CIS 6280 · FALL 2026

WORLD MODELS

LECTURE 11

Dreamer, Diffusion and Flow Matching

Jiatao Gu

University of Pennsylvania · October 13, 2026

Course announcements

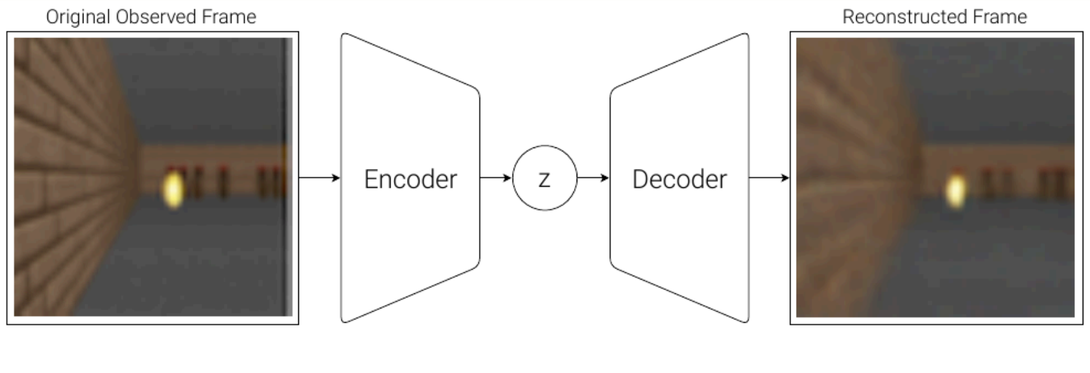
Final project

Project information will be released on **October 16**.

Assignment 2

Due **October 19**.

Learning a latent representation



World Models (2018): encode an observation and reconstruct it from its latent representation.

VAE

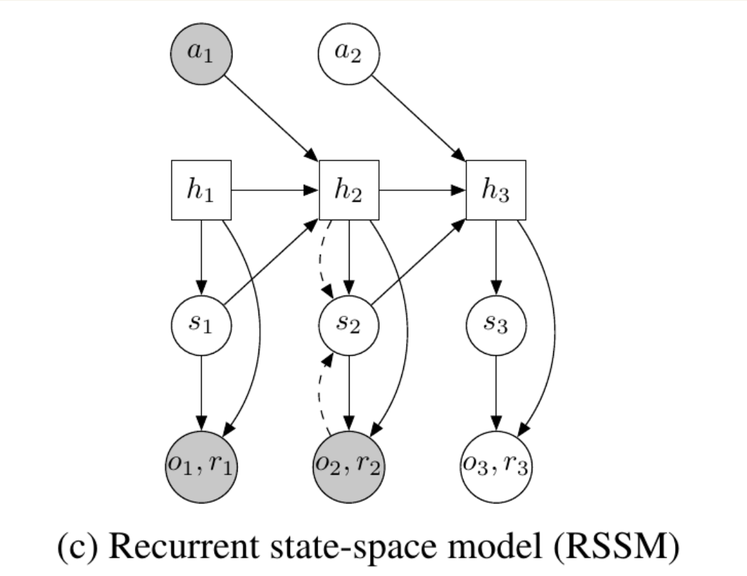
Infer a latent distribution and reconstruct the observation.

JEPA

Predict target features supplied by another encoder.

The training objective shapes what the representation preserves.

Observing and predicting with an RSSM



(c) Recurrent state-space model (RSSM)

PlaNet: recurrent memory, stochastic state, actions, observations and rewards.

$$x_t = (h_t, s_t)$$

Model state = recurrent memory + stochastic state.

When an observation arrives

Use real sensory evidence to update the **posterior**.

When predicting a future

Supply actions and use the transition **prior**. No future observations are available.

History informs the current state. Actions determine its predicted successors.

