Kuehne, H., and Deussen, O. Learning with algorithmic supervision via continuous relaxations. *NeurIPS*, 2021.

Samvelyan, M., Kirk, R., Kurin, V., Parker-Holder, J., Jiang, M., Hambro, E., Petroni, F., Kuttler, H., Grefenstette, E., and Rocktäschel, T. Minihack the planet: A sandbox for open-ended reinforcement learning research. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021.

Ścibior, A., Masrani, V., and Wood, F. Differentiable particle filtering without modifying the forward pass. In *International Conference on Probabilistic Programming (PROBPROG)*, 2021.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

Tsamoura, E., Carral, D., Malizia, E., and Urbani, J. Materializing knowledge bases via trigger graphs. *Proceedings of the VLDB Endowment*, 14(6):943–956, 2021.

van Krieken, E., Thanapalasingam, T., Tomczak, J., Van Harmelen, F., and Ten Teije, A. A-nesi: A scalable approximate method for probabilistic neurosymbolic inference. *Advances in Neural Information Processing Systems*, 36, 2024.

Van Roy, M., Robberechts, P., Yang, W.-C., De Raedt, L., and Davis, J. A markov framework for learning and reasoning about strategies in professional soccer. *Journal of Artificial Intelligence Research*, 77:517–562, 2023.

Yang, Z., Ishay, A., and Lee, J. Neurasp: Embracing neural networks into answer set programming. In *IJCAI*, 2020.

Younis, A. and Sudderth, E. B. Differentiable and stable long-range tracking of multiple posterior modes. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

A. Probabilistic Logic Programming

Logic programming has three main building blocks, being *terms*, *atoms* and *rules*. A term is the logic construct used to place the values one would like to reason over in a logical formula, like a constant `c` or a variable `V` in the simplest case. Additionally, a term can also be formed recursively by applying a functor `f` to a tuple of terms, i.e., something of the form $\texttt{f(t}_1\texttt{,...,t}_K\texttt{)}$. Next, atoms are relations that can be either true or false depending on their arguments and the given background knowledge. They are written using a predicate symbol $\texttt{q}/K$ of arity K filled in with terms, e.g., $\texttt{q(t}_1\texttt{,...,t}_K\texttt{)}$. Atoms and terms are used in the definition of rules of the form $\texttt{h :- b}_1\texttt{,...,b}_K$, where `h` is an atom and each $\texttt{b}_i$ is a *literal*, i.e. an atom or the negation of an atom. We call `h` the *head* of the rule while the conjunction $\texttt{b}_1\texttt{,...,b}_K$ is the body of the rules. If the body of the rule is logically true, then the rule expresses that the head of the rule is true as well by logical consequence.

Example A.1 (Logic Program)**.** *The following logic program expresses the background knowledge necessary to find out if someone is a grandparent of someone else. There is a shared knowledge base at the top in the form of two atoms that express that* `george` *is the father of* `alice` *and that* `alice` *is the mother of* `william`*. The first two rules define the parent relation as being either the mother or the father of someone. Finally, the last rule defines that* `X` *is a grandparent of* `Y` *if there exists an intermediate person* `Z` *that is the parent of* `Y` *and whose parent is* `X`*.*

```
father(george, alice). mother(alice, william).

parent(X, Y) :-
    father(X, Y).
parent(X, Y) :-
    mother(X, Y).

grandparent(X, Y) :-
    parent(X, Z), parent(Z, Y).
```

While atoms are either true or false in traditional logic programming, they can be probabilistically true or false when going to probabilistic logic programming. That is, an atom can now be annotated with the probability that it is true. For instance, `0.42 :: father(george, alice)` expresses that one is only 42 percent sure that `george` is in fact the father of `alice`.

Modern implementations of probabilistic logic programming allow for the probabilities of atoms to be parameterised by neural networks to achieve a neurosymbolic integration. Apart from being limited to parametrising atoms that represent finite random variables that can only be true or false, these implementations have also been extended to the infinite domain (De Smet et al., 2023). To facilitate the definition and inclusion of such atoms, two additional building blocks were added. First, there is the *neural distributional fact* (NDF), an expression of the form $\texttt{x ~ distribution(n}_1\texttt{,...,n}_K\texttt{)}$. Here, `x` is a term representing a random variable distributed according to `distribution` filled in with the numeric terms $\texttt{n}_1\texttt{,...,n}_K$. These numeric terms can be constant numerical values or the output of a neural network. Second, to use neural distributional facts in a logical expression, which is always either true or false, *probabilistic comparison formulae* (PCF) were also introduced. These are specific atoms that take the terms coming from a neural distributional fact as argument.

