UNIVERSITY OF TECHNOLOGY
IN THE EUROPEAN CAPITAL OF CULTURE
CHEMNITZ

Deep Reinforcement Learning

Model-based RL, augmentation

Julien Vitay

Professur für Künstliche Intelligenz - Fakultät für Informatik

Summary of model-free methods

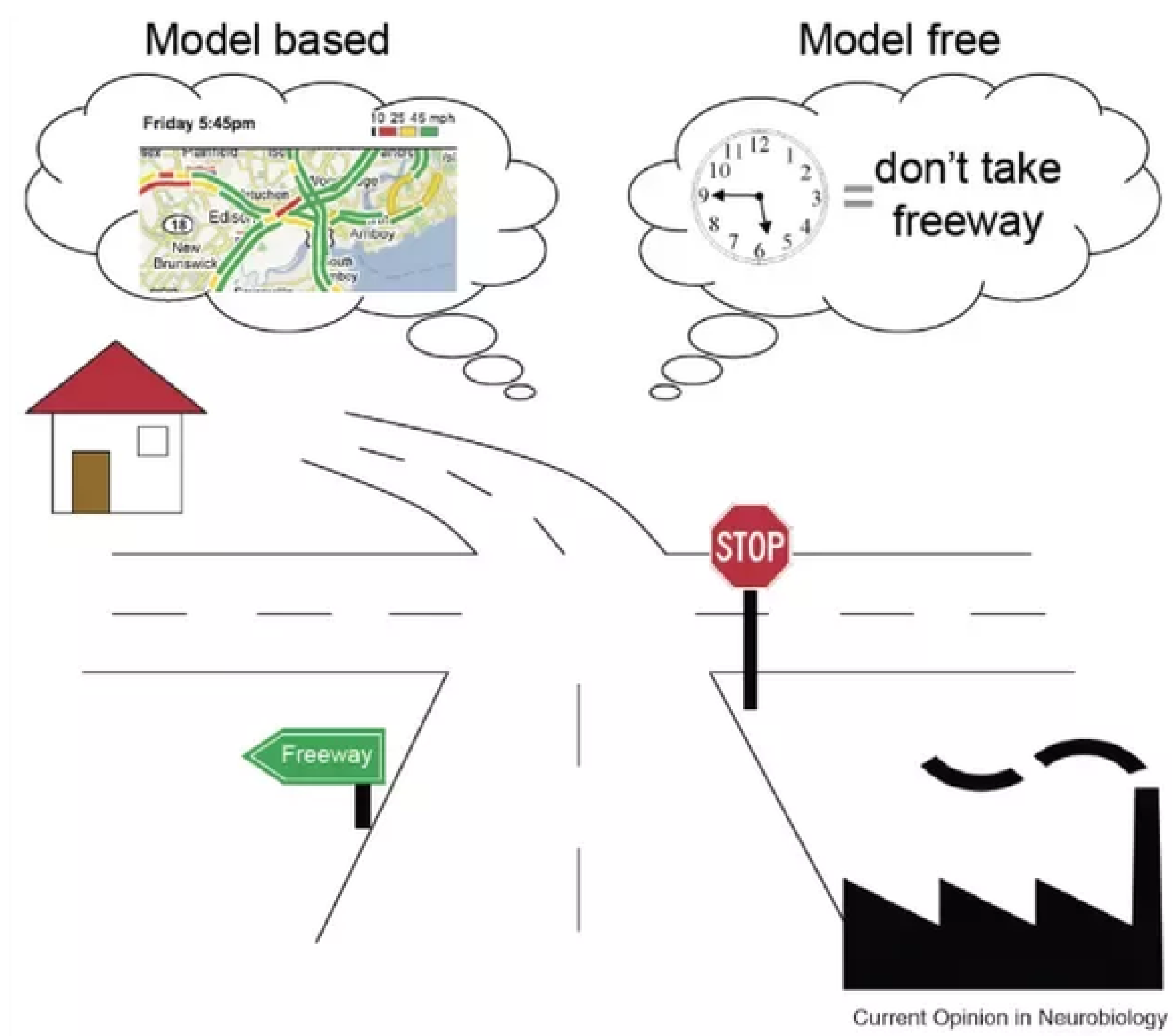
- Model-free methods allow to solve MDPs without knowing anything about the model.

Algorithm	Action Space	Exploration	On- or Off-Policy	Sample Efficiency	Learning Stability	Policy Optimality
DQN	Discrete	ϵ-greedy	Off-policy	+	-	+
A3C	Discrete and Continuous	Gaussian policy	On-policy	-	-	+
DDPG	Continuous	Exploration noise	Off-policy	+	-	+
TD3	Continuous	Exploration noise	Off-policy	++	+	+++
PPO	Discrete and Continuous	Gaussian policy	On-policy	+	+++	+++
SAC	Continuous	Soft policy	Off-policy	+++	-	+++

- In practice, you should use PPO if you do not really care about sample efficiency and prefer learning stability (ChatGPT). It works well for both discrete and continuous spaces.
- For continuous ction spaces (robotics), you should prefer TD3 or SAC, TD3 being less computationally expensive, but SAC being more sample efficient.

1 - Model-based RL

Model-free vs. model-based RL



Source: Dayan and Niv (2008) Reinforcement learning: The Good, The Bad and The Ugly. Current Opinion in Neurobiology, Cognitive neuroscience 18:185–196. doi:10.1016/j.conb.2008.08.003

- In **model-free RL** (MF) methods, we do not need to know anything about the dynamics of the environment to start learning a policy:

$$p(s'|s, a) \;\; r(s, a, s)$$

- We just sample transitions (s, a, r, s') from the environment and update either Q-values or a policy network.
- The main advantage is that the agent does not need to "think" when acting: just select the action with highest Q-value (**reflexive behavior**).
- The other advantage is that you can use MF methods on **any** MDP: you do not need to know anything about them.

- MF methods are very slow (**sample complexity**): as they make no assumption, they have to learn everything by trial-and-error from scratch.
- MF methods are **not safe**: it is very hard to use external knowledge to avoid exploring dangerous actions.

Model-free vs. model-based RL

- If you had a **model** of the environment, you could plan ahead (what would happen if I did that?) and speed up learning (do not explore stupid ideas): **model-based RL** (MB).

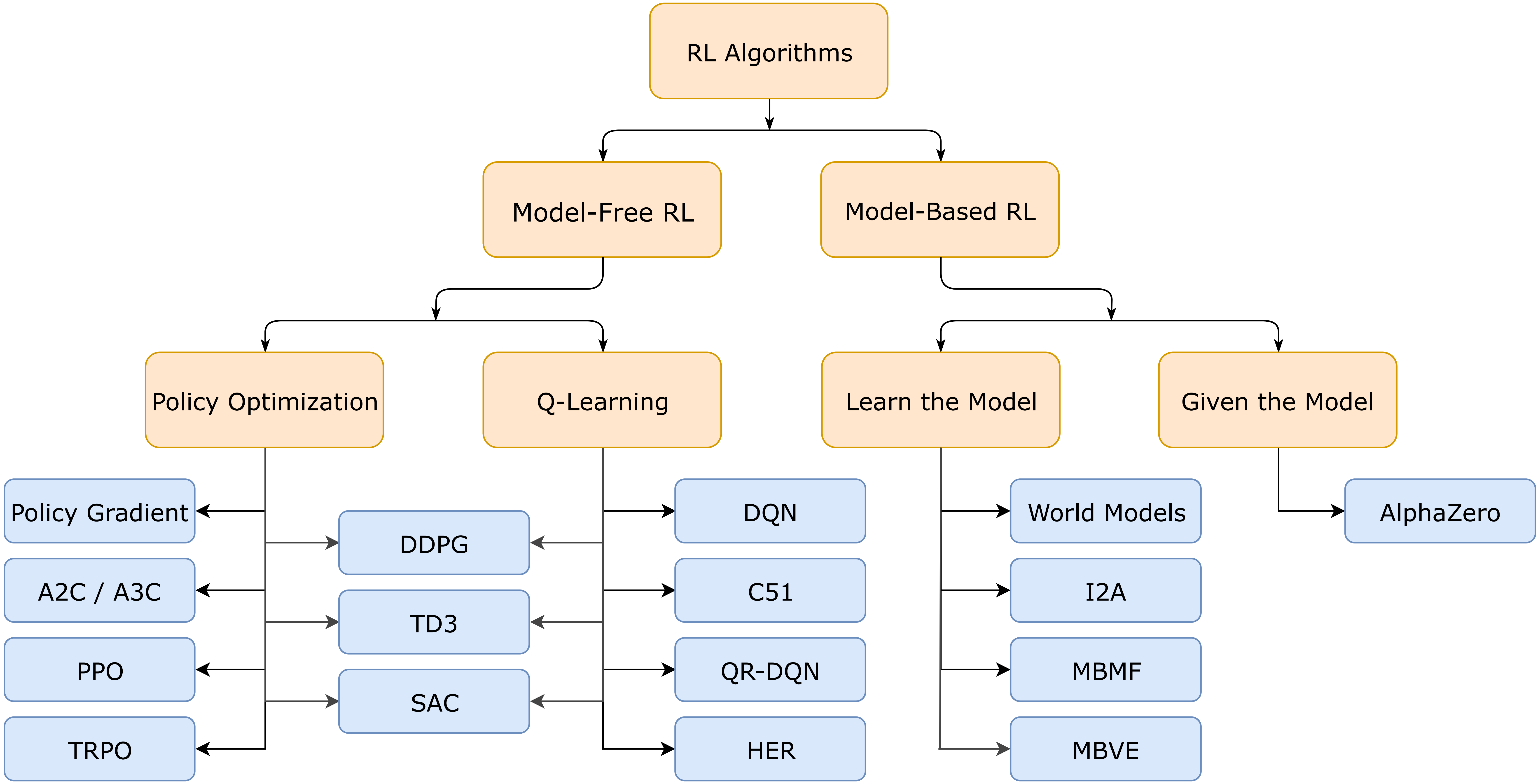
- In chess, players **plan** ahead the possible moves up to a certain horizon and evaluate moves based on their emulated consequences.