Hafner et al., PlaNet (2019)

Learning from recorded sequences

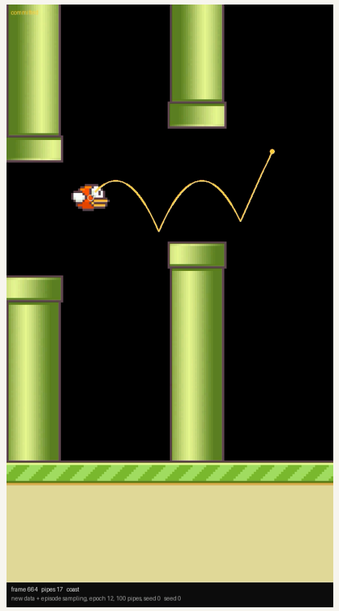
Recorded observations and actions → Infer model states → Predict observed outcomes

Training signal	What the RSSM learns
Observation reconstruction	Retain information about sensory evidence.
Reward prediction	Connect predicted states to task feedback.
Posterior–prior consistency	Make dynamics predictions agree with observation-informed states.

These updates change the world model. The recorded actions supply its inputs.

Hafner et al., PlaNet (2019)

Planning with a learned world model



Our earlier Flappy Bird demo: the yellow path shows predicted motion under a selected plan.

At the current inferred state:

1. Propose candidate action sequences.
2. Predict their consequences.
3. Score the predicted outcomes and improve the candidates.

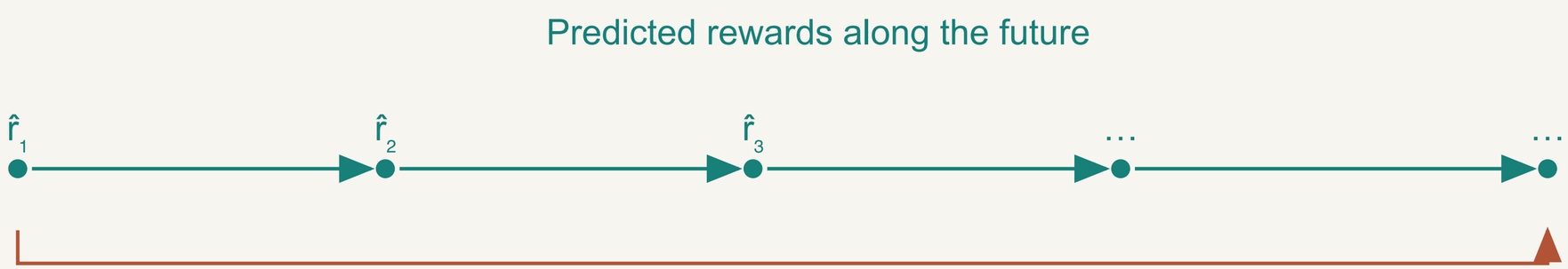
After the search

Execute the first action. Observe, update the state, and plan again.

Planning changes candidate actions while the learned world model stays fixed.

Instructor-provided Flappy Bird demo, shown in Lecture 10 · Hafner et al., PlaNet (2019)

Return: total future reward



The quantity an action should improve

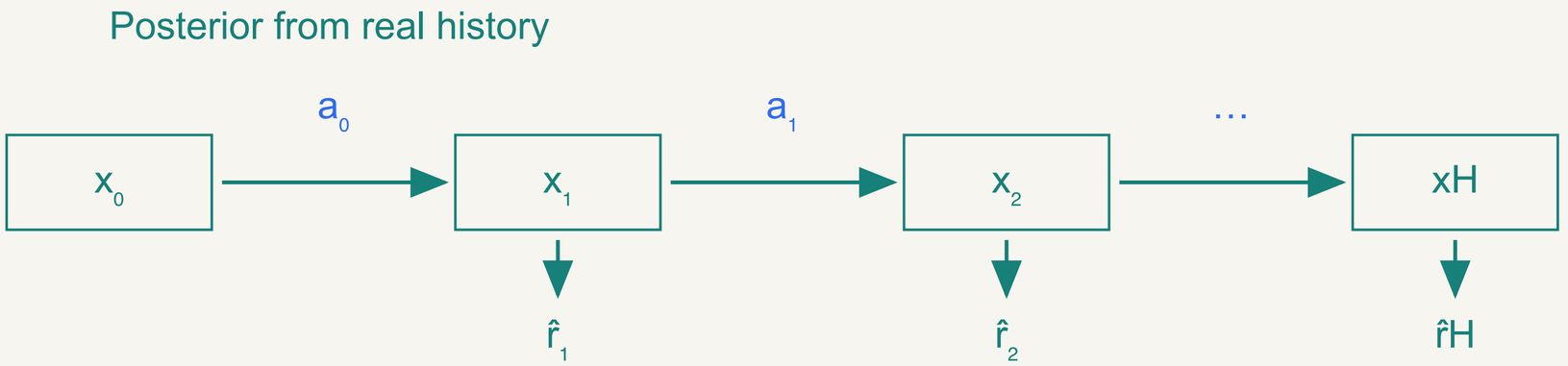
$$\mathcal{R}_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

