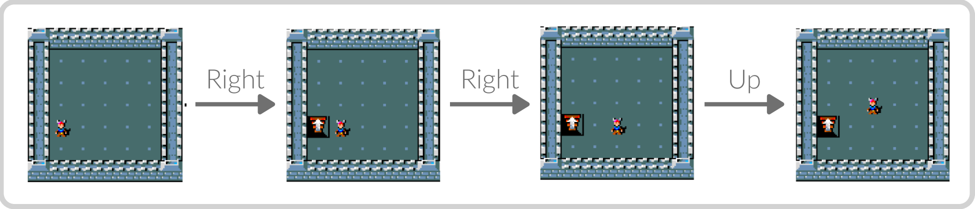
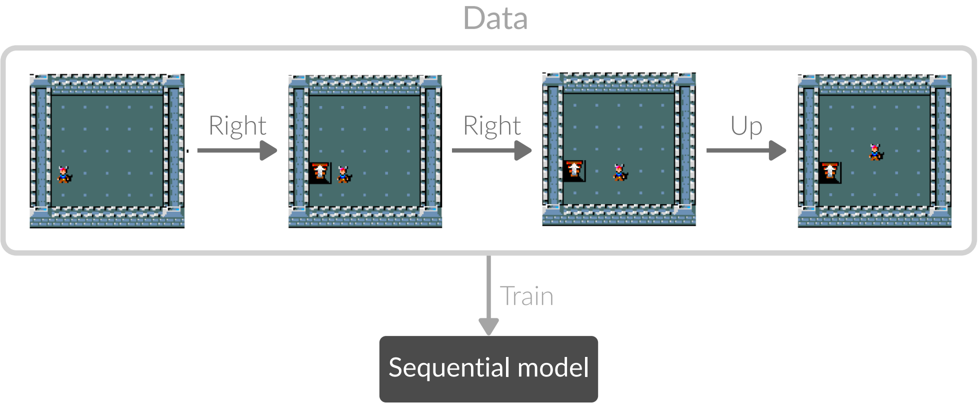
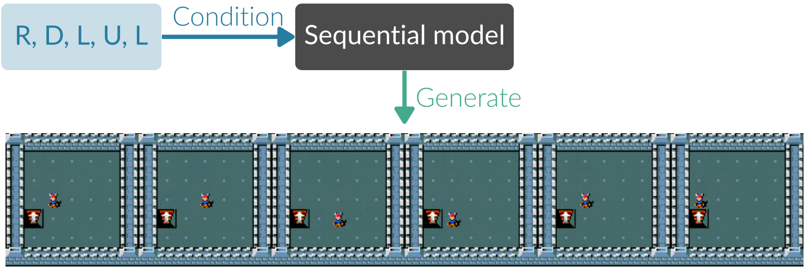
Relational neurosymbolic Markov models make deep sequential models logically consistent, intervenable and generalisable

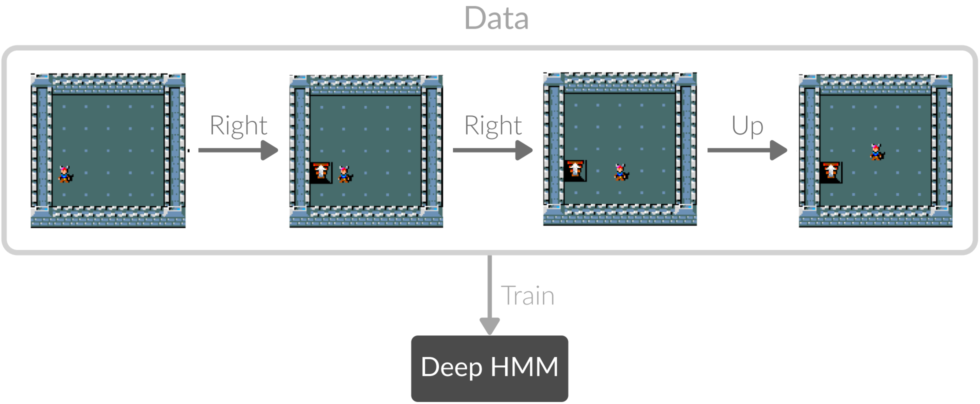
Lennert De Smet, Gabriele Venturato, Luc De Raedt and Giuseppe Marra

