
Neural Bayesian Networks as Continual World Models for Reinforcement Learning

Giovanni Briglia✉, Stefano Mariani, Franco Zambonelli
University of Modena and Reggio Emilia
{name.surname}@unimore.it

Marco Lippi
University of Florence
marco.lippi@unifi.it

Abstract

World models are central to sample-efficient Reinforcement Learning (RL), yet most are monolithic and, even when trained and updated online, they cannot answer the question that *continual learning* raises: “*which part of the world changed?*” Bayesian networks answer exactly that, splitting the dynamics into local mechanisms, one per variable, arranged on a Directed Acyclic Graph (DAG). However, their classical formulation is tabular, which has kept them out of modern world modeling: queries are answered one value at a time and continuous variables are poorly supported. We represent the environment dynamics with *Neural Bayesian Networks* (NBNs): Bayesian Networks whose mechanisms are *learned conditional densities*, which support both continuous and discrete variables and answer queries in batches. To make them continual world models, we build a pipeline that separates changes in a mechanism from shifts in the agent’s behavior, revises only the mechanisms that changed, and plans by inference in the network. In a preliminary experiment on a toy MDP with non-Gaussian dynamics, our pipeline flags only the changed mechanism and keeps its model of the environment dynamics accurate through behavior shifts and mechanism drifts. Released code: <https://github.com/Giovannibriglia/nbn-continual-world-models>.

1 Introduction

Model-based reinforcement learning (RL) rests on world models: learned approximations of the environment dynamics used for prediction, imagination, and planning [Ha and Schmidhuber, 2018, Hafner et al., 2025]. Many are trained once and then frozen [Zidan et al., 2026], yet real environments *change*, and a world model should change with them, by *continual learning*. Single-network models offer no graceful path: fine-tuning may overwrite still-valid knowledge unless past data is replayed [Kirkpatrick et al., 2017, Parisi et al., 2019]; a drop in accuracy says nothing about *what* changed; and no part of a monolithic network corresponds to a part of the world to revise alone.

Bayesian networks (BNs) [Koller and Friedman, 2009] were designed around exactly these missing properties. A BN splits the joint distribution into small local pieces, one conditional mechanism per variable, arranged on a directed acyclic graph (DAG), so the structure of the dynamics is explicit and each mechanism can change without dragging the others with it. This granularity matches how real environments change: usually a few mechanisms shift while the rest keeps working (the *sparse mechanism shift* hypothesis [Schölkopf et al., 2021]), so a factored model can revise only what changed and keep the rest. BNs would be a natural candidate for a world model, as factored Markov decision processes (MDPs) already show for transitions [Boutilier et al., 2000]; however, their classical formulation never met modern world modeling. Their conditional tables grow exponentially with the number of parents, continuous variables cannot be tabulated and need parametric forms, such as linear-Gaussian ones [Koller and Friedman, 2009], and exact inference, and even approximate inference, is NP-hard in general [Cooper, 1990, Dagum and Luby, 1993], while classical engines answer queries one at a time, not in the batches a planner needs.

In this paper, we use *Neural Bayesian Networks* (NBNs) [Briglia et al., 2026] as the backbone of a pipeline for continual world models in RL. An NBN keeps the BN’s graph and factorization, but replaces every conditional distribution with a learned module (linear-Gaussian, neural, or non-parametric) that processes whole batches in parallel. Keeping the graph and learning the mechanisms fits RL particularly well: the decision process fixes the time order of the DAG (Figure 1), and the edges between state variables come from domain knowledge or structure learning, so what is learned, and what drifts, are the local functions attached to the graph, not the structure itself. With this split, the three abilities that a continual world model needs, *detecting change*, *absorbing it*, *keeping what still holds*, become operations on single mechanisms; with planning, they form our pipeline (Section 3), and Section 4 gives a first assessment. The ingredients are not new: BNs have been adapted node by node to non-stationary data [Nielsen and Nielsen, 2008], changing mechanisms have been found with conditional-independence tests [Huang et al., 2020], and model-based agents have adapted their dynamics models online [Nagabandi et al., 2019]; our contribution is a pipeline that combines them, with an NBN as its backbone, into one continual world model.