Return combines rewards over future steps. Discounting determines how much later rewards contribute.

To compare predicted futures, estimate their return.

Hafner et al., Dreamer, ICLR 2020

Predicting a short future



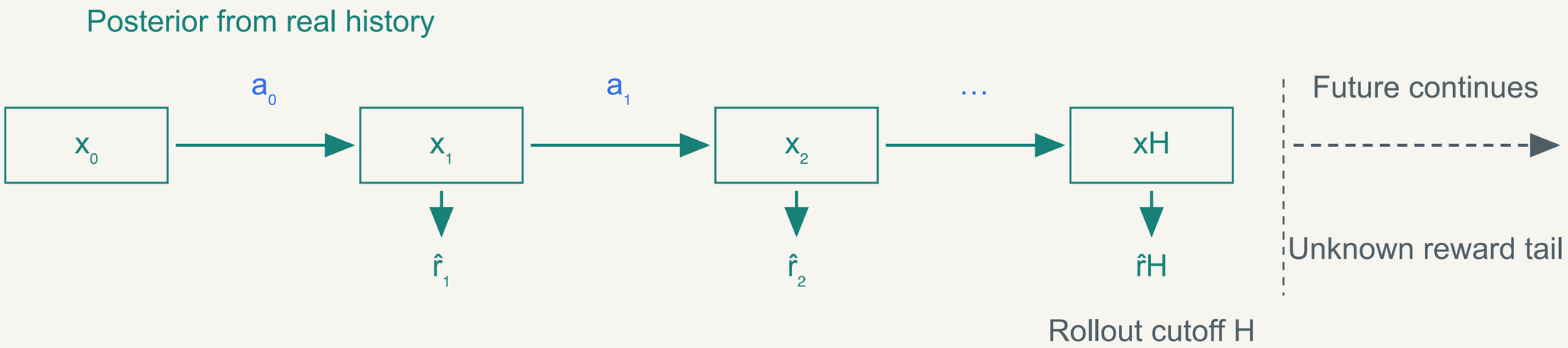
$$x_{t+1} \sim p_\theta(\cdot \mid x_t, a_t), \qquad \hat{r}_{t+1} = r_\theta(x_{t+1})$$

Infer the starting state from real history. Supply actions and predict states and rewards.

Local rollout indices: $x_0, \ldots, x_H$. Future real observations are unavailable.