Source: https://www.chess.com/article/view/announcing-the-chess-com-gif-maker

- In real-time strategy games, learning the environment (**world model**) is part of the strategy: you do not attack right away.

Source: https://towardsdatascience.com/model-based-reinforcement-learning-cb9e41ff1f0d

Two families of deep RL algorithms



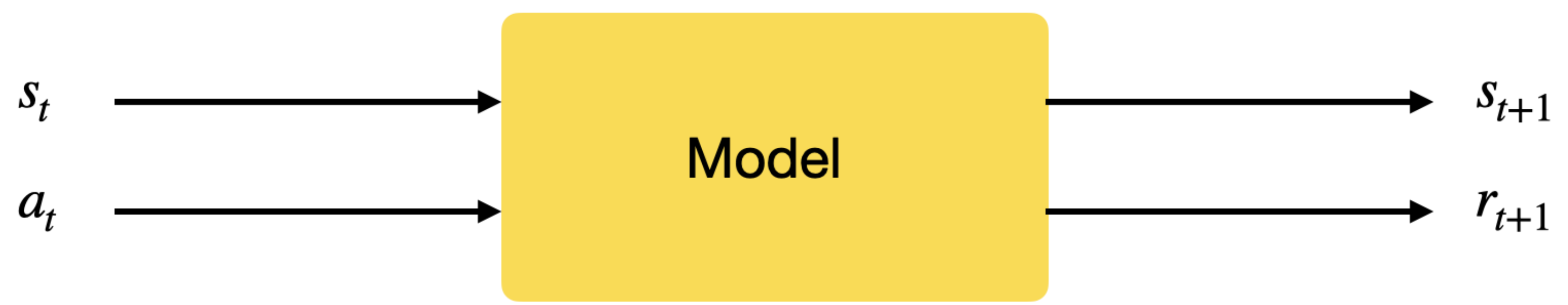
Source: https://github.com/avillemin/RL-Personnal-Notebook

2 - Learning the world model

Learning the world model

- Learning the world model is not complicated in theory.
- We just need to collect *enough* transitions $s_t, a_t, s_{t+1}, r_{t+1}$ using a random agent (or during learning) and train a **supervised** model to predict the next state and reward.

$$s', r = M(s, a)$$

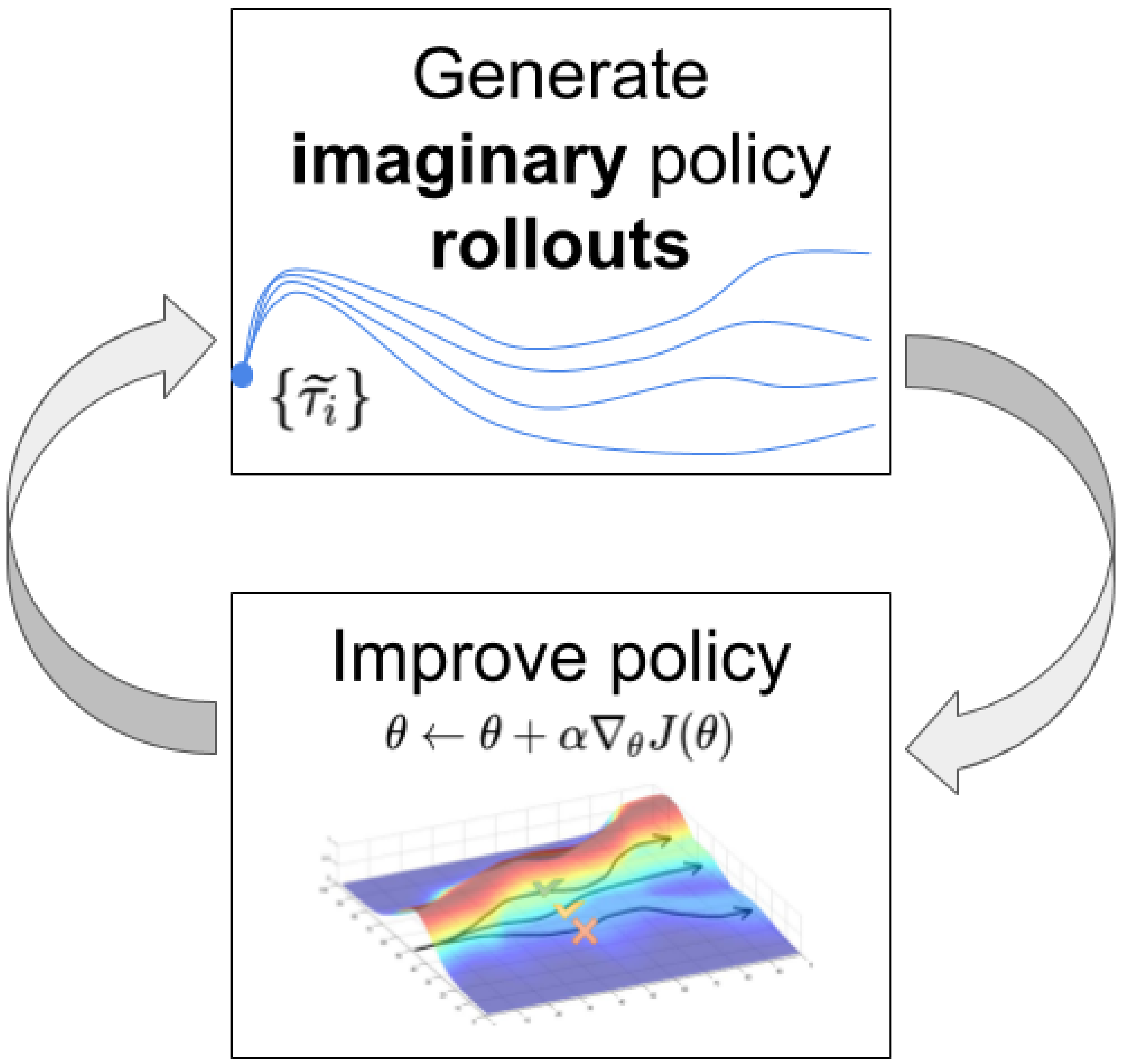


Training in imagination

1. Collect transitions (s, a, r, s') using a (random/expert) policy b and create a dataset $\mathcal{D} = \{(s_k, a_k, r, s'_k\}_k$.
2. Train the model $M(s, a) = (s', r)$ on $\mathcal{D}$ using supervised learning.
3. Optimize the policy π on rollouts τ generated by the model.

- Such a model is called the **dynamics model**, the **transition model** or the **forward model**.
  - **What happens if I do that?**
- The model can be deterministic (use neural networks) or stochastic (use Gaussian Processes).
- Given an initial state s_0 and a policy π, you can unroll the future using the local model.

Learning from imaginary rollouts



- Once you have a good transition model, you can generate **rollouts**, i.e. imaginary trajectories / episodes using the model.
- Given an initial state s_0 and a policy π, you can unroll the future using the model $s', r = M(s, a)$.

$$\tau = s_0 \xrightarrow[\pi]{} a_0 \xrightarrow[M]{} s_1 \xrightarrow[\pi]{} a_1 \xrightarrow[\pi]{} s_2 \xrightarrow[]{} \ldots \xrightarrow[M]{} s_T$$

- You can then feed these trajectories to any optimizer (classical or model-free RL algorithm) that will learn to maximize the returns.

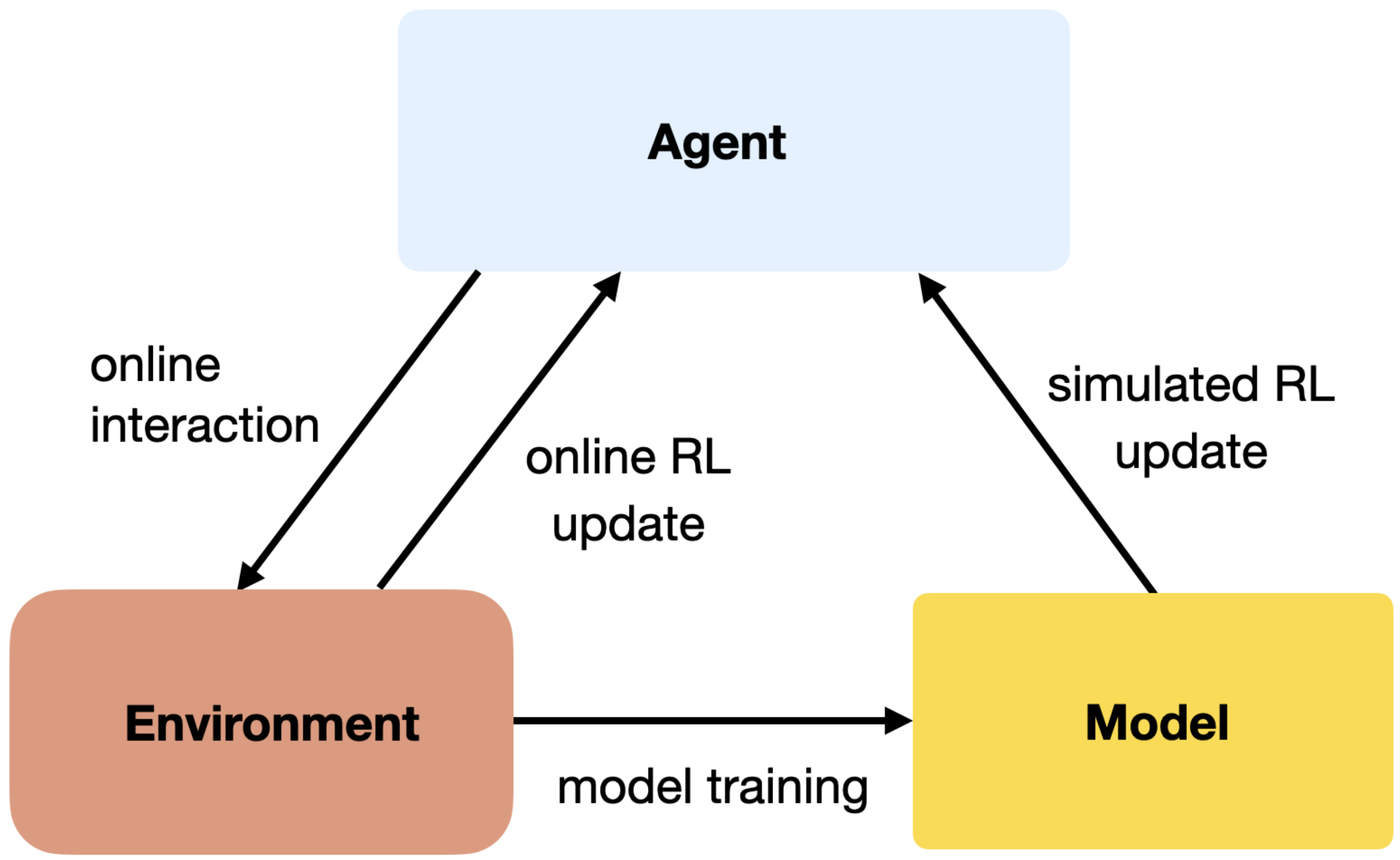
$$\mathcal{J}(\theta) = \mathbb{E}_{\tau}[R(\tau)]$$

