Hierarchical, Uncertainty-Aware Dynamics Models

Ethan R. Yu
Perception, Action, & Learning Group, GRASP Lab
University of Pennsylvania, United States
`ethanyu@upenn.edu`

Abstract: Model–based reinforcement learning (MBRL) promises data–efficient control by learning a forward dynamics model and using it for planning. Existing uncertainty–aware planners either terminate rollouts or penalise actions when the *aggregate* predictive variance becomes large, implicitly assuming that **all** state dimensions must be forecasted with equal fidelity. We challenge this assumption and propose *Hierarchical, Uncertainty–Aware Dynamics Models* (HUDM): an ensemble transformer that is trained to operate on *arbitrary subsets* of the state vector and a planning procedure that *drops* dimensions whose predictive variance exceeds a threshold. By letting uncertainty dictate which variables matter, HUDM forms an *implicit state hierarchy* on–the–fly, enabling robust and efficient planning even when sensors fail or dynamics are partially unknown. Implementation and videos can be found here: https://github.com/ethayu/HUDM.

Keywords: Uncertainty, Hierarchical, Dynamics Models, MBRL

1 Introduction

Learning forward models of robot dynamics has re–emerged as a central theme in reinforcement learning and control. Yet despite impressive progress, today's model–based planners remain brittle when their predictions become uncertain. Popular approaches such as PETS [1] and MOPO [2] either truncate rollouts or down–weight actions when *any* part of the high–dimensional state drifts outside the training distribution. In real robots equipped with dozens of sensors and latent physical interactions, enforcing uniform confidence across all variables is unrealistic. We hypothesize that uncertainty itself can serve as a *guide* for abstraction: dimensions whose predictive variance explodes are often non–essential (or currently uncontrollable) for the task at hand and can be *ignored* without harming performance. Figure 1 qualitatively illustrates this intuition—while the full state prediction drifts, the end–effector pose stays accurate (with variance 0).

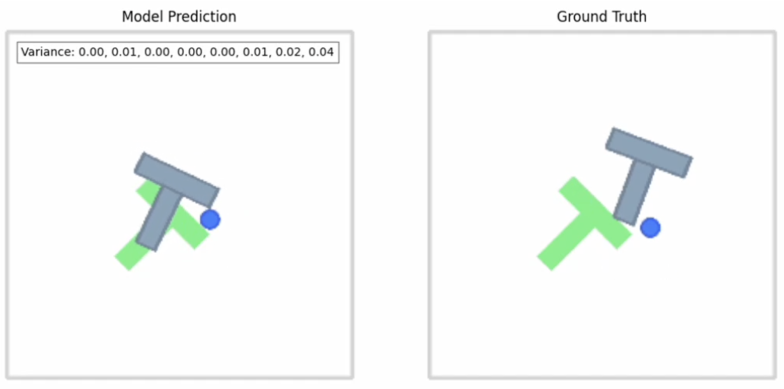


Figure 1: PUSHT w/ nonuniform uncertainty during rollout

This paper asks:

Can we learn a dynamics model and planner that automatically discovers and exploits such task–relevant state hierarchies?

In other words, can an ensemble dynamics model learn to plan over arbitrary subsets of the state—dropping dimensions whose predictive variance becomes large—and thereby induce a useful, emergent hierarchy of abstractions?

Contributions

1. We introduce HUDM, the first dynamics model ensemble explicitly trained to make accurate predictions *given any subset of observed state dimensions*. Each dimension (or logical group) is treated as a token within a transformer encoder and randomly masked during training.
2. We propose **variance–based state dropout**, a planning procedure that removes state dimensions from the rollout once their ensemble variance crosses a threshold, thereby creating a dynamic abstraction hierarchy.
3. We provide a principled scoring mechanism that enables Cross–Entropy Method (CEM) planners to compare action sequences yielding different state subsets.

2 Related Work

Uncertainty in MBRL. Probabilistic Ensembles with Trajectory Sampling (PETS) [1] propagate model variance but still evaluate cost on the full state. MOPO [2] adds a KL penalty when the *aggregate* uncertainty is high, and MBDP [3] discards *entire* rollouts whose total variance exceeds a threshold. All require full–state forecasting. State–feature dropout has been used as a *training augmentation* for policy learning in autonomous driving [4], but not as an *online planning* mechanism.