Hafner et al., Dreamer, ICLR 2020

The rollout ends before the task

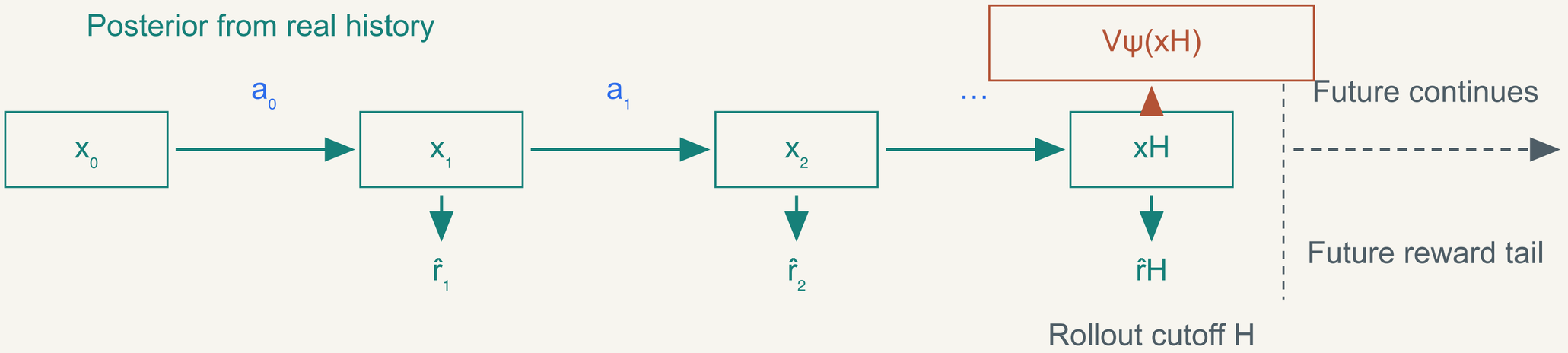


At x_H, explicit prediction stops. Rewards can continue after that state.

A computational cutoff leaves a missing tail of return.

Hafner et al., Dreamer, ICLR 2020

Value estimates the remaining return



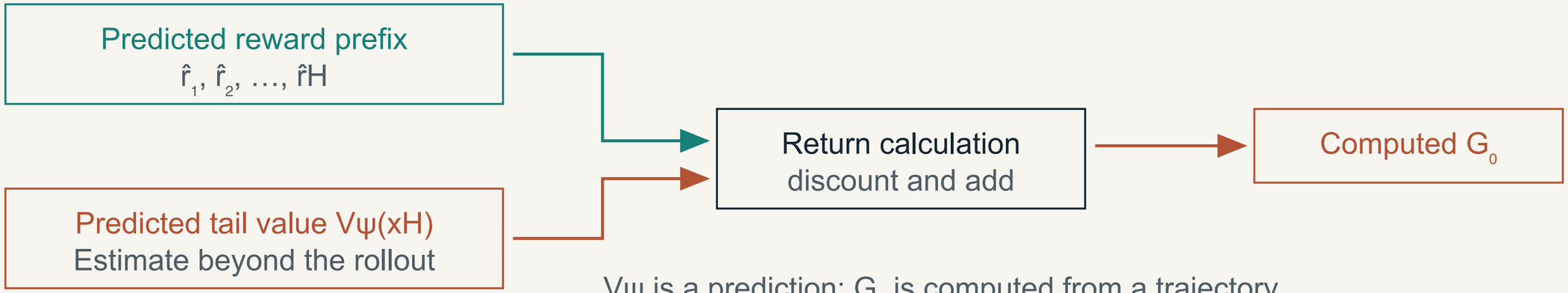
The world model predicts a sequence of future states and rewards.

The **value model** predicts one number: the expected reward remaining from a state.

V(x) summarizes the future beyond the rollout. It assumes a specified way of continuing to act.

Hafner et al., Dreamer, ICLR 2020

Predicted rewards plus tail value



Vψ is a prediction; G_0 is computed from a trajectory

$$G_0 = \underbrace{\hat{r}_1 + \gamma \hat{r}_2 + \cdots + \gamma^{H-1} \hat{r}_H}_{\text{rewards inside the rollout}} + \underbrace{\gamma^H V_\psi(x_H)}_{\text{estimated reward after the rollout}}$$

G is calculated from a rollout. V is predicted directly from a state.

The rollout and value prediction refer to the same way of continuing to act.

Hafner et al., Dreamer, ICLR 2020

Training the value model

Prediction

$$V_\psi(x_t)$$

The number produced by the value model.

Training target

$$G_t$$

The return estimate calculated from a rollout.