Example A.2 (DeepSeaProbLog program)**.** *In the following piece of code, we show how NDFs and PCFs work together to write a probabilistic logic program that represents a simple model of the weather. Two finite variables first model whether it is cloudy and what degree of humidity currently holds. The* `temperature` *NDF models a normal distribution with mean* `15` *and standard deviation* `3`*. Because of the uncertain nature of these variables, the truth value of the atoms* `rain` *and* `good_weather` *will also be uncertain. These atoms are defined under the NDFs; it is rainy when it is cloudy and humid, and the weather is good when it does not rain and the temperature is high or if it does rain, but the temperature is low.*

```
cloudy ~ bernoulli(0.7)
humid ~ categorical([0.3, 0.5, 0.2], [dry, moist, wet]).
temperature ~ normal(15, 3).

rain :-
    cloudy, not (humid =:= dry).
```

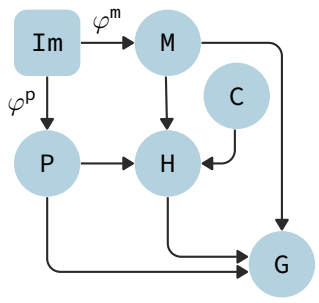


Figure 7: Probabilistic graphical model view of the DeepSeaProbLog encoding in Figure 2.

```
good_weather :-
    not rain, temperature > 20.
good_weather :-
    rain, temperature < 0.
```

Finally, let us provide some more details regarding the running example in the main body of the paper.

Example A.3. *Figure 7 shows the probabilistic graphical model view of Example 2.1. Notice how the two rules* `H` *(player hit by the monster) and* `G` *(game lost) are represented as binary random variables in the graphical model. The logical rules among them and the other variables are implicitly defined by the topology of the model itself. Specifically, the rules are encoded in the probabilistic conditional tables, that we omit for brevity. In mathematical terms, Equation (2) tells us that, the probability that* `image.png` *represents a lost game is*

$$p(\texttt{game\_over(image.png)}) = \int_{(P,M,C)\models_{HG}} p(P \mid \texttt{image.png})p(M \mid \texttt{image.png})p(C)\,\mathrm{d}P\mathrm{d}M\mathrm{d}C, \tag{10}$$

with a small abuse of notation for the discrete variable `C`.

B. Neurosymbolic Particle Filter

Let us focus on the global NeSy state variable $\mathbf{X} = (\mathbf{N}, \mathbf{S})$ and how it evolves over time. The recursive equation of a particle filter applied to a NeSy-MM $(\mathbf{X}, \mathbf{Z})$ computing the probability of a state $\mathbf{X_{t+1}}$ at time step $t+1$ given observations $\mathbf{Z}_{0:t+1}$ from time steps 0 to $t+1$ is

$$p_{\boldsymbol{\varphi}}(\mathbf{X}_{t+1} \mid \mathbf{Z}_{0:t+1}) = \frac{p(\mathbf{Z}_{t+1} \mid \mathbf{X}_{t+1})}{p(\mathbf{Z}_{t+1} \mid \mathbf{Z}_{0:t})} \int p_{\boldsymbol{\varphi}}(\mathbf{X}_{t+1} \mid \mathbf{x}_t) p_{\boldsymbol{\varphi}}(\mathbf{x}_t \mid \mathbf{Z}_{0:t})\,\mathrm{d}\mathbf{x}_t. \tag{11}$$

Practical implementations of these recursive equations task require three steps. First, a set of samples is drawn from the current time t distribution, i.e., $\{\mathbf{x}_t^{(n)}\}_{n=1}^N \sim p_{\boldsymbol{\varphi}}(\mathbf{X}_t \mid \mathbf{Z}_{0:t})$. Then, this set is transitioned to the next time step via the transition distribution $p_{\boldsymbol{\varphi}}(\mathbf{X}_{t+1} \mid \mathbf{x}_t)$. Finally, each of these samples is reweighted according to the observation probabilities $p_{\boldsymbol{\varphi}}(\mathbf{Z}_{t+1} \mid \mathbf{X}_{t+1})$. The resulting set of weighted samples is then approximately distributed according to $p_{\boldsymbol{\varphi}}(\mathbf{X}_{t+1} \mid \mathbf{Z}_{0:t+1})$.