2 Background

World models learn the transition probabilities $p(s' | s, a)$ and drive planning in imagination [Ha and Schmidhuber, 2018, Hafner et al., 2025], while continual RL studies agents in changing environments [Khetarpal et al., 2022]; its standard tools, such as replay and weight-change penalties as in EWC [Kirkpatrick et al., 2017, Parisi et al., 2019], operate on an unstructured weight vector: they slow forgetting but cannot say *which part* of the world changed. A Bayesian network says which part

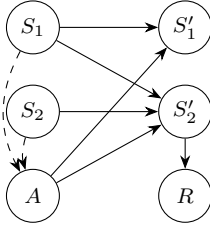


Figure 1: The DAG of the online factored MDP of Section 4, following the structural view of Buesing et al. [2019]. Primes denote time $t+1$. Solid edges: learned mechanisms; dashed: behavior policy; $R=1$ at the goal. The graph is assumed; drift changes the local functions attached to the nodes, never the structure.

changed by construction: it factorizes $p(x) = \prod_i p_i(x_i | x_{\text{Pa}(i)})$ along a DAG whose node i has parents $\text{Pa}(i)$. For an online MDP the time order of that DAG is given: state and action at time t are the parents of the next state and of the reward (Figure 1), and we assume the edges known throughout; since the agent executes its own policy and observes the parents of its actions, its data are interventional, $p(s' | s, do(a)) = p(s' | s, a)$ [Pearl, 2009, Schulte and Poupart, 2025], while logged data could be confounded. What remains uncertain, and drifts, are the mechanisms.

Structured world models are not new: deep structural causal models pair neural networks with an explicit graph for counterfactual estimation [Pawlowski et al., 2020, Xia et al., 2021], causal dynamics learning discovers the graph and exploits its sparsity for robust RL [Wang et al., 2022], and end-to-end discovery couples graph learning with neural conditionals [Lorch et al., 2021]. All assume a *static* setting, fitted once, then queried; none adapts a deployed model per mechanism. Dreamer trains a recurrent latent world model, with actor and critic, “concurrently from replayed experience while the agent interacts with the environment” [Hafner et al., 2025]. V-JEPA 2 pretrains a video encoder on over a million hours of video and plans towards goal images with a predictor trained on under 62 hours of robot video [Assran et al., 2025]. Both learn a state from pixels at scale, which we assume away (Section 3, (i)), but neither addresses maintenance: the dynamics live in one network, so a change in the world shows up as a global loss of accuracy. These encoders and NBNs are complementary: an NBN could maintain the dynamics over the factored latents of such encoders.

3 Neural Bayesian Networks as Continual World Models

In a BN every conditional is a self-contained mechanism, fitted from its own variable and parents alone; classically it is a table or a closed-form density, evaluated one configuration at a time. An NBN replaces each local conditional with a parametrized mapping $p_{\phi_i}(x_i | x_{\text{Pa}(i)}) = \mathcal{M}_i(x_{\text{Pa}(i)}; \phi_i)$, where $\mathcal{M}_i$ is a learned module that can *sample* x_i and *score* $\log p_{\phi_i}(x_i | x_{\text{Pa}(i)})$ for a batch of B

parent configurations at once. The graph and the factorization $p_\phi(X) = \prod_i p_{\phi_i}(X_i | X_{\text{Pa}(i)})$ of the classical BN are preserved exactly; sampling-based inference needs nothing more than these two operations, so every mechanism works on batches and families can be mixed within one model, from linear-Gaussian models to neural densities such as mixture density networks [Bishop, 1994]. A factored MDP becomes a two-slice NBN with the DAG of Figure 1.

Assumptions. The approach requires: (i) the state is *factored* into meaningful variables, not raw pixels; (ii) the DAG is available: in RL the decision process fixes its time order (Figure 1), and the edges between state variables come from expert knowledge or structure learning [Lorch et al., 2021, Wang et al., 2022]; (iii) the variables in the graph are observed in the experience stream; (iv) change affects the mechanisms, i.e. the local functions, while the graph stays fixed (graph drift: Section 5).