State abstraction and mixed observability. Mixed–Observability MDPs [5] assume a *fixed* split between fully and partially observed variables. HUDM differs by learning a *data–driven, time–varying* partition based on epistemic variance.

3 Background

3.1 Model-Based Reinforcement Learning

In finite-horizon MBRL an agent seeks a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximises the expected return

$$J(\pi) = \mathbb{E}\Big[\textstyle\sum_{t=0}^{T-1} \gamma^t \, r\big(s_t, a_t\big)\Big], \tag{1}$$

where actions are chosen by the planner and states evolve according to an *unknown* transition $p_*(s_{t+1} \mid s_t, a_t)$. Instead of learning π directly, MBRL learns a *forward model* $\hat{p}_\theta$ and plans in imagination [6]. The classic Dynamics architecture alternates real interaction, model learning, and simulated rollouts [7].

One-step parametric models. Most modern agents fit a conditional density network

$$\hat{p}_\theta(s_{t+1} \mid s_t, a_t) = \mathcal{N}\big(s_t + f_\theta(s_t, a_t),\ \Sigma_\theta(s_t, a_t)\big), \tag{2}$$

where f_θ predicts the *delta* and the covariance Σ_θ captures per-dimension noise.

3.2 Aleatoric *vs.* Epistemic Uncertainty

Uncertainty in prediction can be decomposed as

$$\underbrace{\mathrm{Var}\big[\hat{s}_{t+1}\big]}_{\text{total}} = \overbrace{\mathbb{E}_\theta\big[\mathrm{Var}[\hat{s}_{t+1} \mid \theta]\big]}^{\text{aleatoric}} + \underbrace{\mathrm{Var}_\theta\big[\mathbb{E}[\hat{s}_{t+1} \mid \theta]\big]}_{\text{epistemic}}, \tag{3}$$

known as the *law of total variance* [8]. Aleatoric uncertainty is irreducible measurement or process noise; epistemic uncertainty shrinks with more data and is therefore the quantity of interest for safe planning.

3.3 Deep Ensembles as an Epistemic Estimator

Deep ensembles approximate the distribution over parameters $p(\theta \mid \mathcal{D})$ using N bootstrapped replicas $\{\theta_i\}_{i=1}^N$ [9]. Given input (s_t, a_t)

$$\hat{\mu} = \frac{1}{N}\sum_{i=1}^{N}\big(s_t + f_{\theta_i}(s_t, a_t)\big), \tag{4}$$

$$\hat{\sigma}^2_{\text{epi}} = \frac{1}{N-1}\sum_{i=1}^{N}\big\| f_{\theta_i}(s_t, a_t) - \hat{\mu} + s_t\big\|^2, \tag{5}$$

which is an unbiased estimator of the epistemic term in (3). Ensembles are trivially parallelizable and have become the de-facto uncertainty estimator in PETS [1], MOPO [2], and related algorithms.

3.4 Cross-Entropy Method Planning

The Cross-Entropy Method (CEM) maintains a Gaussian $q_\phi = \mathcal{N}(\mu, \Sigma)$ over action sequences $\mathbf{a}_{0:H-1}$ and updates it by iterative importance sampling [10**?**]. Given costs $\{J^{(k)}\}_{k=1}^K$ from model rollouts, the *elite set* $\mathcal{E} = \arg\operatorname{top}_k(K_{\text{elite}}, -J^{(k)})$ is used to compute

$$\mu \leftarrow \frac{1}{|\mathcal{E}|}\sum_{k\in\mathcal{E}}\mathbf{a}^{(k)}, \quad \Sigma \leftarrow \frac{1}{|\mathcal{E}|}\sum_{k\in\mathcal{E}}\big(\mathbf{a}^{(k)} - \mu\big)\big(\mathbf{a}^{(k)} - \mu\big)^\top. \tag{6}$$

Existing methods penalise $J^{(k)}$ whenever $\max_j \hat{\sigma}^{2,(k)}_{\text{epi},j}$ exceeds a threshold, aborting the entire sequence. In contrast, our method *drops* the offending dimensions (sets $\rho_{t+1,j} = 0$) and continues the rollout, thereby focusing optimisation on the well-modelled subspace.