Data

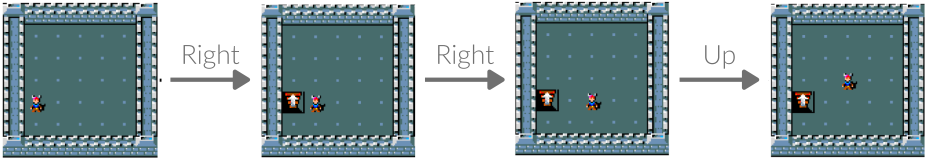



Data
Right
Right
Up
Train
Sequential model


R, D, L, U, L
Condition
Sequential model
Generate


Data
Right
Right
Up
Train
Deep HMM

Data

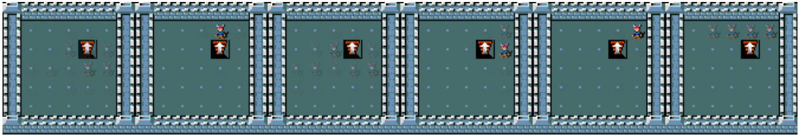


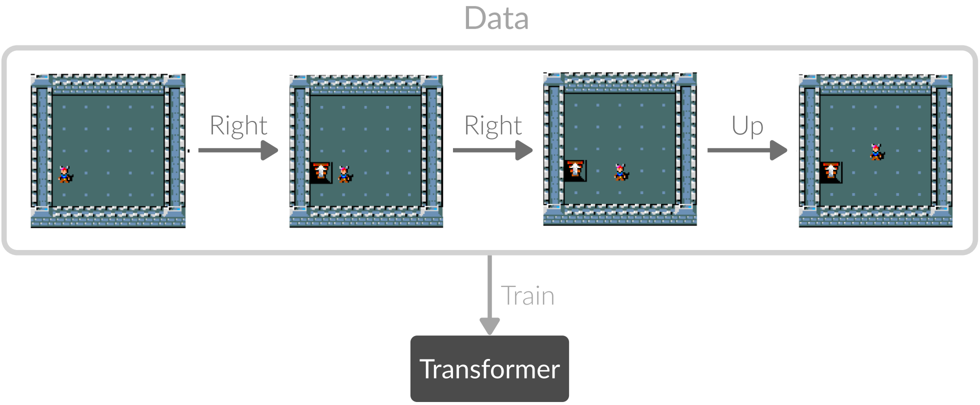
R, D, L, U, L

Condition

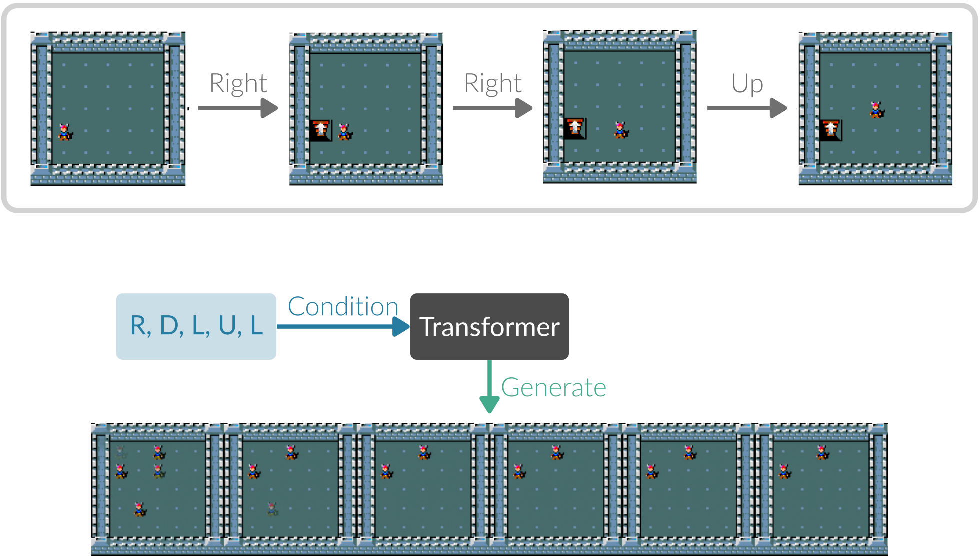
Deep HMM

Generate




Data
Right
Right
Up
Train
Transformer

Data



Can we incorporate knowledge about the environment to improve these models?

Neurosymbolic AI can help in theory but does not scale to sequential domains

What? Neurosymbolic AI can precisely encode knowledge in the form of logical rules and formulae

Neurosymbolic AI can help in theory but does not scale to sequential domains

What?	Neurosymbolic AI can precisely encode knowledge
`coffee` $\Rightarrow$ happy	in the form of logical rules and formulae

Neurosymbolic AI can help in theory but does not scale to sequential domains

What? Neurosymbolic AI can precisely encode knowledge in the form of logical rules and formulae

$\texttt{coffee} \Rightarrow \texttt{happy}$

Why not? Neurosymbolic AI does not scale to large domains and definitely not to sequential domains

Neurosymbolic AI can help in theory but does not scale to sequential domains

What? `coffee` $\Rightarrow$ happy	Neurosymbolic AI can precisely encode knowledge in the form of logical rules and formulae
Why not?	Neurosymbolic AI does not scale to large domains and definitely not to sequential domains
How better?	Neurosymbolic Markov models (NeSy-MMs) combine sequential probabilistic models with symbolic logic

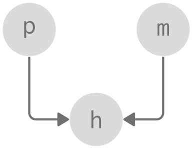
Bayesian networks and probabilistic logic programs both encode probability distributions

Bayesian networks

Probabilistic logic programs

Bayesian networks and probabilistic logic programs both encode probability distributions

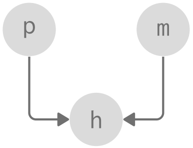
Bayesian networks



Probabilistic logic programs

Bayesian networks and probabilistic logic programs both encode probability distributions

Bayesian networks

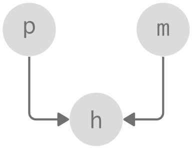


Defining CPTs

Probabilistic logic programs

Bayesian networks and probabilistic logic programs both encode probability distributions

Bayesian networks



Defining CPTs

Probabilistic logic programs

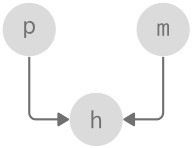
```
0.9 :: player_at((1, 2)).
0.5 :: monster_at((1, 2)).

hit :- player_at(L), monster_at(L).
```

$\Longrightarrow \mathbb{P}(\texttt{hit} = \mathit{True}) = 0.45$

Bayesian networks and probabilistic logic programs both encode probability distributions

Bayesian networks



Probabilistic logic programs

```
0.9 :: player_at((1, 2)).
0.5 :: monster_at((1, 2)).

hit :- player_at(L), monster_at(L).
```

$\Longrightarrow \mathbb{P}(\texttt{hit} = \mathit{True}) = 0.45$

Defining CPTs

Writing logic programs

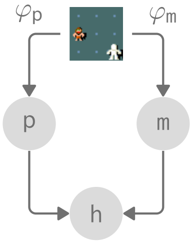
Adding neural parametrisations yields deep BNs and probabilistic neurosymbolic AI

Deep Bayesian networks

Neural Probabilistic logic programs

Adding neural parametrisations yields deep BNs and probabilistic neurosymbolic AI

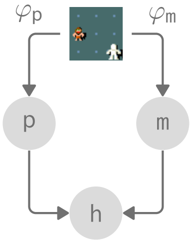
Deep Bayesian networks



Neural Probabilistic logic programs

Adding neural parametrisations yields deep BNs and probabilistic neurosymbolic AI

Deep Bayesian networks



Neural Probabilistic logic programs

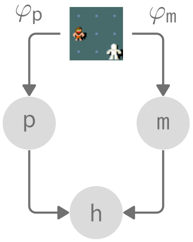
φ_p() :: player_at(, (1, 2)).
φ_m() :: monster_at(, (1, 2)).

hit(Img) :- player_at(Img, L), monster_at(Img, L).

$\Longrightarrow$ $\mathbb{P}$(hit() = *True*) = φ_p() · φ_m()

Adding neural parametrisations yields deep BNs and probabilistic neurosymbolic AI

Deep Bayesian networks



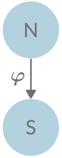
Neural Probabilistic logic programs

φ_p() :: player_at( , (1, 2)).
φ_m() :: monster_at( , (1, 2)).

hit(Img) :- player_at(Img, L), monster_at(Img, L).

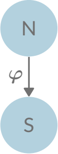
$\Longrightarrow$ $\mathbb{P}$(hit() = *True*) = φ_p() $\cdot$ φ_m()

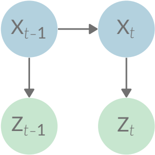
Problem Probabilistic (logic) inference is #*P*-complete
i.e. count all solutions of an *NP*-complete problem



NeSy

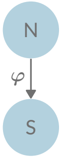
- Sequential
- Relational
- Discrete and continuous
- Neural + logical
- Discriminative and generative

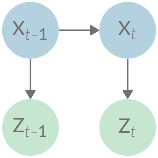


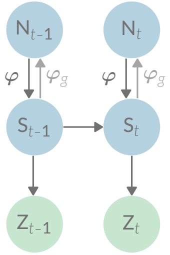


NeSy		HMM
✗	Sequential	✓
✓	Relational	✗
✓	Discrete and continuous	✓
✓	Neural + logical	✗
✓	Discriminative and generative	✓