Under these assumptions, four properties make an NBN world model continual; on each new batch W of experience they run as a pipeline: test, update, plan. **(1) Change is detected and localized on the graph.** Because the log-likelihood splits into one term per node, $\log p_\phi(x) = \sum_i \log p_{\phi_i}(x_i | x_{\text{Pa}(i)})$, the surprise of an incoming batch is attributed to individual mechanisms at no extra cost. Surprise, however, is not change. If q is the distribution of the parents in the batch and p_i^* the true mechanism, so that the parents $U_i = X_{\text{Pa}(i)} \sim q$ and $X_i \sim p_i^*(\cdot | U_i)$, then

$$\mathbb{E}[\log p_{\phi_i}(X_i | U_i)] = -\mathbb{E}_q[H(p_i^*(\cdot | U_i))] - \mathbb{E}_q[\text{KL}(p_i^*(\cdot | U_i) \| p_{\phi_i}(\cdot | U_i))]. \quad (1)$$

Only the second term measures a wrong mechanism; the first moves whenever q moves, for example when the agent visits noisier states, even if the mechanism is untouched. We therefore test the conditional itself, $H_0: X_i \perp E | \text{Pa}(i)$, where E indicates whether a transition comes from the stored data or from the new batch, the invariance condition of Peters et al. [2016]. With continuous parents no such test is valid against all alternatives [Shah and Peters, 2020], so we use a simple one and check its calibration: a permutation test on pairs of new and stored transitions with close parents (Appendix A). It does not use the model, so it is the same for every mechanism family, and runs at level α with a Bonferroni correction over mechanisms [Dunn, 1961]. A flag on node i reads: *the mechanism of X_i changed*. **(2) New evidence is absorbed where the world changed.** Each mechanism keeps the data $\mathcal{D}_i$ seen since its last restart: if the test flags it, it restarts from W ; otherwise $\mathcal{D}_i \leftarrow \mathcal{D}_i \cup W$ and it is refit. **(3) Stability comes from structure, plasticity from the test.** A sliding window or a forgetting factor would wear down every mechanism, including those that did not change; here old data are dropped only where the test fires, and a mechanism that is not flagged keeps all its data: knowledge is kept because the factorization isolates it. This is the *informed* strategy of concept-drift adaptation [Gama et al., 2014], applied to each mechanism. **(4) Planning is a batched query.** “What if I take action a ?” is the interventional query $p(s' | s, do(a))$: cut the behavior edges into A and set $A=a$ [Pearl, 2009]. With $R=1$ at the goal and a uniform prior on A , $p(a | s, R=1) \propto P(R=1 | s, do(a))$: the most probable action given success is the one most likely to succeed [Levine, 2018]. Likelihood weighting [Koller and Friedman, 2009] samples actions from the uniform prior and next states from the mechanisms, and weights them by $P(R=1 | s')$; one batched call serves many states.

4 Experiment and Discussion

Setting. The MDP has the DAG of Figure 1, two continuous state variables and an action $A \in [-1, 1]$. S'_1 is linear in (S_1, A) with skewed noise; S'_2 is nonlinear in A , with *bimodal* noise whose spread grows with S_1 (Appendix A). The goal is $|S'_2| < 0.2$. Experience arrives in 8 phases of 2,000 transitions. At phase 2 the *behavior* changes: the agent visits other, noisier states and its policy depends on S_1 ; no mechanism changes. At phase 5 the *mechanism* of S'_2 changes: the effect of the action is reversed. Transition mechanisms are mixture density networks (MDNs) [Bishop, 1994] with 5 components (Appendix A). We compare the pipeline with four other ways to maintain the same NBN: *frozen* after phase 0; *pooled*, refit on all data; a *sliding window*, refit on the last phase only; and *raw likelihood + restart*, the pipeline with a one-sided z -test for a drop of log-likelihood in place of our test. Model error is the Wasserstein-1 distance W_1 to the current dynamics (Appendix A). Tests run at $\alpha=0.01$ (0.5% per mechanism, Bonferroni); 5 seeds.

Which mechanism changed? (Figure 2a). Over 7 batches, 2 mechanisms and 5 seeds, the test ran 70 times; only 5 of these faced a real change (S'_2 at phase 5), and the test flagged exactly those 5, with no flag at the behavior shift. The raw likelihood also catches the drift, but it flags S'_2 at the behavior shift in all 5 runs, and again at phase 3 in 3 runs: the agent now visits states where S'_2 is noisier, so