Connection to MPC. In the limit of one CEM iteration per real control step, the algorithm reduces to Model Predictive Control (MPC) with an adaptive cost surface, as noted by Hansen et al. [11].

4 Method

We describe the subset–robust dynamics ensemble, variance–based dropout, and the modified CEM planner.

Problem set-up. At each decision step we observe a length-H history $\mathcal{H}_t = \{(s_{t-H+1}, a_{t-H+1}), \dots, (s_t, a_t)\}$, where $s_\tau \in \mathbb{R}^D$ and $a_\tau \in \mathbb{R}^{d_a}$. We group the D state dimensions into G semantic tokens $x_\tau^{(g)} \in \mathbb{R}^{d_g}$ ($\sum_g d_g = D$) and introduce one [ACT] token for a_τ. A learned [NEXT] query token is prepended.

Input encoding. Each token is embedded as

$$e_\tau^{(g)} = E_{\text{role}}^{(g)} + E_{\text{time}}(\tau) + E_{\text{rel}}\big(\rho_\tau\big) + \begin{cases}\mathrm{MLP}_s\big(x_\tau^{(g)}\big), & g \le G,\\ \mathrm{MLP}_a\big(a_\tau\big), & g = \texttt{[ACT]},\end{cases}$$

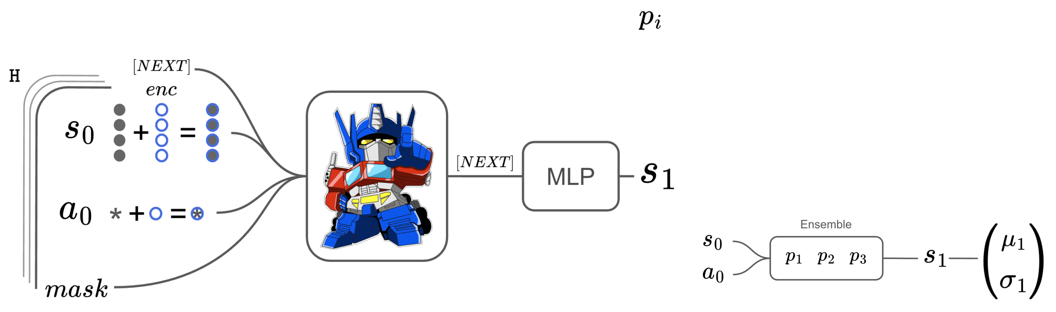
where ρ_τ is a sequence of $\rho_\tau^{(g)} \in \{0,1\}$, which are *reliability bits* (1 = observed, 0 = masked). The reliability embedding allows the model to react differently to genuine observations and placeholders, while the same bit also drives the *key–value padding mask* that prevents masked tokens from influencing other tokens in self-attention.

Transformer encoder. All $(H(G+1)+1)$ tokens are processed by an L-layer Transformer encoder f_θ with pre-norm, n_{head} attention heads and feed-forward width set to `dim_feedforward` (default $4d_{\text{model}}$ in our experiments). The output corresponding to the `[NEXT]` token is interpreted as a latent summary z_t of the history.

Delta prediction and loss gating. The encoder's `[NEXT]` token is decoded by a two-layer MLP to produce a deterministic delta $\Delta\hat{s}_{t+1} \in \mathbb{R}^D$, and the network predicts the next state via residual addition, $\hat{s}_{t+1} = s_t + \Delta\hat{s}_{t+1}$. During training the loss for dimension j is masked out *iff* the corresponding input at time t was unreliable ($\rho_t^{(j)} = 0$). With $\rho_t \in \{0,1\}^D$ the per-sample objective is therefore

$$\mathcal{L} = \frac{1}{\sum_j \rho_t^{(j)}} \sum_{j=1}^{D} \rho_t^{(j)} \left(\hat{s}_{t+1,j} - s_{t+1,j}\right)^2, \tag{4}$$

i.e. mean-squared error over the *observed* dimensions only.