Instead of keeping a set of samples $\{\mathbf{x}_t^{(n)}\}_{n=1}^N$ weighted by their observations $p_{\boldsymbol{\varphi}}(\mathbf{Z}_{0:t+1} \mid \mathbf{x}_t^{(n)})$, practical particle filters use *resampling* to avoid the sample set from collapsing to samples with very low probability. Concretely, they use the observations probabilities as the weights of a finite random variable with N outcomes, one for each sample $\mathbf{x}_t^{(n)}$ and take N samples from the distribution of this variable to use as the recursive set of samples for the next filtering step. Unfortunately, while the original set of weights could be differentiated and used to approximate gradients for every sample, the resampling step is not differentiable, preventing the particle filter from being used in our setting.

C. Reproducibility

Architectures. The variational transformer architectures consist of a separate initial convolutional VAE architecture coupled with a transformer decoder that autoregressively generated the next images in the sequence. This transformer has a

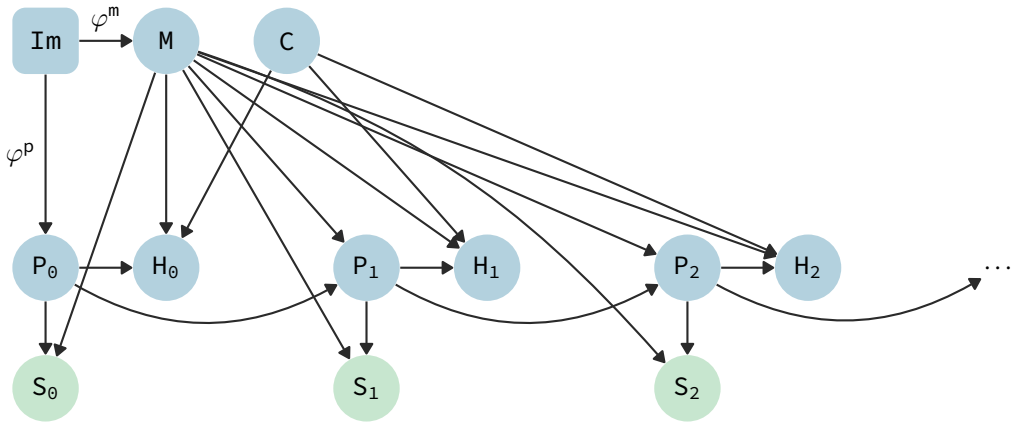


Figure 8: Rolled-out graphical model of Figure 3, from Example 3.1. We leave out the query node G because the edges to connect it would just cram the picture.

causal self-attention layer with 8 heads and 64 keys, followed by a cross attention layer with the same parameters. After these layers, the decoder portion of the VAE is used to generate the images. The NeSy-MM model used multiple smaller networks as it is naturally decomposed. Concretely, there is a small convolutional neural network that classifies the initial location of the agent from the first image into either 5 or 10 classes, depending on the grid size. Additionally, a small convolutional encoder network encodes the same initial image into a two-dimensional Gaussian distribution. The NeSy-MM moves the location of the agent according to the rules of MiniHack for a given set of actions, resulting in an estimated distribution for the agent at every subsequent time step. A final convolutional decoder, the same as used by the transformer architecture, then generates an image from the encoding of the initial image together with the planned location of the agent for every time step. The deep Markov model (Deep-MM) uses the exact same setup, only replacing the logical transitions by small neural networks with two hidden layers of size 64 and 32. Train, test, and validation sets contain, respectively 5000, 1000, and 500 trajectories.

Setup and hyperparameters. We ran the experiments on an RTX 3080 Ti (12GB) GPU coupled with an Intel Xeon Gold 6230R CPU @ 2.10GHz and 256 GB of RAM. All experiments were repeated 5 times on this setup for our method and all baselines. Results are reported using averages and standard errors. The hyperparameters, being the β value of the variational optimisation criterion, the number of epochs to train and the learning rate, were obtained via a separate grid search using a held-out validation set. The specific optimal value for the learning rate was 0.001 for most methods and grid sizes. The only exceptions were the NeSy-MMs on grid size 5 and the variational transformers, where a learning rate of 0.0005 and 0.0001 proved optimal, respectively. Adam (Kingma & Ba, 2015) was used as the optimiser for all methods. The optimal values for β varied more from method to method. The variational transformer performed best with $\beta = 10$ and $\beta = 100$ on grid size 5 and 10, respectively. For the deep Markov model, these values were 125 and 2, while NeSy-MMs performed best with 150 and 10 on the smaller and larger grid sizes. The number of epochs is again different per method. Transformers needed 100 epochs to reach good performance on both grid sizes, while the deep Markov models and NeSy-MMs optimised best for 25 epochs. Finally, the deep Markov models and NeSy-MMs also have a number of samples used for their particle filters, which we set to 1000 and 100, respectively.