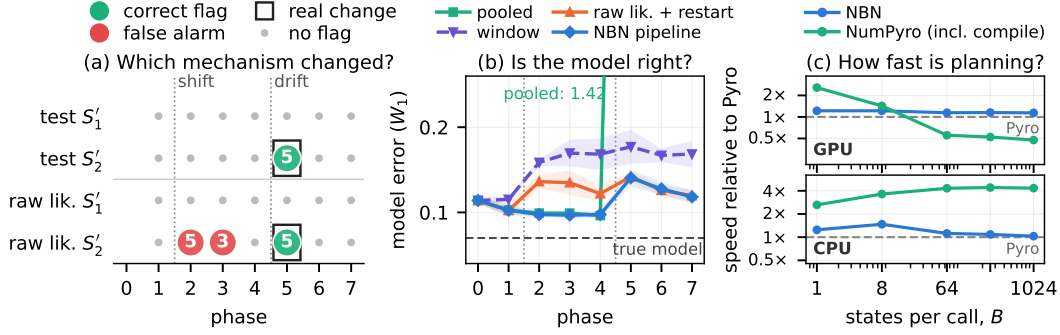


Figure 2: MDN mechanisms; behavior shift at phase 2, drift of S'_2 at phase 5 (dotted lines). **(a)** For each detector (*test*: ours; *raw lik.*: raw likelihood) and mechanism, the number of runs, out of 5, that flagged a change at each phase; the only real change is boxed. **(b)** W_1 of the model’s next-state distribution to the *current* true one; dashed: the true model; mean $\pm$ std over 5 seeds; frozen not shown. **(c)** Planner speed relative to vectorized Pyro on the same device (dashed), for 10,240 queries in calls of B states; top: GPU, bottom: CPU; NumPyro with compilation.

the entropy term of Eq. (1) drops, and even the true mechanism loses 0.35 nats. It never flags S'_1 , whose noise does not depend on its parents, as Eq. (1) predicts. More results in Appendix A, Table 1.

Is the model right? (Figure 2b). At the behavior shift the pipeline keeps all history and is as accurate as pooling ($W_1 = 0.10$), while the raw likelihood, which dropped the history of S'_2 on a false alarm, rises to 0.14, and the window to 0.16. After the drift, pooling mixes the two regimes ($W_1 = 1.42$), while the pipeline, which restarts only S'_2 , is at 0.14, below the window (0.18). More results in Appendix A, Table 2.

How fast is planning? (Figure 2c). We compare the NBN with the state-of-the-art probabilistic programming library Pyro [Bingham et al., 2019] as a direct competitor, and its JAX-optimized NumPyro version [Phan et al., 2019] as a ceiling. All three run the same planner on the same trained MDNs, on both CPU and GPU: each answers 10240 queries in batches of B states (complete results in Appendix A, Tables 3–4). Overall, all three methods produce plans of equal quality, and the NBN is competitive in speed. It is faster than Pyro on both devices. Against NumPyro, it is slower on the CPU and on the GPU for small batches. On the GPU for $B \geq 64$ it is faster on this workload, because NumPyro’s compilation, paid once per batch size, outweighs its faster JIT-compiled steady state (Appendix A). Unlike Pyro and NumPyro, where the factorization lives inside a model program, the NBN explicitly represents each mechanism as a node of Figure 1, with its own data, test and update, and planning is a query over those nodes.

5 Conclusions and Future Developments

As our pipeline’s backbone, NBNs make the classical Bayesian network flexible and reactive enough to serve as a continual world model for RL; our experiment is a first feasibility check, one toy MDP, abrupt changes and one-step goals. Next are: 1) *gradual drift and longer horizons*, where planning as inference needs a correction for stochastic dynamics [Levine, 2018]; 2) *graph drift*, with the per-mechanism test triggering local structure search; 3) *from pixels to variables*, via the encoders of Section 2; 4) *the causal reading* [Pearl, 2019]: with structural equations, abduction [Pearl, 2009] and a noise coupling [Oberst and Sontag, 2019], NBNs could deconfound offline experience [Zeng et al., 2025] and answer “*what if I had acted differently?*”.