(a) Subset-robust Dynamics Model (b) Ensemble

Figure 2: Ensemble Architecture

4.1 Variance–Based State Dropout

For each dimension j the ensemble provides an epistemic variance $\hat{\sigma}^2_{t+1,j} = \mathrm{Var}_i\big(\hat{s}^{(i)}_{t+1,j}\big)$. If this variance exceeds a user-defined threshold τ at step $t{+}1$, we mark j as *dropped* for the *remainder* of the current rollout of that candidate action sequence:

$$\rho_{t+1,j} = \begin{cases} 0, & \hat{\sigma}^2_{t+1,j} > \tau, \\ \rho_{t,j}, & \text{otherwise.} \end{cases}$$

Because the transformer was trained with random input masks, subsequent evaluations with the updated reliability matrix remain well-posed and do not require any network re-initialization.

4.2 CEM Planning on Variable State Subsets

Refer to Figure 3 for this section. We maintain a sampling distribution q_ϕ over action sequences $\mathbf{a} = (a_t, \ldots, a_{t+H-1})$ and iterate:

1. **Sampling.** Draw K candidates $\{\mathbf{a}^{(k)}\}_{k=1}^{K} \sim q_\phi$.
2. **Rollout with dropout.** Propagate each candidate through the ensemble, yielding a set of final states $\{\hat{s}_T^{(k)}\}$ and reliability masks $\{\rho_T^{(k)}\}$ that may differ across k.

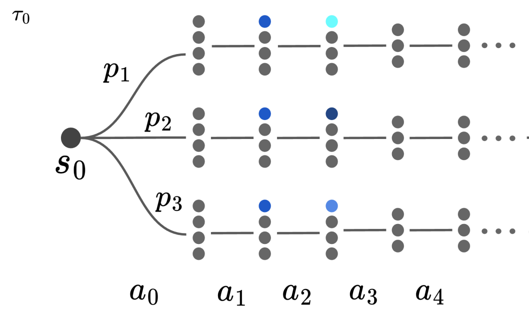
3. **Scoring with optional imputation.**
For dimensions marked *dropped* ($\rho_{T,j}^{(k)} = 0$) we sample M imputations. The candidate cost is aggregated as

$$J^{(k)} = \underset{m=1,\dots,M}{\operatorname{agg}} c\big(s_T^{(k,m)}\big), \qquad \operatorname{agg} \in \{\max, \min, \text{mean}\},$$

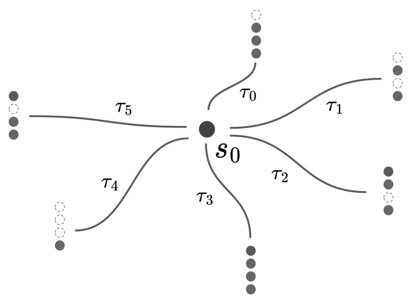
where agg and M are user-controlled hyperparameters. Setting $M = 1$ and `agg=mean` recovers the baseline used in PETS, while `agg=max` (worst-case) or `agg=min` (best-case) give risk-sensitive variants.

4. **CEM update.** The K_{elite} lowest-cost sequences re-estimate the Gaussian parameters (μ, Σ) of q_ϕ.

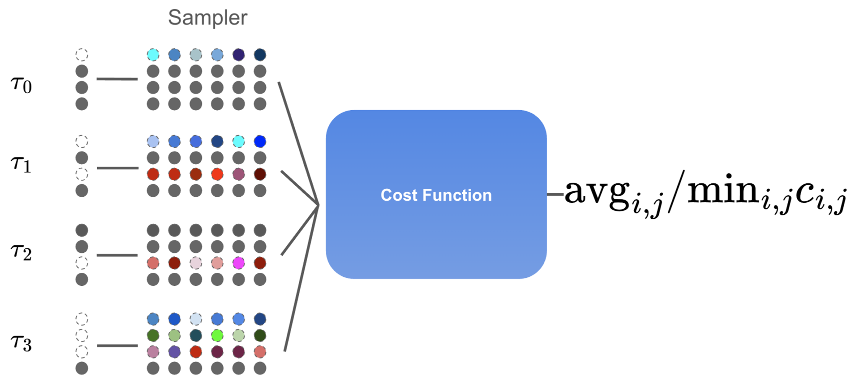
This procedure adds negligible overhead compared with standard PETS, because dropout evaluation re-uses the already computed ensemble predictions.