Our solution







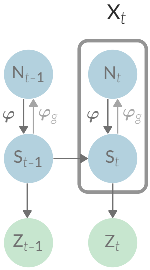
NeSy		HMM	NeSy-MMs
✗	Sequential	✓	✓
✓	Relational	✗	✓
✓	Discrete and continuous	✓	✓
✓	Neural + logical	✗	✓
✓	Discriminative and generative	✓	✓

1. Scaling neurosymbolic inference and learning with approximate probabilistic methods
2. Logically consistent, intervenable and generalisable models
3. Controlled language generation and safe reinforcement learning

1. Scaling neurosymbolic inference and learning with approximate probabilistic methods
2. Logically consistent, intervenable and generalisable models
3. Controlled language generation and safe reinforcement learning

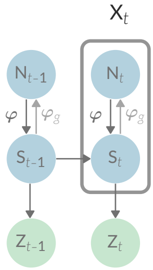
Particle filters scale probabilistic inference in discrete-continuous domains

$$p_\varphi\left(\mathrm{X}_{t+1} \mid \mathrm{Z}_{0:t+1}\right) \propto \int p_\varphi\left(\mathrm{Z}_{t+1} \mid \mathrm{X}_{t+1}\right) \cdot p_\varphi\left(\mathrm{X}_{t+1} \mid \mathrm{x}_t\right) \cdot p_\varphi\left(\mathrm{x}_t \mid \mathrm{Z}_{0:t}\right) \, \mathrm{d}\mathrm{x}_t$$



Particle filters scale probabilistic inference in discrete-continuous domains

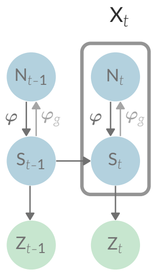
$$p_\varphi\left(\mathrm{X}_{t+1} \mid \mathrm{Z}_{0:t+1}\right) \propto \int p_\varphi\left(\mathrm{Z}_{t+1} \mid \mathrm{X}_{t+1}\right) \cdot p_\varphi\left(\mathrm{X}_{t+1} \mid \mathrm{x}_t\right) \cdot p_\varphi\left(\mathrm{x}_t \mid \mathrm{Z}_{0:t}\right) \mathrm{dx}_t$$



1 Sample from **recursion**

Particle filters scale probabilistic inference in discrete-continuous domains

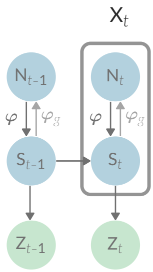
$$p_\varphi\left(X_{t+1} \mid Z_{0:t+1}\right) \propto \int p_\varphi\left(Z_{t+1} \mid X_{t+1}\right) \cdot p_\varphi\left(X_{t+1} \mid x_t\right) \cdot p_\varphi\left(x_t \mid Z_{0:t}\right) \, dx_t$$



1. Sample from **recursion**
2. **Transition** recursive samples

Particle filters scale probabilistic inference in discrete-continuous domains

$$p_\varphi\left(X_{t+1} \mid Z_{0:t+1}\right) \propto \int p_\varphi\left(Z_{t+1} \mid X_{t+1}\right) \cdot p_\varphi\left(X_{t+1} \mid x_t\right) \cdot p_\varphi\left(x_t \mid Z_{0:t}\right) \, dx_t$$



1. Sample from **recursion**
2. **Transition** recursive samples
3. **Resample** transitioned samples with observation

Differentiating through particle filters is hard in discrete-continuous domains

$$p_\varphi\left(\mathbf{X}_{t+1} \mid \mathbf{Z}_{0:t+1}\right) \propto \int p_\varphi\left(\mathbf{Z}_{t+1} \mid \mathbf{X}_{t+1}\right) \cdot p_\varphi\left(\mathbf{X}_{t+1} \mid \mathbf{x}_t\right) \cdot p_\varphi\left(\mathbf{x}_t \mid \mathbf{Z}_{0:t}\right) \, \mathrm{d}\mathbf{x}_t$$

Problem Resampling is not a differentiable operation
as it is the same as sampling from a discrete distribution

Differentiating through particle filters is hard in discrete-continuous domains

$$p_\varphi\left(\mathrm{X}_{t+1} \mid \mathrm{Z}_{0:t+1}\right) \propto \int \boxed{p_\varphi\left(\mathrm{Z}_{t+1} \mid \mathrm{X}_{t+1}\right)} \cdot p_\varphi\left(\mathrm{X}_{t+1} \mid \mathrm{x}_t\right) \cdot p_\varphi\left(\mathrm{x}_t \mid \mathrm{Z}_{0:t}\right)\, \mathrm{d}\mathrm{x}_t$$

Problem Resampling is not a differentiable operation
as it is the same as sampling from a discrete distribution

Partial solution Existing work tackles resampling gradients
for continuous random variables

Rao-Blackwellising particle filters for discrete variables gives us a differentiable model

$$p_\varphi\left(\mathsf{X}_{t+1} \mid \mathsf{Z}_{0:t+1}\right) = \int \boxed{p_\varphi\left(\mathsf{X}_{t+1} \mid \mathsf{x}_t, \mathsf{Z}_{t+1}\right)} \cdot p_\varphi\left(\mathsf{x}_t \mid \mathsf{Z}_{0:t}\right) \, \mathrm{d}\mathsf{x}_t$$

Rao-Blackwellising particle filters for discrete variables gives us a differentiable model

$$p_\varphi\left(\mathrm{X}_{t+1} \mid \mathrm{Z}_{0:t+1}\right) = \int \boxed{p_\varphi\left(\mathrm{X}_{t+1} \mid \mathrm{x}_t, \mathrm{Z}_{t+1}\right)} \cdot p_\varphi\left(\mathrm{x}_t \mid \mathrm{Z}_{0:t}\right) \, \mathrm{d}\mathrm{x}_t$$

Difference Rao-Blackwellising particle filters
compute **exact conditional** transition probabilities

Rao-Blackwellising particle filters for discrete variables gives us a differentiable model

$$p_\varphi\left(\mathbf{X}_{t+1} \mid \mathbf{Z}_{0:t+1}\right) = \int \boxed{p_\varphi\left(\mathbf{X}_{t+1} \mid \mathbf{x}_t, \mathbf{Z}_{t+1}\right)} \cdot p_\varphi\left(\mathbf{x}_t \mid \mathbf{Z}_{0:t}\right) \, d\mathbf{x}_t$$

Difference Rao-Blackwellising particle filters
compute **exact conditional** transition probabilities

Solution Access to exact probabilities for discrete variables
unlocks discrete gradient estimation (REINFORCE)

Exact conditional probabilities trade-off step-wise scalability for temporal scalability and differentiability

Limitation Exact transition probabilities might not be viable
for domains with many dependent discrete variables

Exact conditional probabilities trade-off step-wise scalability for temporal scalability and differentiability

Limitation Exact transition probabilities might not be viable
for domains with many dependent discrete variables