References

- Mido Assran, Adrien Bardes, David Fan, Quentin Garrido, Russell Howes, Mojtaba Komeili, et al. V-JEPA 2: Self-supervised video models enable understanding, prediction and planning. *arXiv preprint arXiv:2506.09985*, 2025. URL <https://arxiv.org/abs/2506.09985>.
- Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D. Goodman. Pyro: Deep universal

- probabilistic programming. *Journal of Machine Learning Research*, 20(28):1–6, 2019. URL <https://jmlr.org/papers/v20/18-403.html>.
- Christopher M. Bishop. Mixture density networks. Technical Report NCRG/94/004, Aston University, 1994. URL <https://publications.aston.ac.uk/id/eprint/373/>.
- Craig Boutilier, Richard Dearden, and Moisés Goldszmidt. Stochastic dynamic programming with factored representations. *Artificial Intelligence*, 121(1–2):49–107, 2000. doi: 10.1016/S0004-3702(00)00033-3.
- Giovanni Briglia, Marco Lippi, Stefano Mariani, and Franco Zambonelli. Neural Bayesian networks for modular and scalable decision-making under uncertainty. *arXiv preprint arXiv:2610.16902*, 2026. URL <https://arxiv.org/abs/2610.16902>.
- Lars Buesing, Theophane Weber, Yori Zwols, Nicolas Heess, Sébastien Racanière, Arthur Guez, and Jean-Baptiste Lespiau. Woulda, coulda, shoulda: Counterfactually-guided policy search. In *International Conference on Learning Representations*, 2019. URL <https://arxiv.org/abs/1811.06272>.
- Gregory F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42(2-3):393–405, 1990. doi: 10.1016/0004-3702(90)90060-D.
- Paul Dagum and Michael Luby. Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial Intelligence*, 60(1):141–153, 1993. doi: 10.1016/0004-3702(93)90036-B.
- Olive Jean Dunn. Multiple comparisons among means. *Journal of the American Statistical Association*, 56(293):52–64, 1961. doi: 10.1080/01621459.1961.10482090.
- João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM Computing Surveys*, 46(4):44:1–44:37, 2014. doi: 10.1145/2523813.
- David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems*, volume 31, pages 2455–2467, 2018. URL <https://proceedings.neurips.cc/paper/2018/hash/2de5d16682c3c35007e4e92982f1a2ba-Abstract.html>.
- Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse control tasks through world models. *Nature*, 640(8059):647–653, 2025. doi: 10.1038/s41586-025-08744-2.
- Biwei Huang, Kun Zhang, Jiji Zhang, Joseph Ramsey, Ruben Sanchez-Romero, Clark Glymour, and Bernhard Schölkopf. Causal discovery from heterogeneous/nonstationary data. *Journal of Machine Learning Research*, 21(89):1–53, 2020. URL <https://jmlr.org/papers/v21/19-232.html>.
- Khimya Khetarpal, Matthew Riemer, Irina Rish, and Doina Precup. Towards continual reinforcement learning: A review and perspectives. *Journal of Artificial Intelligence Research*, 75:1401–1476, 2022. doi: 10.1613/jair.1.13673.
- James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017. doi: 10.1073/pnas.1611835114.
- Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, Cambridge, MA, 2009.
- Sergey Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review. *arXiv preprint arXiv:1805.00909*, 2018. URL <https://arxiv.org/abs/1805.00909>.
- Lars Lorch, Jonas Rothfuss, Bernhard Schölkopf, and Andreas Krause. DiBS: Differentiable Bayesian structure learning. In *Advances in Neural Information Processing Systems*, volume 34, pages 24111–24123, 2021. URL <https://proceedings.neurips.cc/paper/2021/hash/ca6ab34959489659f8c3776aaf1f8efd-Abstract.html>.