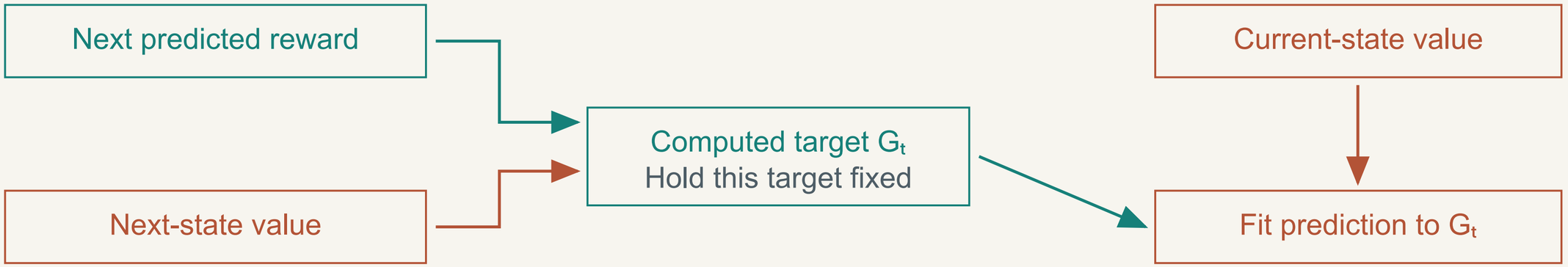
Train the prediction to match the target. This is a regression problem.

$$\mathcal{L}_V(\psi) = \mathbb{E}\left[\frac{1}{2}\left(V_\psi(x_t) - \mathrm{sg}(G_t)\right)^2\right]$$

Square the prediction error and average over training states. Keep G fixed during this update.

Hafner et al., Dreamer, ICLR 2020

Where the training target comes from



$$G_t = \underbrace{\hat{r}_{t+1}}_{\text{next predicted reward}} + \gamma \underbrace{V_{\text{old}}(x_{t+1})}_{\text{predicted reward remaining after it}}$$

Predict one transition. Add its reward to the value already predicted at the next state.

A longer rollout uses more predicted rewards before asking the value model for the tail.

Hafner et al., Dreamer, ICLR 2020

Why a target can contain a value prediction

The current value should agree with **the next reward plus the value of the next state**.

$$V(x_t) \approx \hat{r}_{t+1} + \gamma V(x_{t+1})$$

What anchors this relation?

The reward predictions come from a world model trained on real rewards.

What can still be wrong?

The model and the next-state value can both make prediction errors.

Fitting this relation transfers reward information between states. It does not make the predictions automatically correct.

Value learning repeats

1. Calculate targets

Roll out the model and use the current value prediction for the remaining tail.

2. Fit the value model

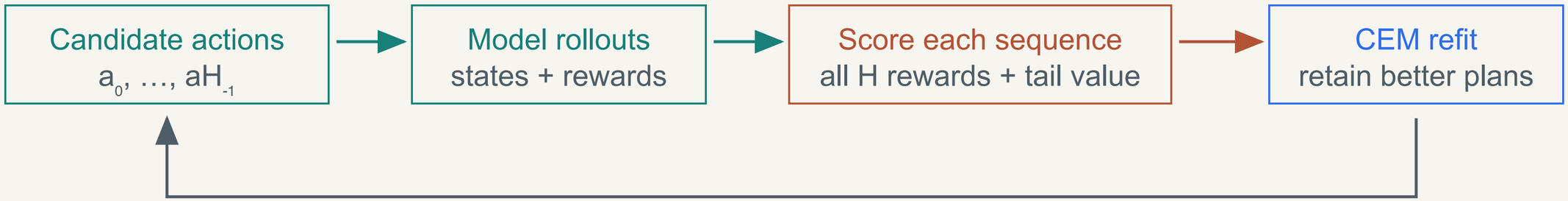
Hold those targets fixed. Update the value predictions toward them.

Recompute targets with the updated value model, then fit again.

G is a temporary training target. V is the reusable predictor we are learning.