- The only sample complexity is the one needed to train the model: the rest is **emulated**.
- Drawback: This can only work when the model is close to perfect, especially for long trajectories or probabilistic MDPs. See MPC in the next chapter.

Kurutach et al. (2018) Model-Ensemble Trust-Region Policy Optimization. arXiv:1802.10592.

3 - Dyna-Q

Dyna-Q



- A simple approach to MB RL is to **augment** MF methods with MB rollouts.
- The MF algorithm (e.g. Q-learning) learns from transitions (s, a, r, s') sampled either with:
    - **real experience**: interaction with the environment.
    - **simulated experience**: simulation by the model.
- If the simulated transitions are good enough, the MF algorithm can converge using much less **real transitions**, thereby reducing its **sample complexity**.

- The **Dyna-Q** algorithm is an extension of Q-learning to integrate a model $M(s, a) = (s', r')$.
- The model can be tabular or approximated with a NN.

Sutton (1990) Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming. Machine Learning Proceedings.

Dyna-Q

- Initialize values $Q(s, a)$ and model $M(s, a)$.
- **for** $t \in [0, T_{\text{total}}]$:
  - Select a_t using Q, take it on the **real environment** and observe s_{t+1} and r_{t+1}.
  - Update the Q-value of the **real** action:

$$\Delta Q(s_t, a_t) = \alpha \, (r_{t+1} + \gamma \, \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$$

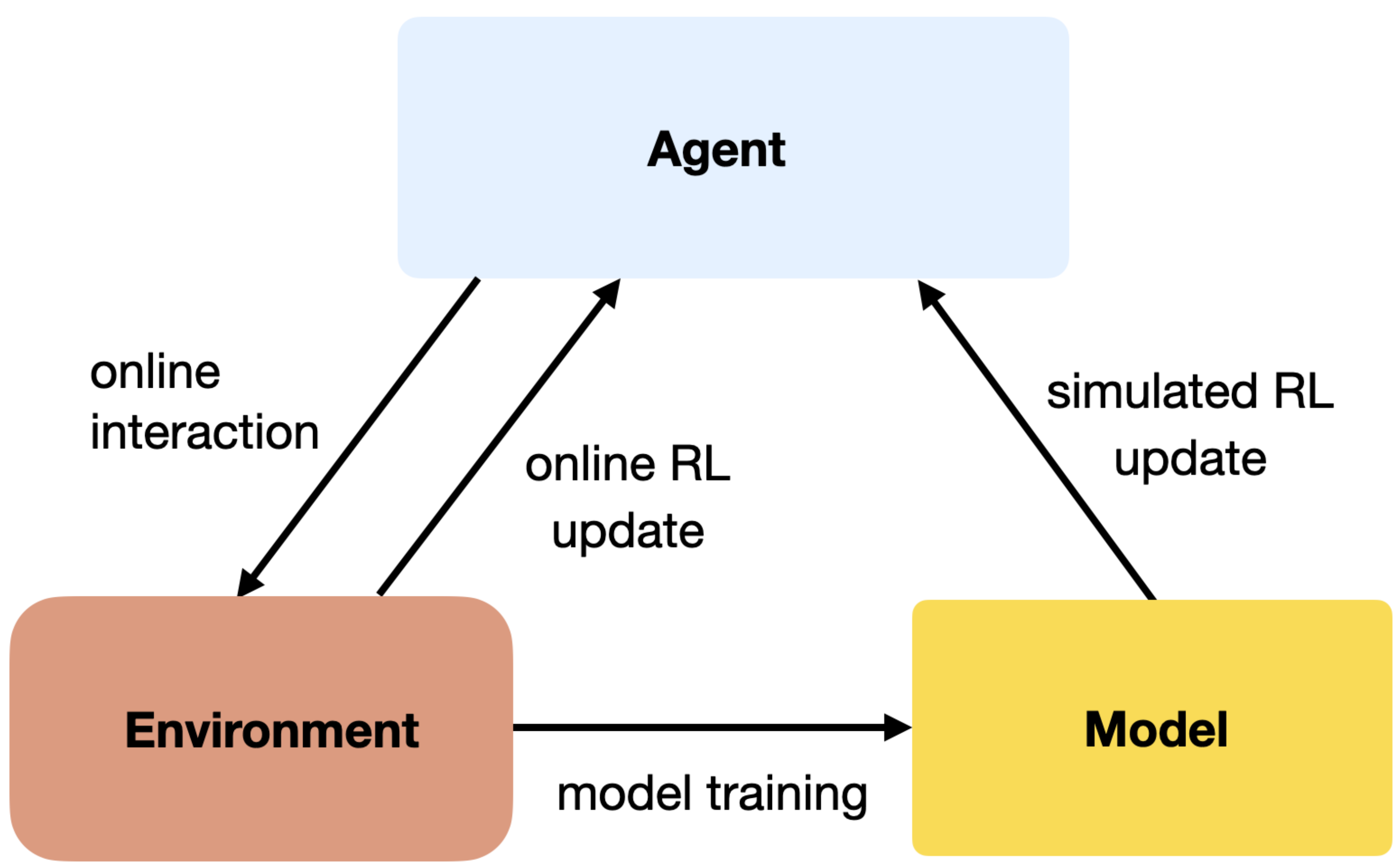
  - Update the model:

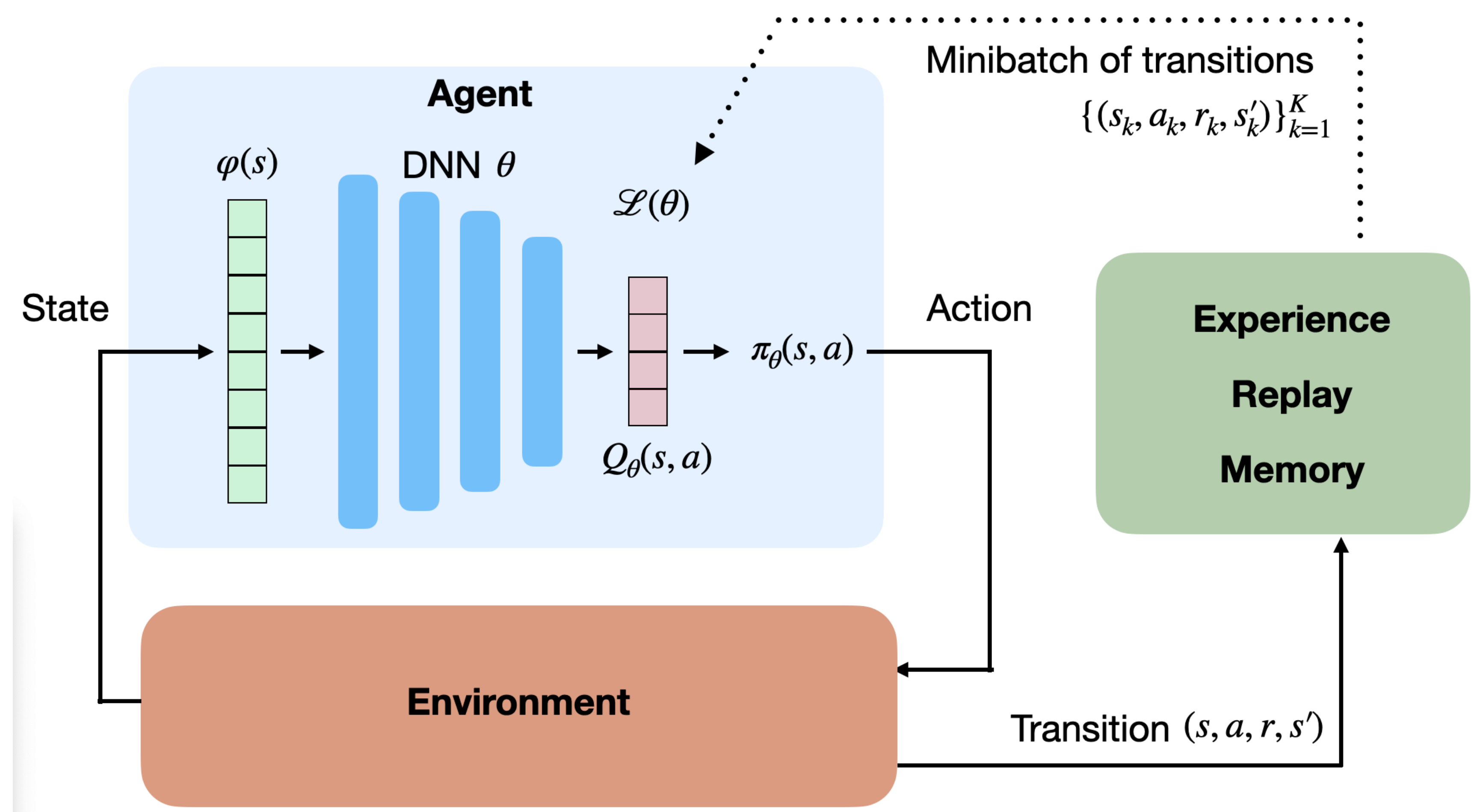
$$M(s_t, a_t) \leftarrow (s_{t+1}, r_{t+1})$$