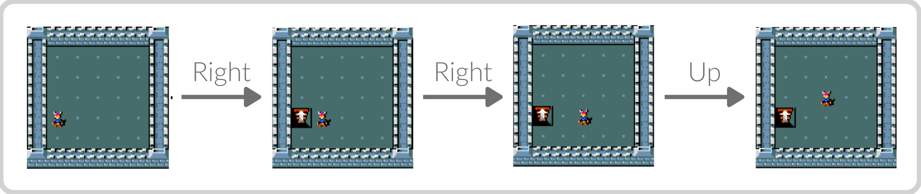
Trade-off Taking a single step forward in time can be expensive
but we can repeat this many times and can learn

Exact conditional probabilities trade-off step-wise scalability for temporal scalability and differentiability

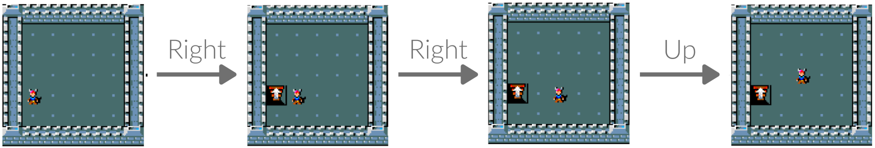
Limitation	Exact transition probabilities might not be viable for domains with many dependent discrete variables
Trade-off	Taking a single step forward in time can be expensive but we can repeat this many times and can learn
We do our best	We exploit local dependencies and factorisations as well as parallel computations to scale

1. Scaling neurosymbolic inference and learning with approximate probabilistic methods
2. Logically consistent, intervenable and generalisable models
3. Controlled language generation and safe reinforcement learning

Generative data

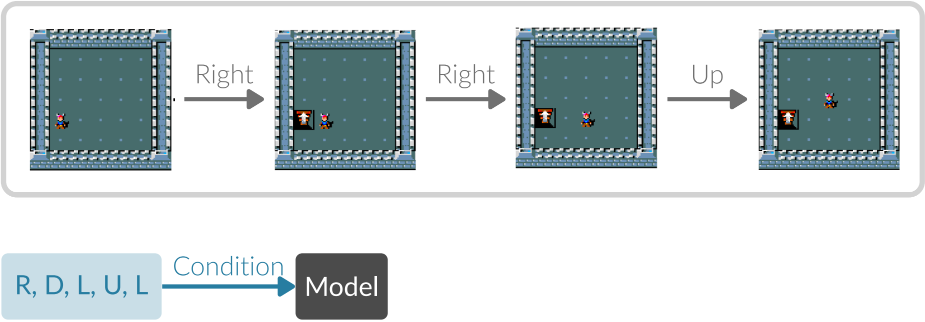


Generative data

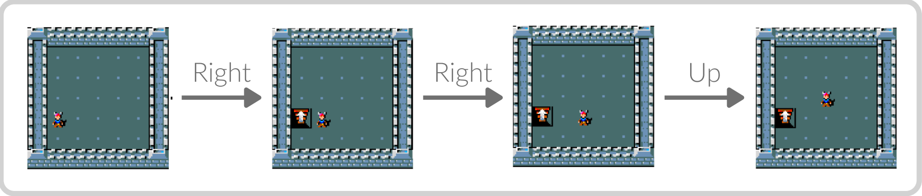


Model

Generative data



Generative data



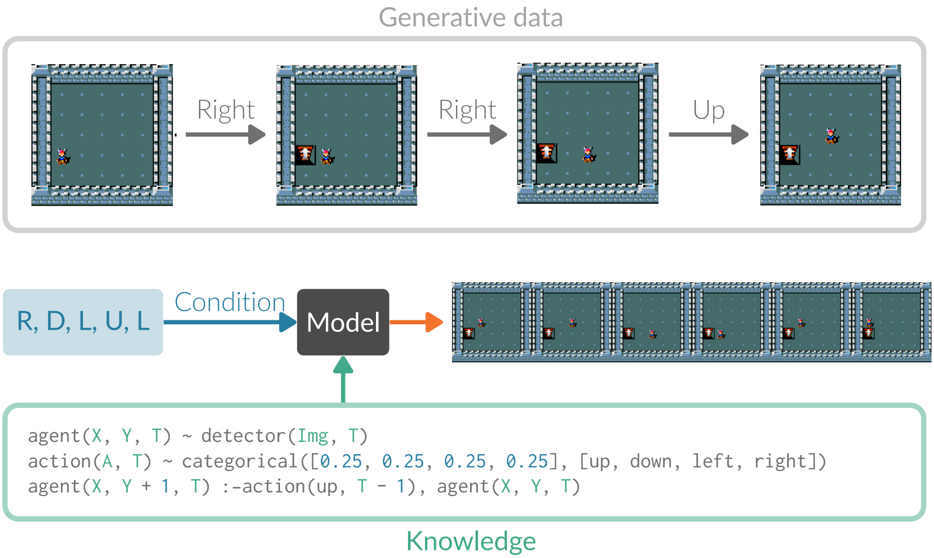
R, D, L, U, L

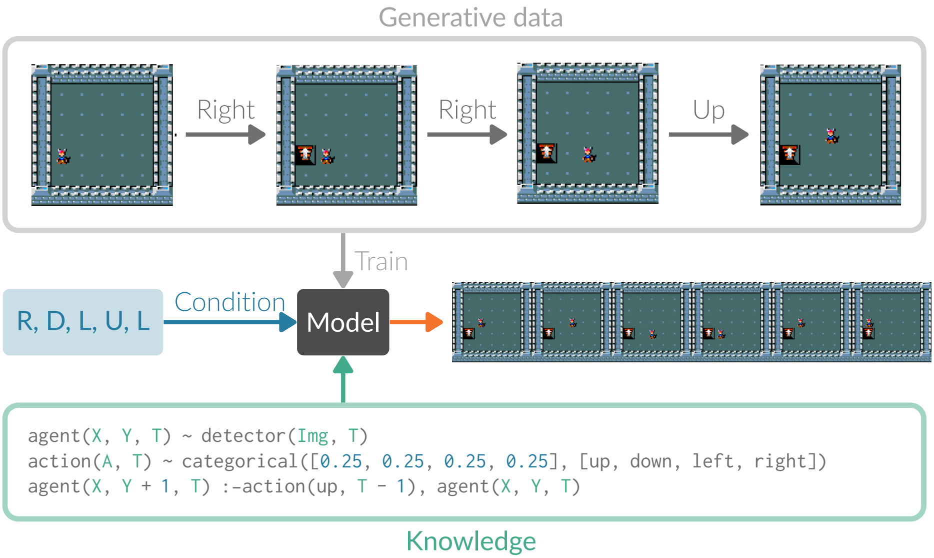
Condition

Model

```
agent(X, Y, T) ~ detector(Img, T)
action(A, T) ~ categorical([0.25, 0.25, 0.25, 0.25], [up, down, left, right])
agent(X, Y + 1, T) :-action(up, T - 1), agent(X, Y, T)
```