(a) Rollout with Dropout for τ_0



(b) Final States over Trajectories τ_i



(c) Comparing Trajectories via Imputation

Figure 3: Planning Pipeline

Training protocol. For every experiment we train an **ensemble of** N=5 *Subset–Robust Transformer* models on the PUSHT demonstration dataset, with H=8. Algorithm 1 summarises the procedure; the key elements map one-to-one to the accompanying public code release.

Curriculum masking. For the first E_{warm} epochs no dimensions are masked; the mask probability p then ramps linearly to the target $p_{\max} = 0.5$ over E_{warm} further epochs. This mirrors the code's `make_random_mask` routine and allows the network to learn basic dynamics before tackling heavily missing inputs.

Loss gating. Only dimensions that are let through by the mask ρ are considered in the mean-squared error. Masked-out dimensions (groups) still flow through the transformer and output head, but their gradients are zeroed.

Algorithm 1: Curriculum–masked, bootstrapped training loop

```
Input: dataset $\mathcal{D}$, window $H$, mask target $p_{\max=0.5}$, warm-up $E_{\text{warm}}$=10 epochs
for $i \leftarrow 1$ to $N$ do                                   // independent bootstrap
    $\mathcal{D}^{(i)} \leftarrow$ sample $|\mathcal{D}|$ items with replacement;
    initialize optimizer Adam;
    for epoch $\leftarrow 1$ to $E_{max}$ do
        foreach minibatch $(\mathbf{s}_{1:H}, \mathbf{a}_{1:H}, s_{t+1}) \sim \mathcal{D}^{(i)}$ do
            $p \leftarrow \min\left(p_{\max},\ p_{\max}\,\frac{\max(0, \text{epoch} - E_{\text{warm}})}{E_{\text{warm}}}\right)$;
            draw Bernoulli mask $\rho \sim \text{Bernoulli}(1-p)$ ;        // shape $B \times H \times D$
            $\hat{s}_{t+1} \leftarrow f_{\theta_i}(\mathbf{s}_{1:H}, \mathbf{a}_{1:H}, \rho)$;
            $\mathcal{L} \leftarrow \frac{1}{\sum_j \rho_{H,j}} \sum_{j=1}^{D} \rho_{H,j}\left(\hat{s}_{t+1,j} - s_{t+1,j}\right)^2$;
            update $\theta_i$ with Adam step;
```

Bootstrapped ensembles. Each ensemble member is trained on a fixed bootstrap resample of the dataset (`Subset` in the script), guaranteeing statistical independence and enabling unbiased epistemic variance estimation.

Optimisation and early stopping. All models use AdamW with learning rate 5×10^{-4}, minibatches of 1000, and a shared validation set (20 % of data). The best epoch is chosen by validation MSE and checkpointed.

5 Results

We first assess how accurately each dynamics model predicts expert trajectories from the PUSHT dataset, then evaluate whether the ensemble variance is sufficiently calibrated to serve as a dropping criterion for the downstream planner.

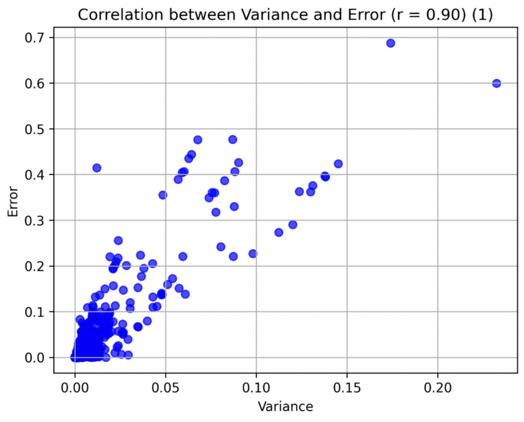
5.1 Predictive rollouts

Note that we opt to not use quantitative but rather use qualitative assessments of performance when comparing dynamics models, since models just need to pass the "eye test" to be useful for downstream planning.