  - **for** K steps:
    - Sample a state s_k from a list of visited states.
    - Select a_k using Q, predict s_{k+1} and r_{k+1} using the **model** $M(s_k, a_k)$.
    - Update the Q-value of the **imagined** action:

$$\Delta Q(s_k, a_k) = \alpha \, (r_{k+1} + \gamma \, \max_a Q(s_{k+1}, a) - Q(s_k, a_k))$$

Dyna-Q





- It is interesting to notice that Dyna-Q is very similar to DQN and its **experience replay memory**.
- In DQN, the ERM stores **real transitions** generated in the past.
- In Dyna-Q, the model generates **imagined transitions** based on past real transitions.

4 - I2A - Imagination-augmented agents

Imagination-Augmented Agents for Deep Reinforcement Learning

Théophane Weber* Sébastien Racanière* David P. Reichert* Lars Buesing
Arthur Guez Danilo Rezende Adria Puigdomènech Badia Oriol Vinyals
Nicolas Heess Yujia Li Razvan Pascanu Peter Battaglia
Demis Hassabis David Silver Daan Wierstra
DeepMind

https://deepmind.com/blog/article/agents-imagine-and-plan

Weber et al. (2017) Imagination-Augmented Agents for Deep Reinforcement Learning. arXiv:1707.06203.

I2A - Imagination-augmented agents

- I2A is a **model-based augmented model-free method**: it trains a MF algorithm (A3C) with the help of **rollouts** generated by a MB model.

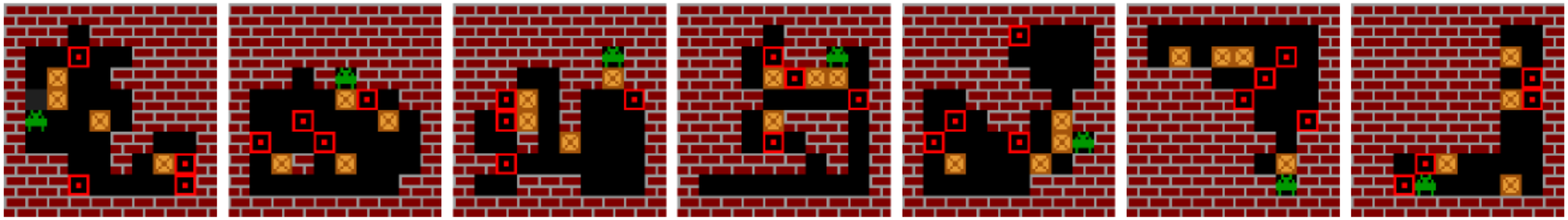


Figure 3: *Random examples of procedurally generated Sokoban levels.* The player (green sprite) needs to push all 4 boxes onto the red target squares to solve a level, while avoiding irreversible mistakes. Our agents receive sprite graphics (shown above) as observations.

- They showcase their algorithm on the puzzle environment **Sokoban**, where you need to move boxes to specified locations.
- Sokoban is a quite hard game, as actions are irreversible (you can get stuck) and the solution requires many actions (sparse rewards).
- MF methods are bad at this game as they learn through trials-and-(many)-errors.

Weber et al. (2017) Imagination-Augmented Agents for Deep Reinforcement Learning. arXiv:1707.06203.

Sokoban

I2A - Imagination-augmented agents

- The **model** learns to predict the next frame and the next reward based on the four last frames and the chosen action.

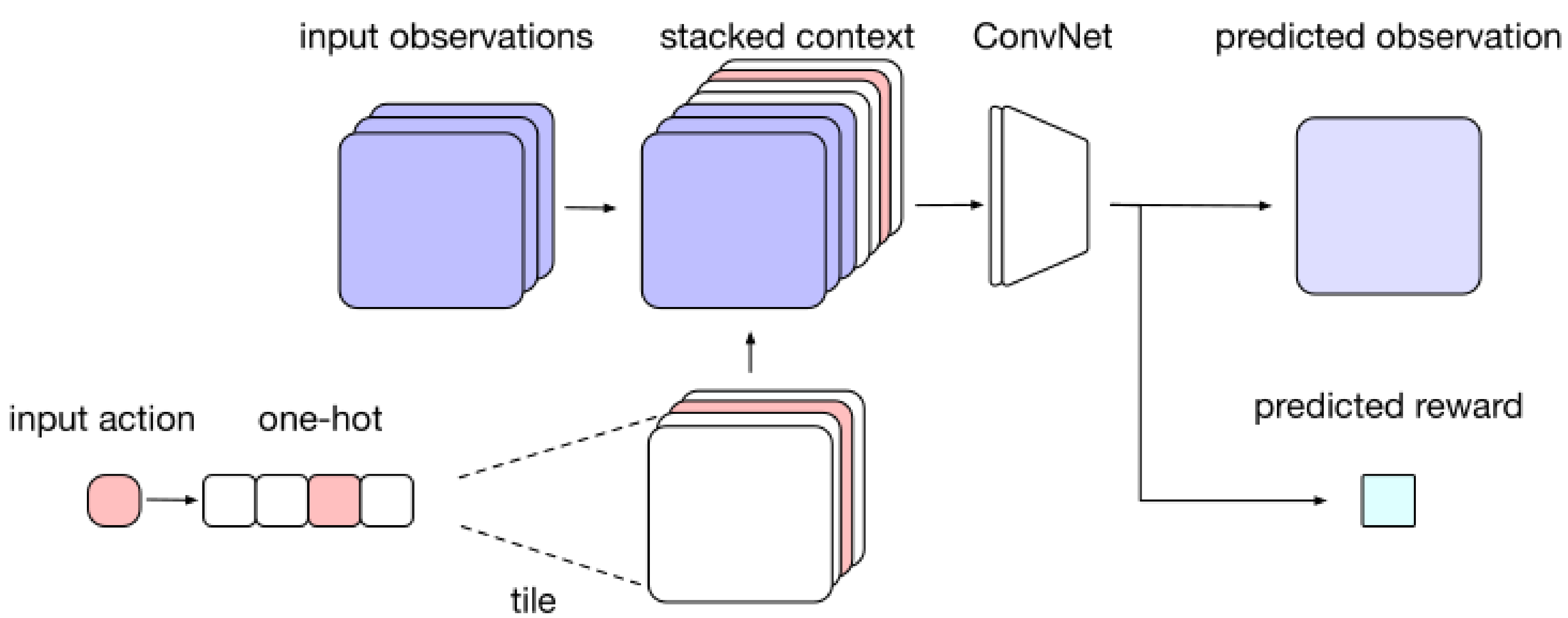
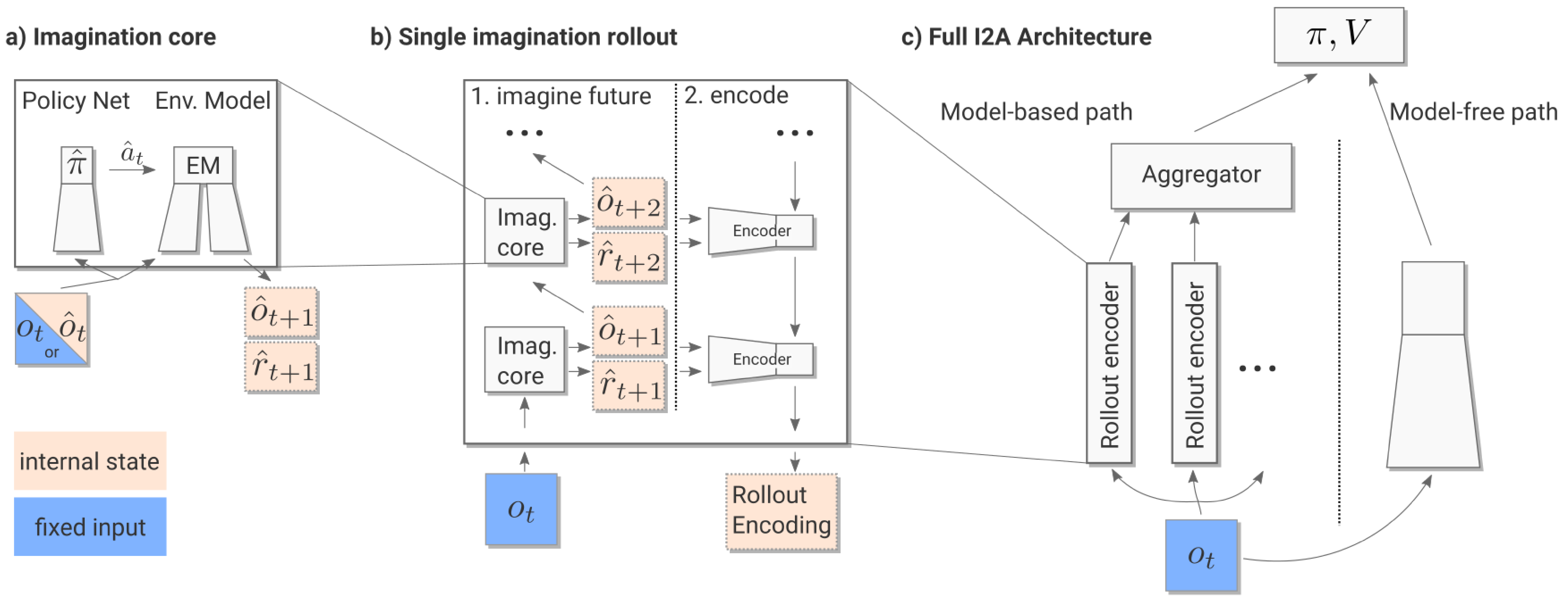


Figure 2: *Environment model.* The input action is broadcast and concatenated to the observation. A convolutional network transforms this into a pixel-wise probability distribution for the output image, and a distribution for the reward.