- Anusha Nagabandi, Chelsea Finn, and Sergey Levine. Deep online learning via meta-learning: Continual adaptation for model-based RL. In *International Conference on Learning Representations*, 2019. URL <https://arxiv.org/abs/1812.07671>.
- Søren Holbech Nielsen and Thomas D. Nielsen. Adapting Bayes network structures to non-stationary domains. *International Journal of Approximate Reasoning*, 49(2):379–397, 2008. doi: 10.1016/j.ijar.2008.02.007.
- Michael Oberst and David Sontag. Counterfactual off-policy evaluation with Gumbel-max structural causal models. In *International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 4881–4890, 2019. URL <https://proceedings.mlr.press/v97/oberst19a.html>.
- German I. Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, 2019. doi: 10.1016/j.neunet.2019.01.012.
- Nick Pawlowski, Daniel Coelho de Castro, and Ben Glocker. Deep structural causal models for tractable counterfactual inference. In *Advances in Neural Information Processing Systems*, volume 33, pages 857–869, 2020. URL <https://proceedings.neurips.cc/paper/2020/hash/0987b8b338d6c90bbedd8631bc499221-Abstract.html>.
- Judea Pearl. *Causality: Models, Reasoning, and Inference*. Cambridge University Press, 2nd edition, 2009. doi: 10.1017/CBO9780511803161.
- Judea Pearl. The seven tools of causal inference, with reflections on machine learning. *Communications of the ACM*, 62(3):54–60, 2019. doi: 10.1145/3241036.
- Jonas Peters, Peter Bühlmann, and Nicolai Meinshausen. Causal inference by using invariant prediction: Identification and confidence intervals. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 78(5):947–1012, 2016. doi: 10.1111/rssb.12167.
- Du Phan, Neeraj Pradhan, and Martin Jankowiak. Composable effects for flexible and accelerated probabilistic programming in NumPyro. *arXiv preprint arXiv:1912.11554*, 2019. URL <https://arxiv.org/abs/1912.11554>.
- Bernhard Schölkopf, Francesco Locatello, Stefan Bauer, Nan Rosemary Ke, Nal Kalchbrenner, Anirudh Goyal, and Yoshua Bengio. Toward causal representation learning. *Proceedings of the IEEE*, 109(5):612–634, 2021. doi: 10.1109/JPROC.2021.3058954.
- Oliver Schulte and Pascal Poupart. When should reinforcement learning use causal reasoning? *Transactions on Machine Learning Research*, 2025. URL <https://openreview.net/forum?id=D1PPuk8ZBI>.
- Rajen D. Shah and Jonas Peters. The hardness of conditional independence testing and the generalised covariance measure. *The Annals of Statistics*, 48(3):1514–1538, 2020. doi: 10.1214/19-AOS1857.
- Zizhao Wang, Xuesu Xiao, Zifan Xu, Yuke Zhu, and Peter Stone. Causal dynamics learning for task-independent state abstraction. In *International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 23151–23180. PMLR, 2022. URL <https://proceedings.mlr.press/v162/wang22ae.html>.
- Kevin Xia, Kai-Zhan Lee, Yoshua Bengio, and Elias Bareinboim. The causal-neural connection: Expressiveness, learnability, and inference. In *Advances in Neural Information Processing Systems*, volume 34, pages 10823–10836, 2021. URL <https://proceedings.neurips.cc/paper/2021/hash/5989add1703e4b0480f75e2390739f34-Abstract.html>.
- Yan Zeng, Ruichu Cai, Fuchun Sun, Libo Huang, and Zhifeng Hao. A survey on causal reinforcement learning. *IEEE Transactions on Neural Networks and Learning Systems*, 36(4):5942–5962, 2025. doi: 10.1109/TNNLS.2024.3403001.
- Arif Hassan Zidan, Yi Pan, Hanqi Jiang, Ruiyu Yan, Wei Ruan, et al. World models: A comprehensive survey of architectures, methodologies, reasoning paradigms, and applications. *arXiv preprint arXiv:2606.00133*, 2026. URL <https://arxiv.org/abs/2606.00133>.

A Experimental details

Dynamics. $S'_1 = 0.8S_1 + 0.6A + 0.3\varepsilon_1$ with $\varepsilon_1 = (G - 2)/\sqrt{2}$, $G \sim \text{Gamma}(2, 1)$ (skewed, unit variance). $S'_2 = 0.6S_2 - 0.4S_1 + c \tanh(2A) + \sigma(S_1)\varepsilon_2$ with $\varepsilon_2 = 0.9B + 0.436Z$, $B = \pm 1$ with equal probability, $Z \sim \mathcal{N}(0, 1)$ (bimodal, unit variance), and $\sigma(S_1) = 0.15 + 0.45/(1 + e^{-2S_1})$; $c = 1.2$ before phase 5 and $c = -1.2$ from phase 5 on. *Behavior:* in phases 0–1, $S_1, S_2 \sim \mathcal{N}(0, 1)$ and $A \sim \mathcal{U}[-1, 1]$; from phase 2, $S_1 \sim \mathcal{N}(0.8, 0.7^2)$, $S_2 \sim \mathcal{N}(-0.4, 1)$ and $A = \text{clip}(-0.4S_1 + 0.5\eta, -1, 1)$, $\eta \sim \mathcal{N}(0, 1)$. States are drawn i.i.d. in each phase.