Hafner et al., Dreamer, ICLR 2020

CEM planning with a value model



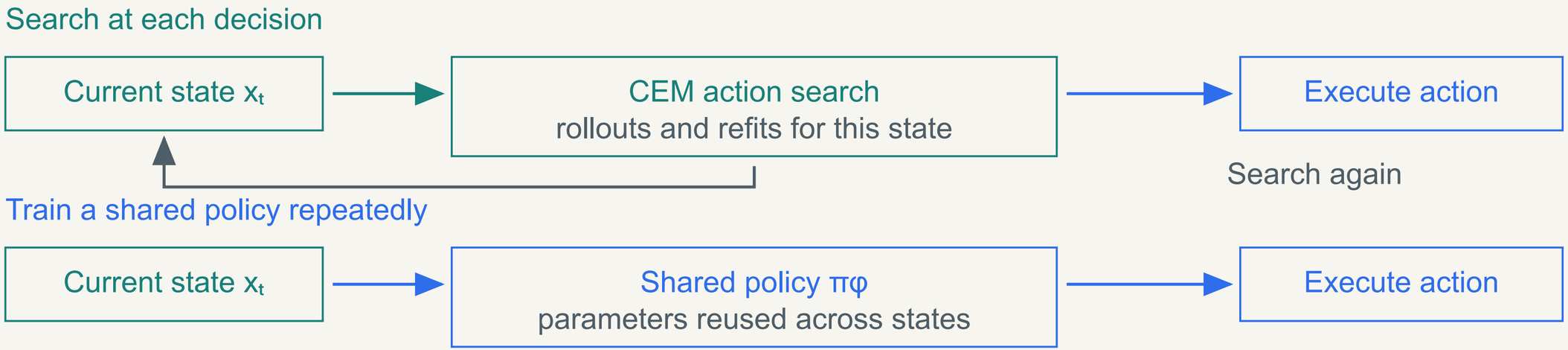
$$\text{Score}(U) = \sum_{t=0}^{H-1} \gamma^t \hat{r}_{t+1} + \gamma^H V_\psi(x_H)$$

Score the candidate's full action prefix, then use value for its subsequent continuation.

Illustrative value-augmented CEM, not the original PlaNet scoring rule.

Hafner et al., Dreamer, ICLR 2020

Learning a reusable policy



$$a_t \sim \pi_\phi(\cdot \mid x_t)$$

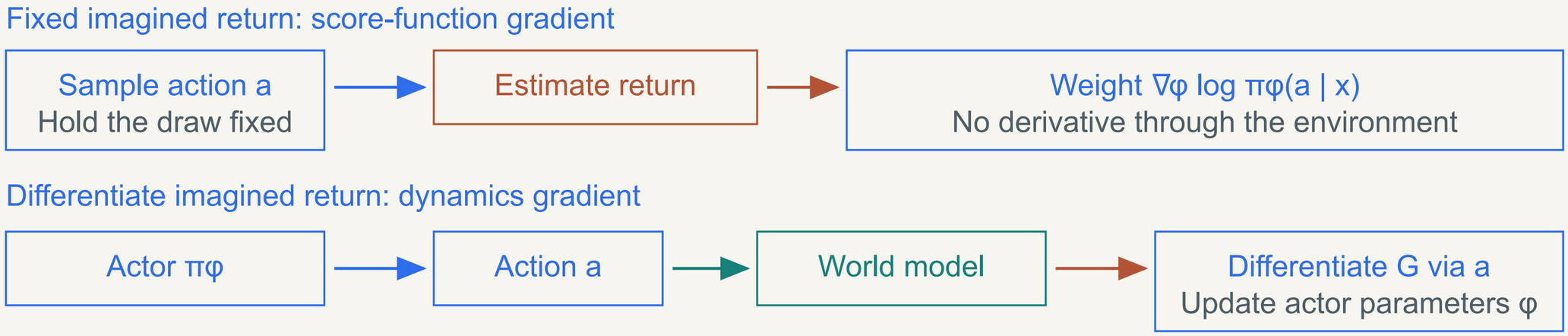
Optimize shared parameters during training. Reuse the policy to choose actions at execution.

Policy = actor. Value model = critic. These are their roles in actor–critic learning.

Imagined actions and value targets now follow the **current policy**.