- It is a **convolutional autoencoder**, taking additionally an action a as input and predicting the next reward.
- It can be pretrained using a random policy, and later fine-tuned during training.

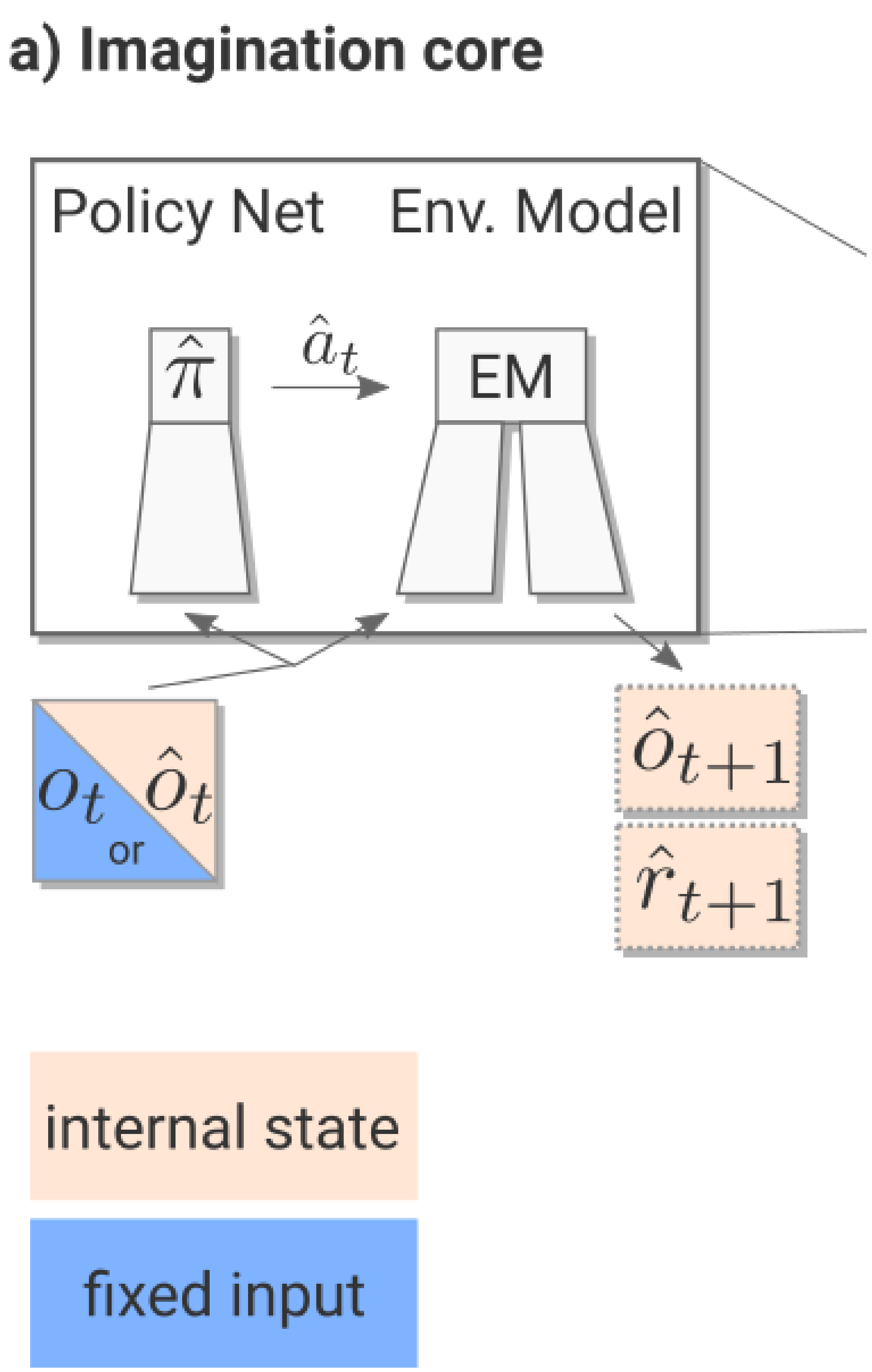
Weber et al. (2017) Imagination-Augmented Agents for Deep Reinforcement Learning. arXiv:1707.06203.

I2A - Imagination-augmented agents



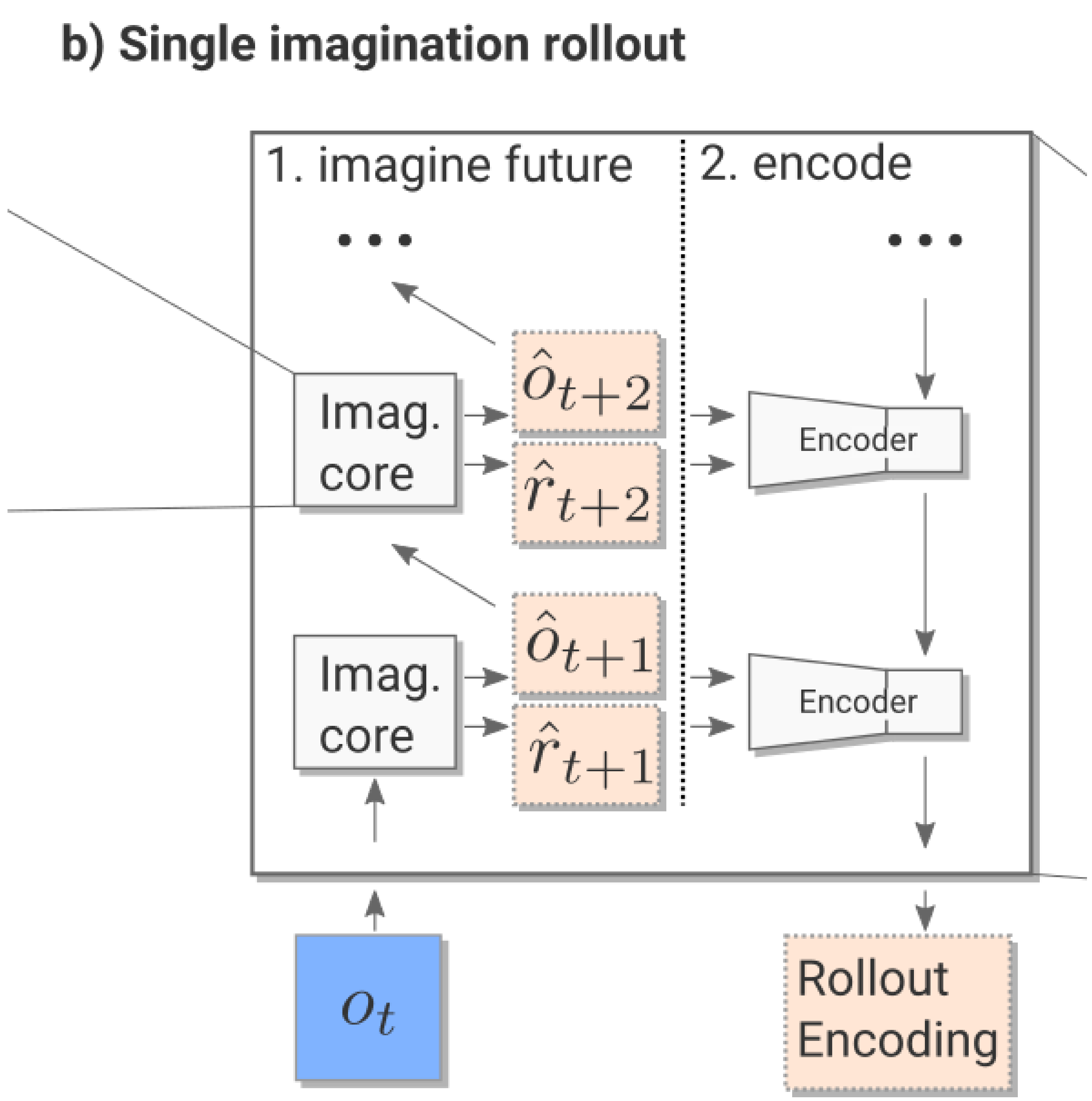
Weber et al. (2017) Imagination-Augmented Agents for Deep Reinforcement Learning. arXiv:1707.06203.