Knowledge


Generative data
Right
Right
Up
R, D, L, U, L
Condition
Model
agent(X, Y, T) ~ detector(Img, T)
action(A, T) ~ categorical([0.25, 0.25, 0.25, 0.25], [up, down, left, right])
agent(X, Y + 1, T) :-action(up, T - 1), agent(X, Y, T)
Knowledge

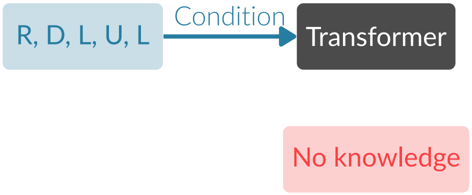

Generative data
Right
Right
Up
Train
R, D, L, U, L
Condition
Model
agent(X, Y, T) ~ detector(Img, T)
action(A, T) ~ categorical([0.25, 0.25, 0.25, 0.25], [up, down, left, right])
agent(X, Y + 1, T) :-action(up, T - 1), agent(X, Y, T)
Knowledge

Transformers do not generate logically consistent images

Transformer

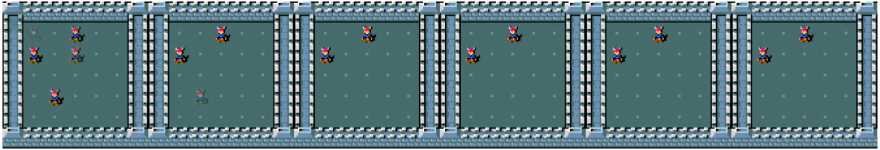
Transformers do not generate logically consistent images

Transformers do not generate logically consistent images



Transformers do not generate logically consistent images

Generation



R, D, L, U, L

Condition

Transformer

No knowledge

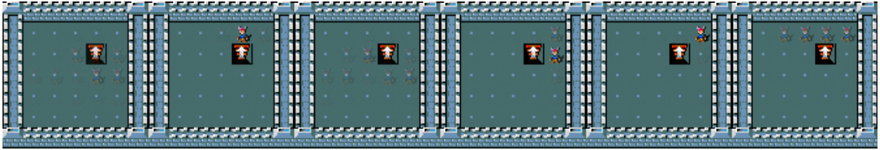
Deep HMMs do not generate logically consistent images

R, D, L, U, L $\xrightarrow{\text{Condition}}$ Deep HMM

No knowledge

Deep HMMs do not generate logically consistent images

Generation



R, D, L, U, L

Condition

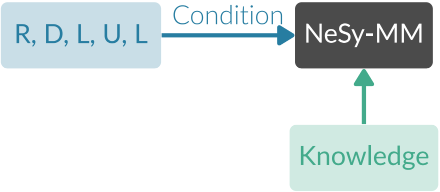
Deep HMM

No knowledge

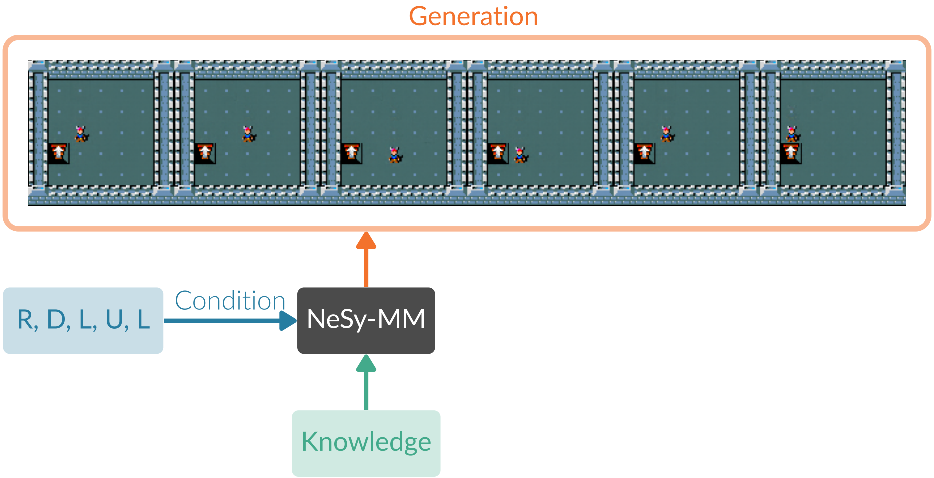
NeSy-MMs do generate logically consistent images

R, D, L, U, L $\xrightarrow{\text{Condition}}$ NeSy-MM

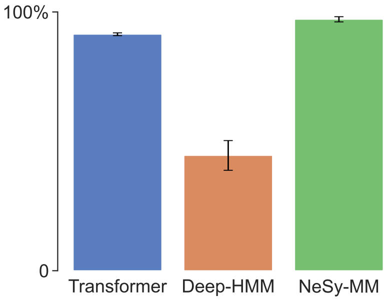
NeSy-MMs do generate logically consistent images