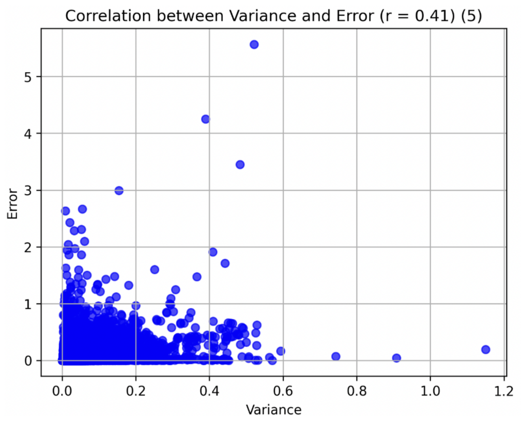
Teacher–forced vs. free rollout. When *teacher forcing* is used—i.e. ground-truth ($H = 8$) history is fed at every step—our transformer achieves near–perfect accuracy (sample 1, sample 2). However, in the free-rollout setting the same model drifts (sample 1, sample 2), indicating that error compounds through the autoregressive history window. A vanilla *MLP ensemble* trained on the full state performs better in the latter instance, suggesting that the architectural inductive bias—not the dataset—is the limiting factor.

5.2 Uncertainty calibration

To gauge whether ensemble variance can serve as a reliable dropping score, we plot variance against error and compute the Pearson correlation coefficient (PCC) per-dimension (Figure 4). For some states the PCC is relatively high, but for others there are a dearth of low variance, high error instances. We are still trying to improve this part of the dynamics modeling, investigating the use of Bayesian MLP's to hopefully model aleatoric uncertainty as well (which we suspect are what is causing the low variance high error instances).



(a) Variance vs. Error for EOF x-position



(b) Variance vs. Error for T-block x-position

Figure 4: Variance vs. Error Plots

5.3 Planning with CEM and variance dropping

Preliminary CEM experiments, with and without dropout, did not provide particularly meaningful results (sample rollout) since the dynamics don't perform well; however, we hope to perform ablations on aggregation method and imputation count once dynamics get to a good spot.

6 Conclusion

We presented HUDM, a hierarchical, uncertainty–aware dynamics model that plans with arbitrary state subsets.

Further work involves calibrating the error and variances, successfully training the subset-aware transformer model, and extending to more tasks and domains. Critically, we hope to uncover our implicit state hierarchy hypothesis in this process. Beyond this, further work could include extending to multi–task settings and learning the variance threshold τ via meta–optimization.

Acknowledgments

This work was initiated in collaboration with Aurora Qian; we thank Siddharth Ancha, Dinesh Jayaraman, and Nicholas Roy for discussions and insightful feedback.

References

[1] K. Chua, R. Calandra, R. McAllister, and S. Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[2] T. Yu, A. Kumar, G. Tucker, and S. Levine. MOPO: Model-based offline policy optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[3] Y. Zhang, J. Frey, M. Laskin, and P. Abbeel. Model-based double-dropout planning. In *Conference on Robot Learning (CoRL)*, 2024. arXiv:2402.01234.

[4] A. Khaitan, D. Dwibedi, Y. Chebotar, and S. Levine. State dropout-based curriculum reinforcement learning for autonomous driving. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2022.

[5] S. J. Ong, S. J. Png, D. Hsu, and W. S. Lee. Planning under uncertainty for robotic tasks with mixed observability. In *Robotics: Science and Systems (RSS)*, 2010.

[6] T. M. Moerland, J. Broekens, A. Plaat, and C. M. Jonker. Model-based reinforcement learning: A survey. *arXiv preprint arXiv:2006.16712*, 2020.

[7] R. S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bulletin*, 2(4):160–163, 1991.

[8] E. Hüllermeier and W. Waegeman. Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods. *Machine Learning*, 110:457–506, 2021.

[9] B. Lakshminarayanan, A. Pritzel, and C. Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[10] R. Y. Rubinstein. The cross-entropy method for combinatorial and continuous optimization. In *Methodology and Computing in Applied Probability*, pages 127–190, 1999.

[11] N. Hansen, A. Srinivas, and P. Abbeel. Decoupling exploration and exploitation for cem planning. In *Conference on Robot Learning (CoRL)*, 2022. arXiv:2210.XXXX.