I2A - Imagination-augmented agents



- The **imagination core** is composed of the environment model $M(s, a)$ and a **rollout policy** $\hat{\pi}$.
- As Sokoban is a POMDP (partially observable), the notation uses **observation** o_t instead of states s_t, but it does not really matter here.
- The **rollout policy** $\hat{\pi}$ is a simple and fast policy. It does not have to be the trained policy π.
- It could even be a random policy, or a pretrained policy using for example A3C directly.
- In I2A, it is a **distilled policy** from the trained policy π (see later).
- Take home message: given the current observation o_t and a policy $\hat{\pi}$, we can predict the next observation $\hat{o}_{t+1}$ and the next reward $\hat{r}_{t+1}$.

I2A - Imagination-augmented agents



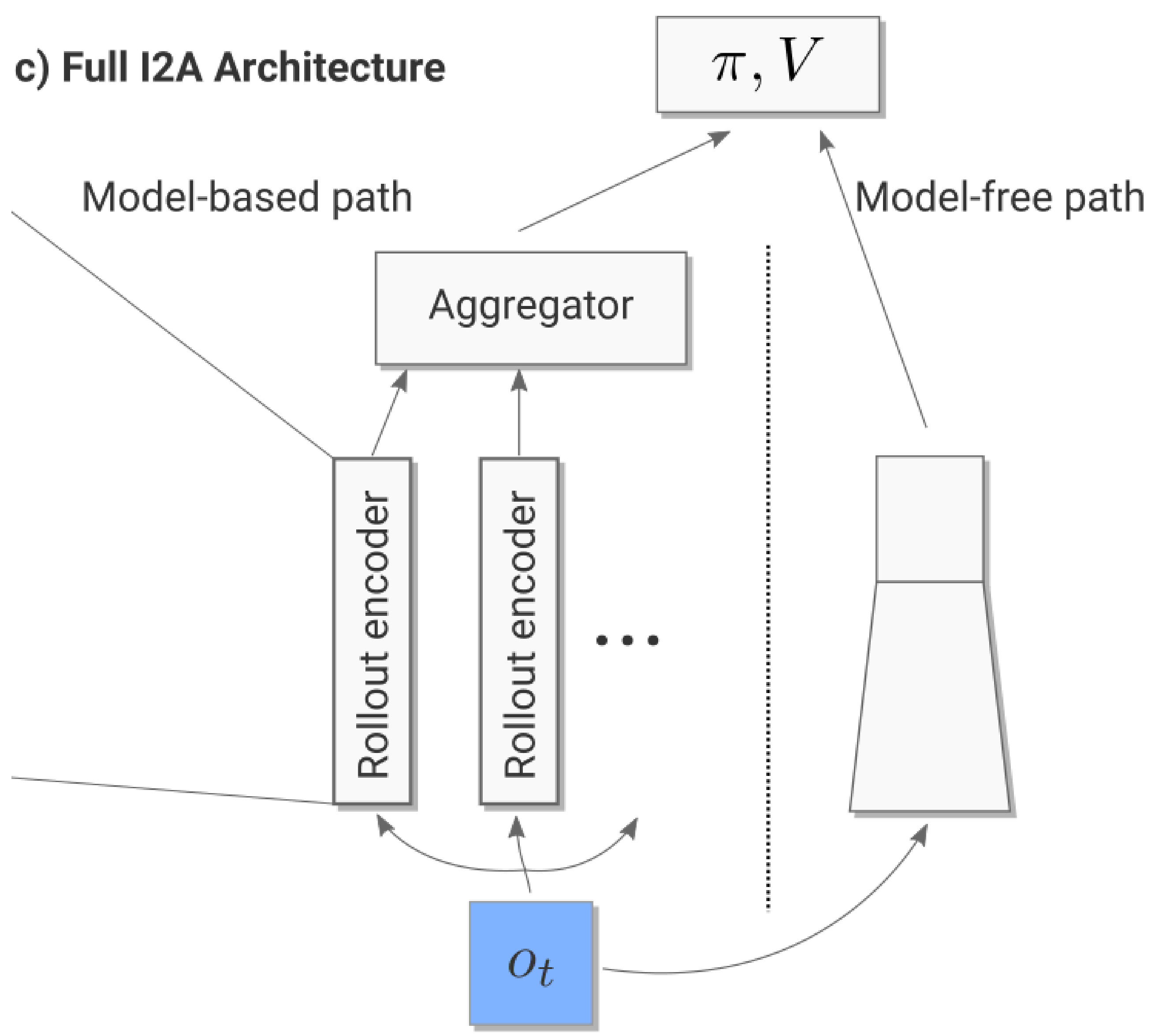
- The **imagination rollout module** uses the imagination core to predict iteratively the next τ frames and rewards using the current frame o_t and the rollout policy:

$$o_t \rightarrow \hat{o}_{t+1} \rightarrow \hat{o}_{t+2} \rightarrow \ldots \rightarrow \hat{o}_{t+\tau}$$

- The τ frames and rewards are passed **backwards** to a convolutional LSTM (from $t+\tau$ to t) which produces an embedding / encoding of the rollout.
- The output of the imagination rollout module is a vector e_i (the final state of the LSTM) representing the whole rollout, including the (virtually) obtained rewards.
- Note that because of the stochasticity of the rollout policy $\hat{\pi}$, different rollouts can lead to different encoding vectors.

Weber et al. (2017) Imagination-Augmented Agents for Deep Reinforcement Learning. arXiv:1707.06203.

I2A - Imagination-augmented agents



- For the current observation o_t, we then generate one **rollout** per possible action (5 in Sokoban):
  - What would happen if I do action 1?
  - What would happen if I do action 2?
  - etc.
- The resulting vectors are concatenated to the output of **model-free** path (a convolutional neural network taking the current observation as input).
- Altogether, we have a huge NN with weights θ (model, encoder, MF path) producing an input s_t to the **A3C** module.

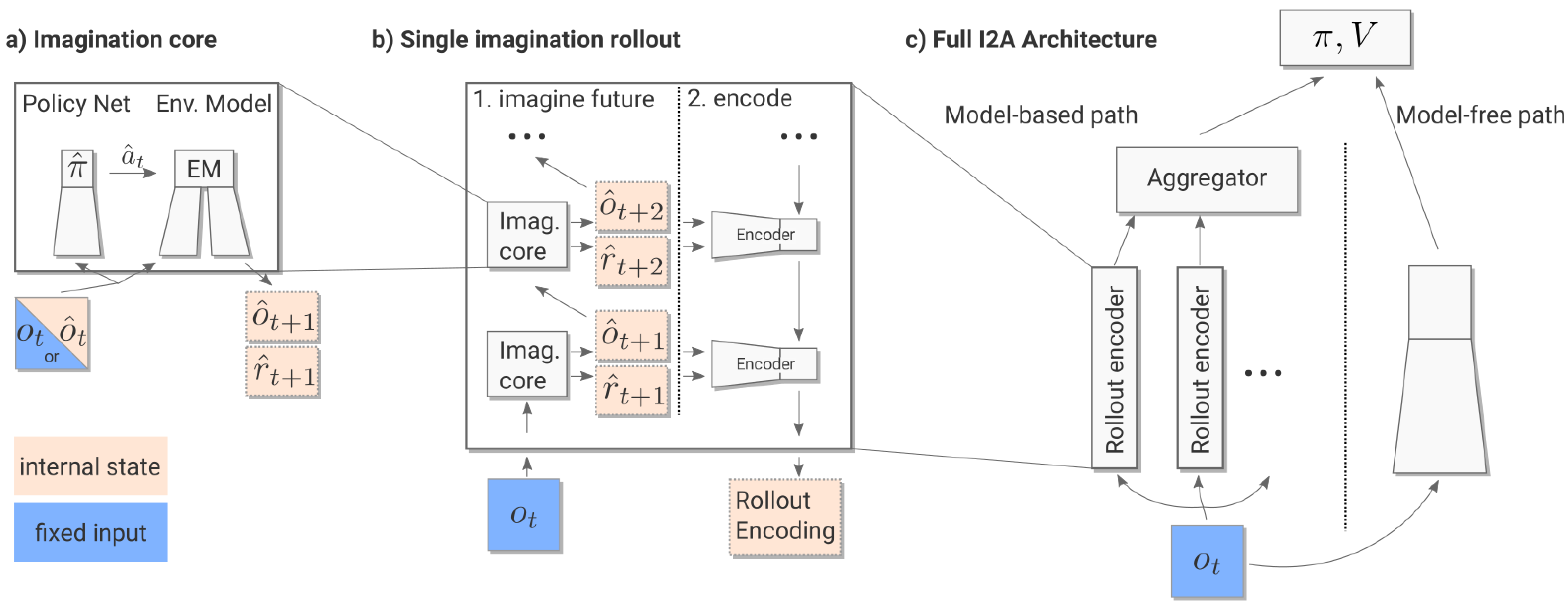
- We can then learn the policy π and value function V based on this input to maximize the returns:

$$\nabla_\theta \mathcal{J}(\theta) = \mathbb{E}_{s_t \sim \rho_\theta, a_t \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(s_t, a_t) \, (\sum_{k=0}^{n-1} \gamma^k \, r_{t+k+1} + \gamma^n \, V_\varphi(s_{t+n}) - V_\varphi(s_t))]$$

$$\mathcal{L}(\varphi) = \mathbb{E}_{s_t \sim \rho_\theta, a_t \sim \pi_\theta}[(\sum_{k=0}^{n-1} \gamma^k \, r_{t+k+1} + \gamma^n \, V_\varphi(s_{t+n}) - V_\varphi(s_t))^2]$$

I2A - Imagination-augmented agents

- The complete architecture may seem complex, but everything is differentiable so we can apply backpropagation and train the network **end-to-end** using multiple workers.
- It is the A3C algorithm (MF), but **augmented** by MB rollouts, i.e. with explicit information about the future.