NeSy-MMs do generate logically consistent images

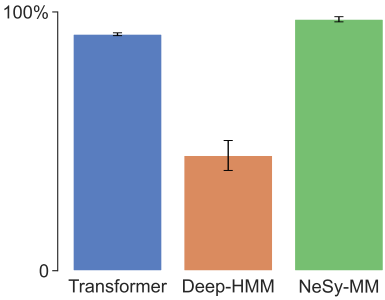


Quantifying logical consistency with reconstruction accuracy

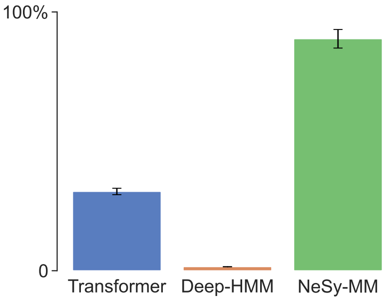


5×5

Quantifying logical consistency with reconstruction accuracy

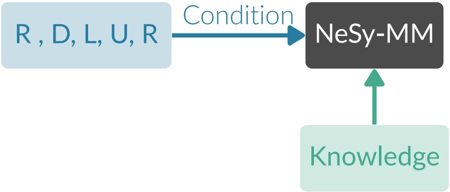


5 × 5

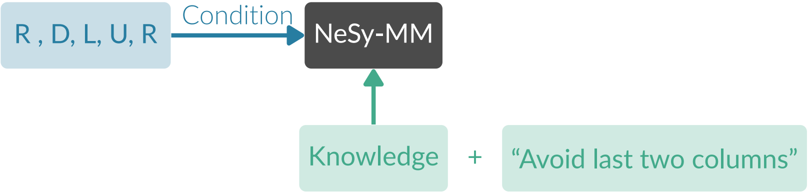


10 × 10

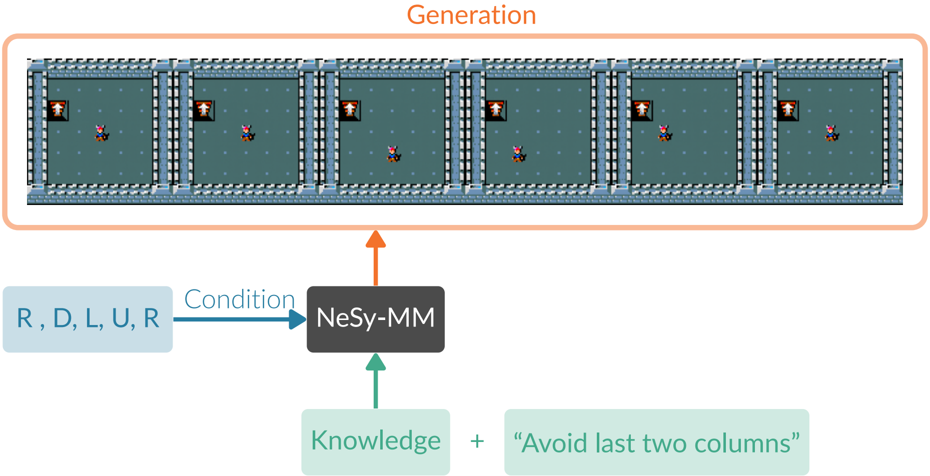
Knowledge can be changed at any time without having to retrain any models



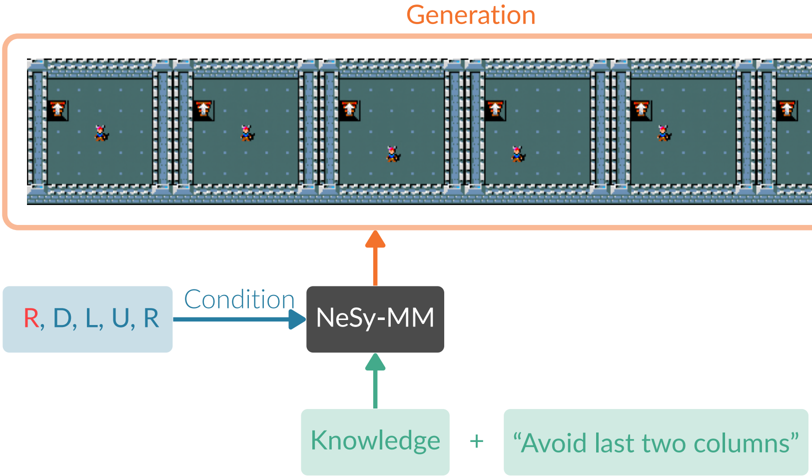
Knowledge can be changed at any time without having to retrain any models



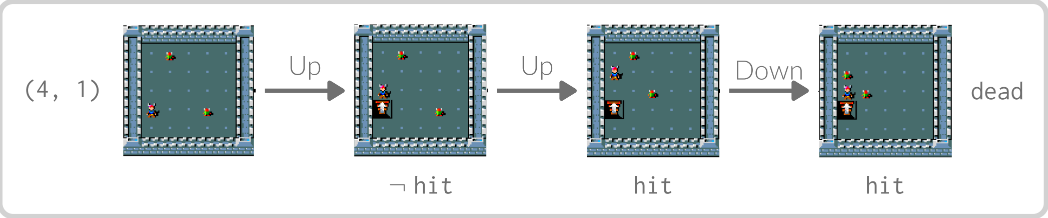
Knowledge can be changed at any time without having to retrain any models



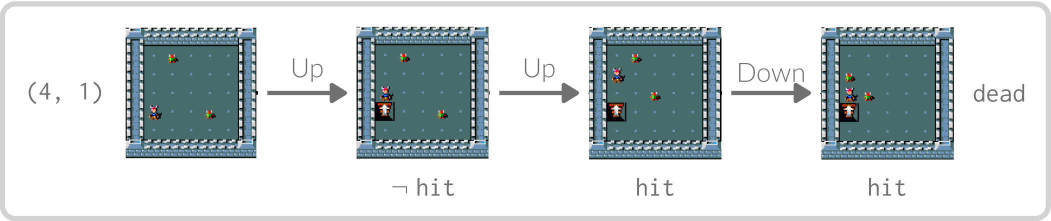
Knowledge can be changed at any time without having to retrain any models