Mechanisms. The transition mechanisms are learned; the roots are fixed: states are always observed, and the action prior is the planner’s uniform prior. MDN: 5 Gaussian components, two hidden layers of 64 units, 200 epochs from scratch (restart, window) or 60 epochs from the previous weights (append, pooled). $P(R=1 \mid s'_2) = \mathbb{P}\{|s'_2| < 0.2\}$, floored at 10^{-6} . Linear-Gaussian reference: least squares with Gaussian noise.

Test. Parents are standardized with the stored data $\mathcal{D}_i$. In random order, each transition of W is matched to the nearest unused transition of $\mathcal{D}_i$ in parent space, if one lies within distance 0.25. Residuals $r = x - g(\text{pa})$ use a cubic polynomial g fitted on $\mathcal{D}_i$, which corrects the small parent mismatch within each pair. With $d = r^W - r^D$ and $d_2 = (r^W)^2 - (r^D)^2$ per pair, the statistic is the sum of the R^2 of the regressions of d and of d_2 on cubic features of the pair’s parents. Under H_0 the two children of a pair are exchangeable, so swapping a pair flips the signs of d and d_2 ; the p -value is the rank of the statistic among 999 random sign flips. *Bonferroni correction* [Dunn, 1961]: with m mechanisms tested on the same batch, each test runs at level α/m ; by the union bound, the probability of at least one false flag in that batch is at most α , whatever the dependence between the tests. The guarantee is per batch: over T batches up to about $T\alpha$ false flags are expected, and controlling them over a whole run is a sequential change-detection problem that we do not address. *Raw likelihood:* a one-sided two-sample z -test of a drop in the mean log-likelihood of the new batch with respect to the previous batch, both scored by the model before absorbing them (hence from phase 2 on). *Checks* (Table 1): on pairs of simulated batches, the test keeps its level with and without a behavior shift and always flags the drift of S'_2 ; with the true mechanism, the mean log-likelihood of S'_2 goes from -0.15 under the original behavior to -0.50 under the shifted one, the 0.35 nats of Section 4.

Table 1: Flag rate of our test at nominal 0.5% per mechanism, on pairs of simulated batches of 2,000 transitions (200 pairs per case; 50 for the drift).

new batch	S'_1	S'_2
no change	1.0%	0.5%
behavior shift	0.5%	1.5%
drift of S'_2	0%	100%

Planning and metrics. 200 fixed start states with $S_1, S_2 \sim \mathcal{N}(0, 1)$. Likelihood weighting draws 2,048 particles per state in one batched call; the posterior mode is read from a weighted Gaussian kernel (bandwidth 0.05) on a grid of 201 actions. The score is the mean true success probability of the chosen actions, in closed form, divided by that of the best grid actions. Model error is the Wasserstein-1 distance between the model’s and the true next-state distributions, averaged over 400 parent points from the original behavior, with 256 samples per point; since both kernels make S'_1 and S'_2 independent given the parents, the W_1 of the joint kernel under an L_1 cost is the sum of the two per-mechanism distances. The true model scores 0.07, from sampling alone. Per phase on a laptop CPU (one thread): test 0.04s, update 0.7s, planning for 200 states 0.6s. All NBNs use the nbn library (version 0.20.2); the code and the instructions to reproduce every result are at <https://github.com/Giovannibriglia/nbn-continual-world-models>.

Model and plan (Table 2). At the behavior shift the pipeline plans at 95% of optimal, as pooling, against 90% for the raw likelihood and 89% for the window. Right after the drift the three strategies that adapt plan alike (92–93%, against 96% with the true dynamics), while pooling falls to 25% and the frozen model to 21%. Over phases 1–7 the pipeline loses the least against the true-dynamics planner (0.17, against 0.25 for the raw likelihood and 0.41 for the window). A linear-Gaussian NBN plans at 72–74% throughout: it aims between the two modes of S'_2 .