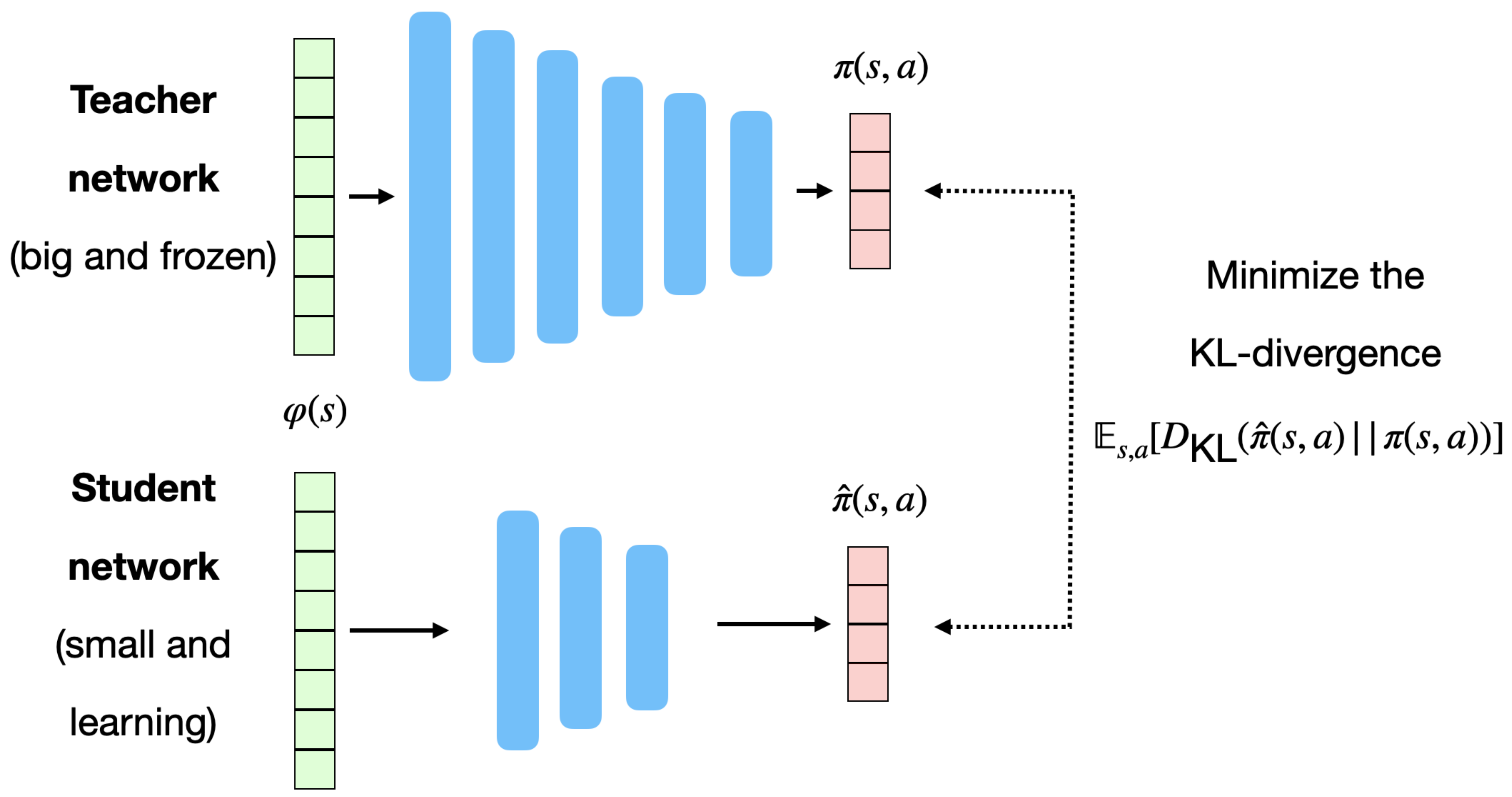


Policy distillation

- The **rollout policy** $\hat{\pi}$ is trained using **policy distillation** of the trained policy π. The small rollout policy network with weights $\hat{\theta}$ tries to copy the outputs $\pi(s, a)$ of the bigger policy network (A3C).
- This is a supervised learning task: just minimize the KL divergence between the two policies:

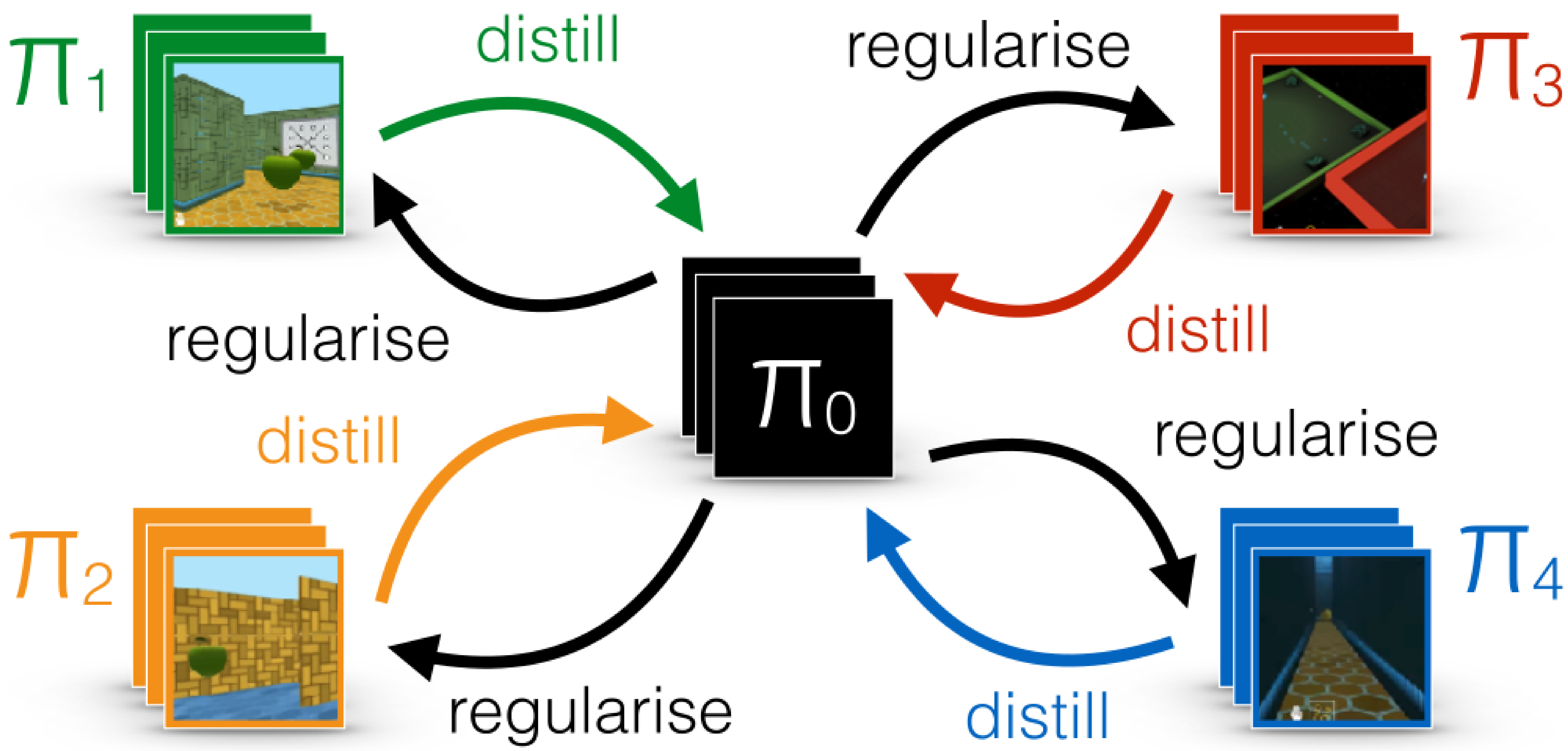
$$\mathcal{L}(\hat{\theta}) = \mathbb{E}_{s,a}[D_{\text{KL}}(\hat{\pi}(s, a)||\pi(s, a))]$$

- As the network is smaller, it won't be as good as π, but its learning objective is easier.



Distral : distill and transfer learning

- FYI: distillation can be used to ensure generalization over different environments.
- Each learning algorithms learns its own task, but tries not to diverge too much from a **shared policy**, which turns out to be good at all tasks.



Teh et al. (2017) Distral: Robust Multitask Reinforcement Learning. arXiv:1707.04175

I2A - Imagination-augmented agents

- Unsurprisingly, I2A performs better than A3C on Sokoban.
- The deeper the rollout, the better.