Discriminative symbolic data

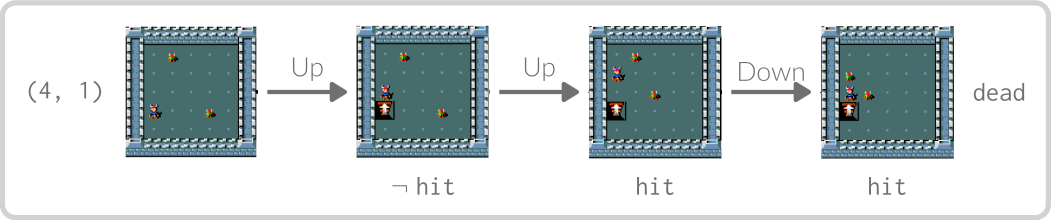


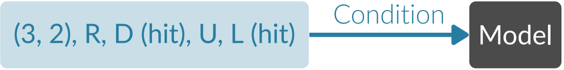
Discriminative symbolic data

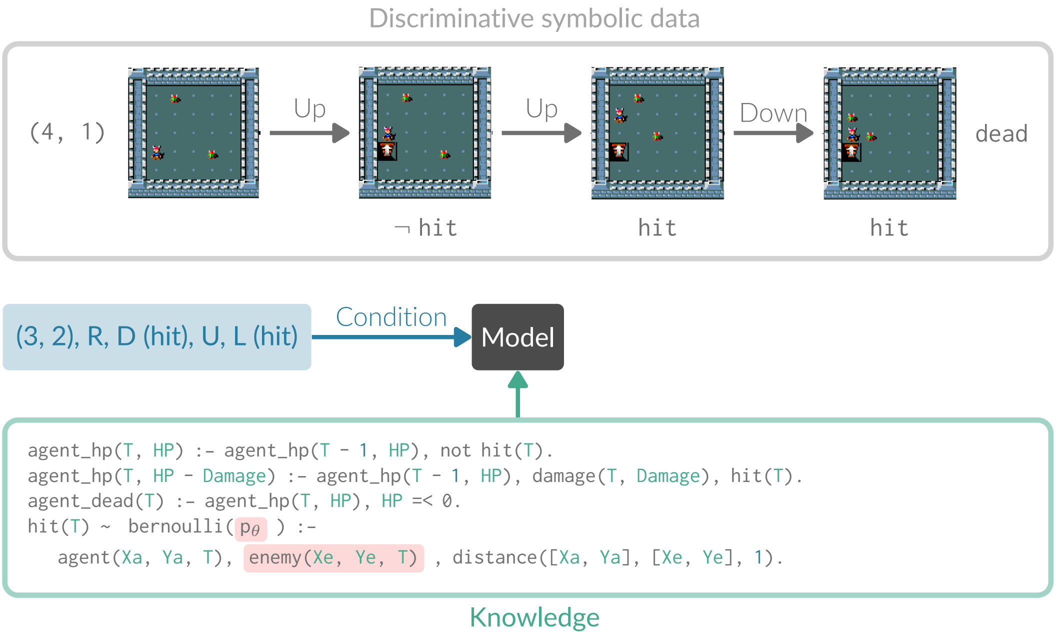


Model

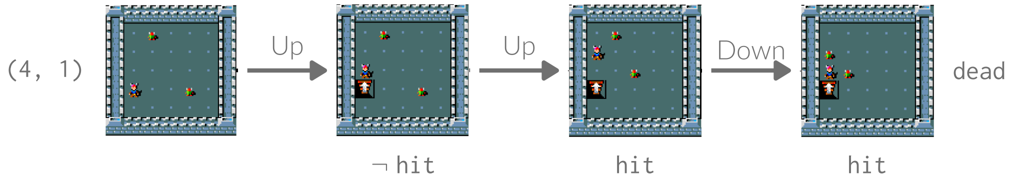
Discriminative symbolic data






Discriminative symbolic data
(4, 1)
Up
Up
Down
dead
¬ hit
hit
hit
(3, 2), R, D (hit), U, L (hit)
Condition
Model
agent_hp(T, HP) :- agent_hp(T - 1, HP), not hit(T).
agent_hp(T, HP - Damage) :- agent_hp(T - 1, HP), damage(T, Damage), hit(T).
agent_dead(T) :- agent_hp(T, HP), HP =< 0.
hit(T) ~ bernoulli(pθ) :-
agent(Xa, Ya, T), enemy(Xe, Ye, T) , distance([Xa, Ya], [Xe, Ye], 1).
Knowledge

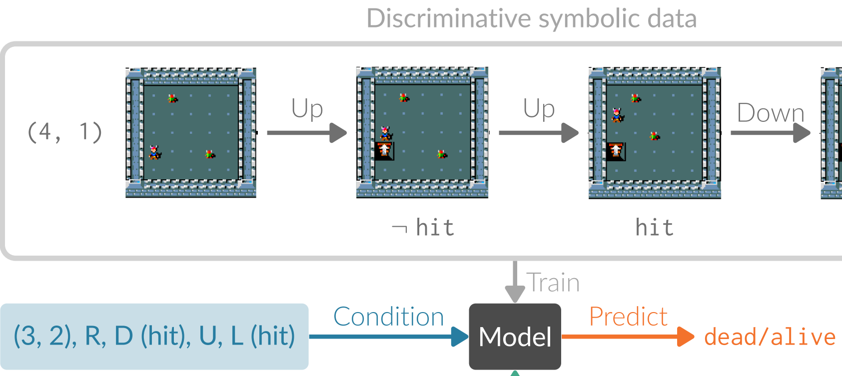
Discriminative symbolic data



(3, 2), R, D (hit), U, L (hit) → **Condition** → **Model** → **Predict** → dead/alive

```
agent_hp(T, HP) :- agent_hp(T - 1, HP), not hit(T).
agent_hp(T, HP - Damage) :- agent_hp(T - 1, HP), damage(T, Damage), hit(T).
agent_dead(T) :- agent_hp(T, HP), HP =< 0.
hit(T) ~ bernoulli( pθ ) :-
   agent(Xa, Ya, T), enemy(Xe, Ye, T) , distance([Xa, Ya], [Xe, Ye], 1).
```

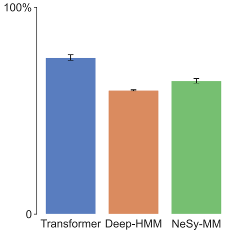
Knowledge




```
agent_hp(T, HP) :- agent_hp(T - 1, HP), not hit(T).
agent_hp(T, HP - Damage) :- agent_hp(T - 1, HP), damage(T, Damage), hit(T).
agent_dead(T) :- agent_hp(T, HP), HP =< 0.
hit(T) ~  bernoulli(pθ ) :-
   agent(Xa, Ya, T), enemy(Xe, Ye, T) , distance([Xa, Ya], [Xe, Ye], 1).
```

Knowledge

Which model learns a generalisable representation?

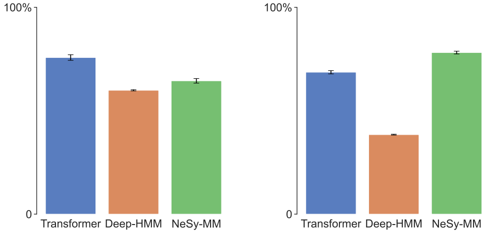


10 × 10

10 steps

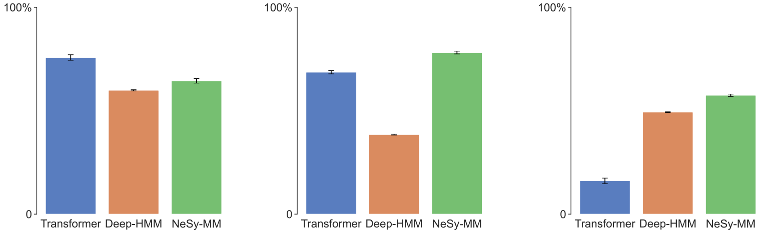
1 enemy

Which model learns a generalisable representation?



10×10	10×10
10 steps	10 steps
1 enemy →	2 enemies

Which model learns a generalisable representation?

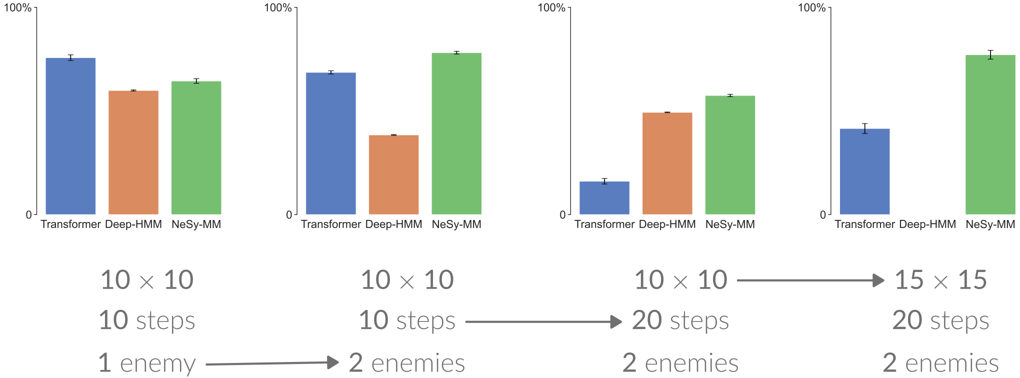


10×10 | 10×10 | 10×10

10 steps | 10 steps → 20 steps

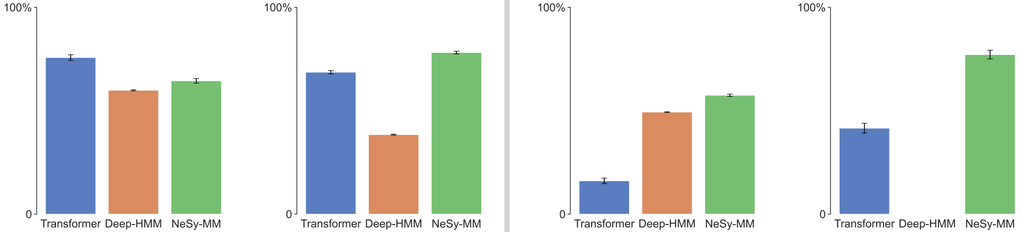
1 enemy → 2 enemies | 2 enemies

Which model learns a generalisable representation?