Table 2: Model error (W_1) and plan score (success relative to the optimal action), means over 5 seeds: at the behavior shift (phase 2), right after the drift (phase 5), and summed over phases 1–7 as excess over the true model (W_1) or loss against the planner with the true dynamics (plan). Bold: best among the five strategies; a tie at the shown precision goes to the lower std over seeds.

strategy	model error W_1			plan score		
	shift	drift	excess, 1–7	shift	drift	loss, 1–7
frozen	0.11	1.59	4.73	0.94	0.21	2.36
pooled	0.10	1.42	3.85	0.95	0.25	2.11
sliding window	0.16	0.18	0.63	0.89	0.92	0.41
raw likelihood + restart	0.14	0.14	0.39	0.90	0.93	0.25
NBN pipeline	0.10	0.14	0.29	0.95	0.92	0.17
true dynamics	0.07	0.07	–	0.96	0.96	–
NBN pipeline, linear-Gaussian	0.28	0.30	1.53	0.74	0.72	1.63

Speed comparison. Pyro and NumPyro load the trained weights of the NBN’s MDNs, fitted on phase 0, and reproduce the log-densities of nbn to a relative error of about 10^{-5} (single precision); the three engines run the same estimator (likelihood weighting, then the mode on the action grid). Each engine answers the same 10,240 random states, at 2,048 particles each, in $10,240/B$ calls of $B \in \{1, 8, 64, 256, 1024\}$ states; we time the whole workload once (1 seed), after one untimed call for each B . For NumPyro, since it is on JAX, we also count the compilation time: JAX compiles once per batch size, 1.2–1.5 s on the CPU and 2.8–3.4 s on the GPU. Engines are timed on one Linux server with an NVIDIA TITAN X (Pascal) GPU and a CPU with 20 logical cores. Table 3 gives the times. On the GPU the NBN is 1.14 – $1.22\times$ faster than Pyro at every B ; for $B \geq 64$ NumPyro takes 3.5–3.9 s, against 1.5–1.9 s for the NBN, because its compilation outweighs its faster steady state (0.18 s at $B=1024$ without compilation), which overtakes the NBN only after about 25,000 queries per batch size. On the CPU the NBN is 1.03 – $1.47\times$ faster than Pyro, and NumPyro 2.6 – $4.4\times$, its compilation being small next to the CPU times. *Accuracy* (Table 4): each engine plans the 200 evaluation states with 64 to 8,192 particles per state, 5 seeds, drawing its own particles; the three overlap within one s.d. at every particle count, at 2,048 particles they score 0.933–0.938, against 0.961 for the same planner with the true dynamics, and the rest of the gap, which widens with the number of particles, is model error.

Table 3: Time (s) to answer 10,240 planning queries (2,048 particles each) in calls of B states, one seed. NumPyro includes its compilation, given in parentheses. Bold: fastest on each device.

B	GPU (NVIDIA TITAN X)			CPU		
	NBN	Pyro	NumPyro	NBN	Pyro	NumPyro
1	48.1	58.6	22.8 (2.8)	56.1	69.8	26.6 (1.3)
8	6.48	7.91	5.54 (2.8)	28.6	42.0	11.6 (1.5)
64	1.87	2.14	3.86 (3.3)	54.9	61.3	14.2 (1.2)
256	1.59	1.83	3.49 (3.2)	53.9	58.6	13.3 (1.3)
1024	1.50	1.71	3.60 (3.4)	53.1	54.7	12.6 (1.3)

Table 4: Plan score of the same planner in the three engines (CPU) and with the true dynamics, versus particles per state; mean $\pm$ std over 5 seeds, 200 evaluation states. Bold: best of the three engines (a tie goes to the lower std).

particles	NBN	Pyro	NumPyro	true dynamics
64	0.814 $\pm$ 0.016	0.824 $\pm$ 0.022	0.814 $\pm$ 0.009	0.815 $\pm$ 0.010
256	0.879 $\pm$ 0.012	0.876 $\pm$ 0.011	0.874 $\pm$ 0.007	0.892 $\pm$ 0.006
1,024	0.923 $\pm$ 0.006	0.925 $\pm$ 0.003	0.918 $\pm$ 0.007	0.940 $\pm$ 0.006
2,048	0.933 $\pm$ 0.006	0.936 $\pm$ 0.002	0.938 $\pm$ 0.003	0.961 $\pm$ 0.002
4,096	0.943 $\pm$ 0.003	0.945 $\pm$ 0.003	0.944 $\pm$ 0.005	0.975 $\pm$ 0.001
8,192	0.948 $\pm$ 0.004	0.952 $\pm$ 0.003	0.950 $\pm$ 0.003	0.983 $\pm$ 0.002