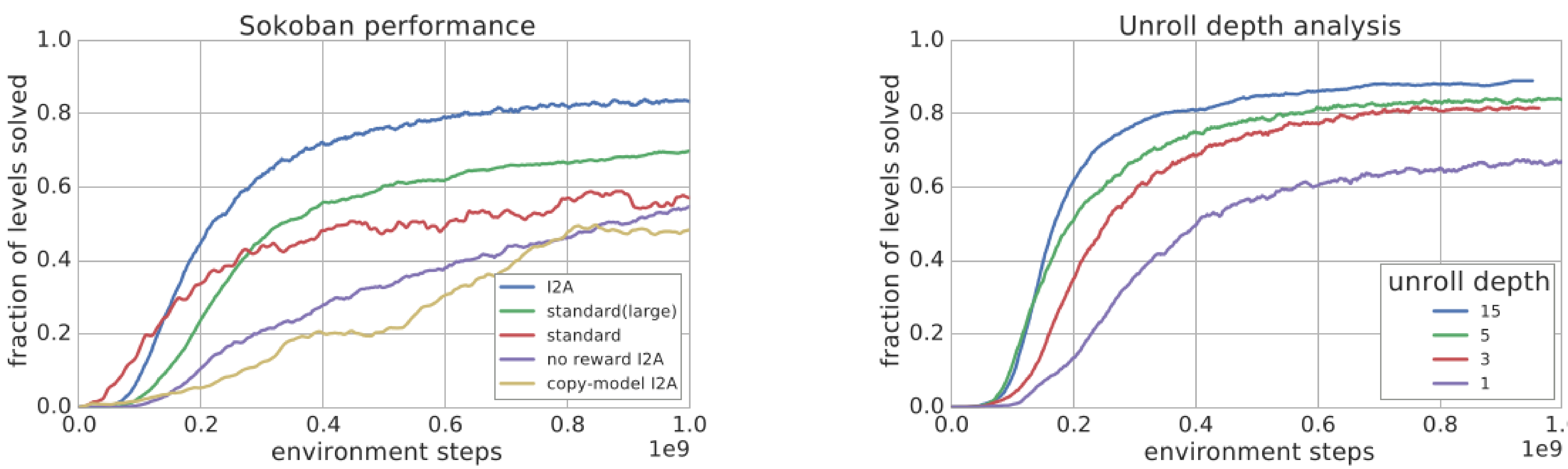


Figure 4: *Sokoban learning curves. Left:* training curves of I2A and baselines. Note that I2A use additional environment observations to pretrain the environment model, see main text for discussion. *Right:* I2A training curves for various values of imagination depth.

I2A - Imagination-augmented agents

- The model does not even have to be perfect: the MF path can compensate for imperfections.

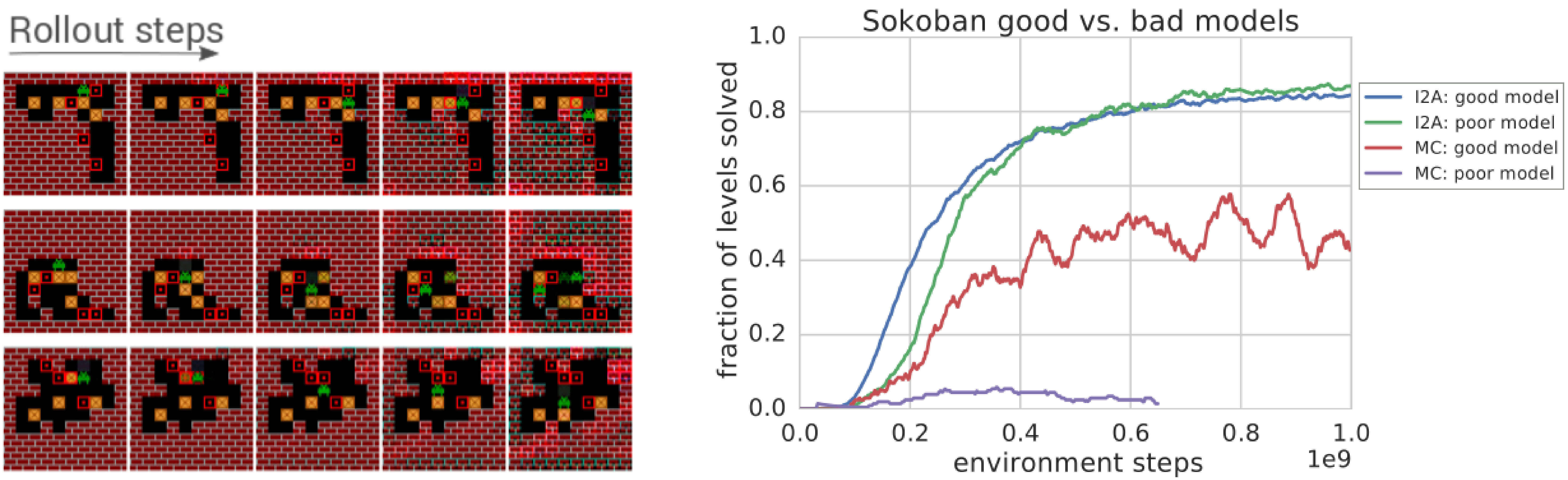
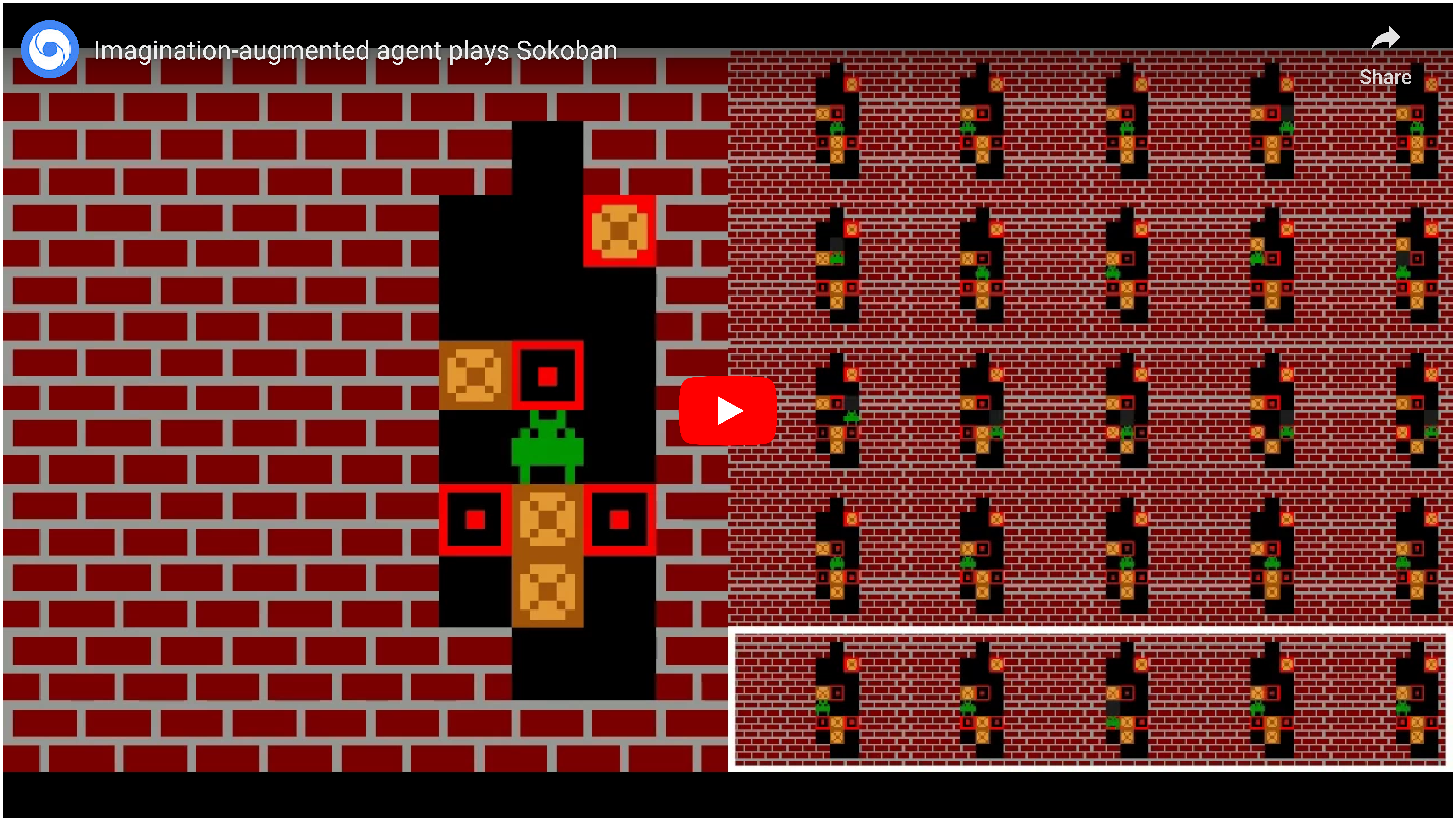


Figure 5: *Experiments with a noisy environment model. Left:* each row shows an example 5-step rollout after conditioning on an environment observation. Errors accumulate and lead to various artefacts, including missing or duplicate sprites. *Right:* comparison of Monte-Carlo (MC) search and I2A when using either the accurate or the noisy model for rollouts.

I2A - Sokoban



References

Dayan, P., and Niv, Y. (2008). Reinforcement learning: The Good, The Bad and The Ugly. *Current Opinion in Neurobiology* 18, 185–196. doi:10.1016/j.conb.2008.08.003.

Kurutach, T., Clavera, I., Duan, Y., Tamar, A., and Abbeel, P. (2018). Model-Ensemble Trust-Region Policy Optimization. http://arxiv.org/abs/1802.10592.

Rusu, A. A., Colmenarejo, S. G., Gulcehre, C., Desjardins, G., Kirkpatrick, J., Pascanu, R., et al. (2016). Policy Distillation. http://arxiv.org/abs/1511.06295.

Sutton, R. S. (1990). Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming. *Machine Learning Proceedings 1990*, 216–224. doi:10.1016/B978-1-55860-141-3.50030-4.

Teh, Y. W., Bapst, V., Czarnecki, W. M., Quan, J., Kirkpatrick, J., Hadsell, R., et al. (2017). Distral: Robust Multitask Reinforcement Learning. http://arxiv.org/abs/1707.04175.

Weber, T., Racanière, S., Reichert, D. P., Buesing, L., Guez, A., Rezende, D. J., et al. (2017). Imagination-Augmented Agents for Deep Reinforcement Learning. http://arxiv.org/abs/1707.06203.