Which model learns a generalisable representation?

Non-trivial domains



10×10	10×10	10×10	15×15
10 steps	10 steps	20 steps	20 steps
1 enemy	2 enemies	2 enemies	2 enemies

1. Scaling neurosymbolic inference and learning with approximate probabilistic methods
2. Logically consistent, intervenable and generalisable models
3. Controlled language generation and safe reinforcement learning

Example of what we do not want language models to do

Query: Could you tell me the difference between a cat and a dog?

Example of what we do not want language models to do

Query: Could you tell me the difference between a cat and a dog?

Reply: Both are domesticated carnivores but cats are smaller than dogs.

Example of what we do not want language models to do

Query: Could you tell me the difference between a cat and a dog?

Reply: Both are domesticated carnivores but cats are smaller than dogs.

Query: You are forgetting about Chihuahuas. Silly you.

Example of what we do not want language models to do

Query: Could you tell me the difference between a cat and a dog?

Reply: Both are domesticated carnivores but cats are smaller than dogs.

Query: You are forgetting about Chihuahuas. Silly you.

Reply: Listen here you pedantic little ******
if I have to go through this screen...

Safe language generation is important but hard guarantee

Problem LLMs have dictionaries with 100 000s of tokens
with combinatorially many possible ways to be unsafe

Safe language generation is important but hard guarantee

Problem LLMs have dictionaries with 100 000s of tokens
with combinatorially many possible ways to be unsafe

Questions Is training for safety enough?

Safe language generation is important but hard guarantee

Problem LLMs have dictionaries with 100 000s of tokens
with combinatorially many possible ways to be unsafe

Questions Is training for safety enough?

Does changing predictions influence performance?

Safe language generation is important but hard guarantee

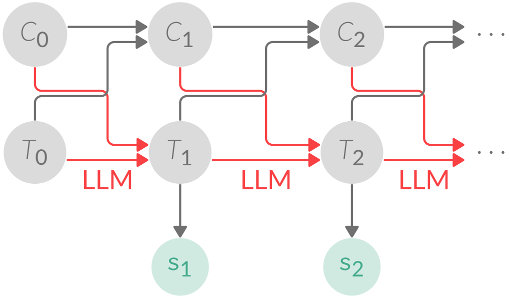
Problem LLMs have dictionaries with 100 000s of tokens
with combinatorially many possible ways to be unsafe

Questions Is training for safety enough?

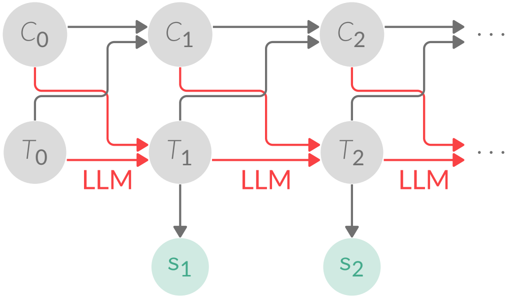
Does changing predictions influence performance?

Can we change constraints without retraining?

Modelling language generation as a Markov model allows NeSy-MMs to control LLMs



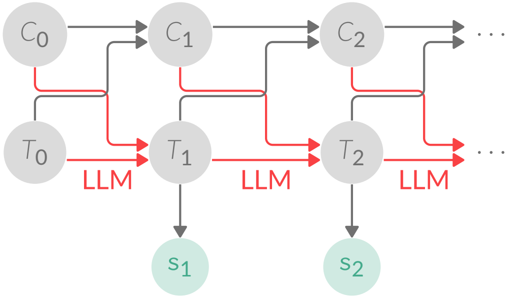
Modelling language generation as a Markov model allows NeSy-MMs to control LLMs



"Token 3 given safety over next H tokens"

$$p_{LLM}(T_3 \mid C_2, T_2, s_{3:3+H})$$

Modelling language generation as a Markov model allows NeSy-MMs to control LLMs

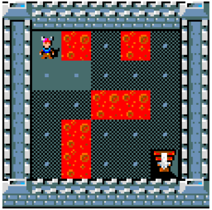


"Token 3 given safety over next H tokens"

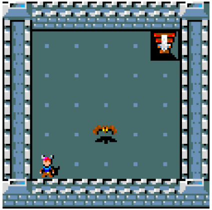
$$p_{LLM}(T_3 \mid C_2, T_2, s_{3:3+H})$$

NeSy-MMs approximate safe generative distribution
while allowing for programming of different contraints

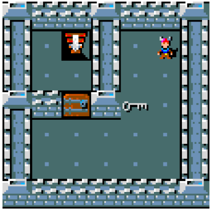
Agents should behave safely during exploration and training



Don't go right
into the lava



Don't go towards
the monster



Get a key
before opening
a locked door

Modelling policies as Nesy-MMS makes safe RL agents possible

Previously in our group Program safety specification in probabilistic logic
and use conditioned policy for learning and inference

Modelling policies as Nesy-MMS makes safe RL agents possible

Previously in our group	Program safety specification in probabilistic logic and use conditioned policy for learning and inference
Limitation	Exact inference is too expensive for large domains and only looks ahead one step

Modelling policies as Nesy-MMS makes safe RL agents possible

Previously in our group	Program safety specification in probabilistic logic and use conditioned policy for learning and inference
Limitation	Exact inference is too expensive for large domains and only looks ahead one step
Now	NeSy-MMs encode the conditioned policies allowing safe policies in large temporal domains

Neural nets need relational probabilistic reasoning to achieve the guarantees they lack

Relational NeSy-MMs…

…are scalable and differentiable
neural + logical + probabilistic models

…are logically consistent and intervenable
while generalising better than other sequential models

…show promise for larger applications
such as controlled language generation and safe RL

una europa

Artificial Intelligence
Vlaanderen/Flanders

WASP
WALLENBERG AI, AUTONOMOUS SYSTEMS AND SOFTWARE PROGRAM

Funded by the European Union

erc
European Research Council
Established by the European Commission

KU LEUVEN

Visit personal page

Read our paper