Hafner et al., Dreamer, ICLR 2020

Two actor updates using the same world model



Two ways to estimate the gradient of expected return

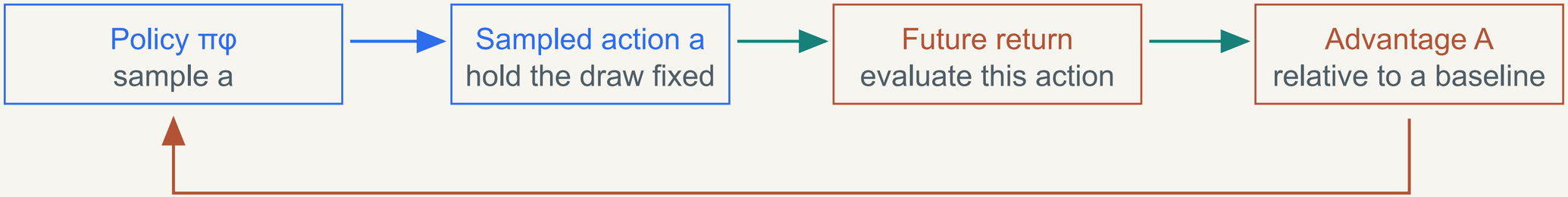
Both routes use **the current policy's imagined trajectories and return estimates**.

Policy gradient: use return to weight the log probability of a fixed sampled action.

Dynamics gradient: differentiate return through the action and world model.

Hafner et al., Dreamer, ICLR 2020

A stochastic policy-gradient surrogate



Positive A: raise log πφ(a | x); negative A: lower it

Update the action probability without differentiating the environment

$$\mathcal{L}_{\mathrm{SF}}(\phi) = -\mathbb{E}_{\text{fixed batch}}\left[\log \pi_\phi(a_t \mid x_t)\, A_t\right]$$

Advantage $A_t = G_t - V_\psi(x_t)$ compares the imagined return with the value prediction.

Hold sampled states, actions and advantages fixed. Update ϕ through log probabilities.

Dreamer: differentiate the imagined return

Forward: imagine consequences and calculate return



Backward: return gradients reach the policy through the model

$$\mathcal{L}_{\text{actor}}(\phi) = -\mathbb{E}[G_0]$$

Change the action. The world model changes the predicted future. The return changes with it.

Backpropagate through that computation to update the policy parameters ϕ.

Hafner et al., Dreamer, ICLR 2020

Two gradient routes for the policy update

Score-function gradient

Treat imagined return as a fixed learning signal.

$$-\log \pi_\phi(a_t \mid x_t)\ \mathrm{sg}(A_t)$$

Dynamics gradient

Retain how imagined return changes with the action.

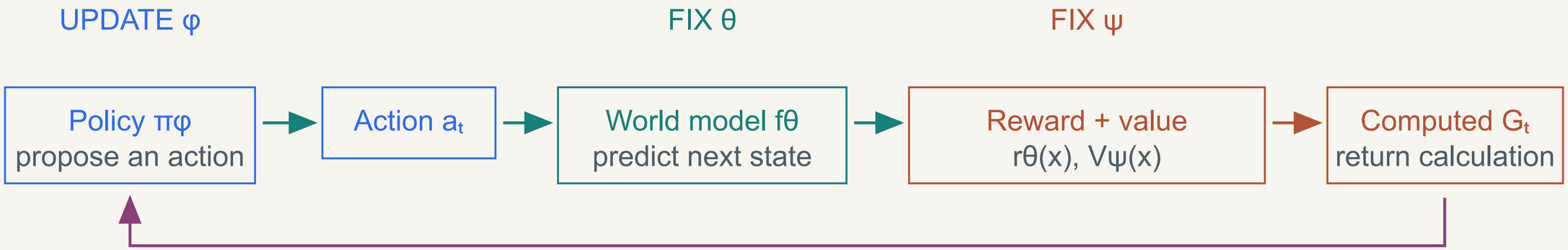
$$-G_t$$

Same world model, value model and imagined experience. Different actor-gradient computations.

Original continuous-action Dreamer uses the dynamics-gradient route.

Hafner et al., Dreamer, ICLR 2020

Fixed parameters, active action derivatives

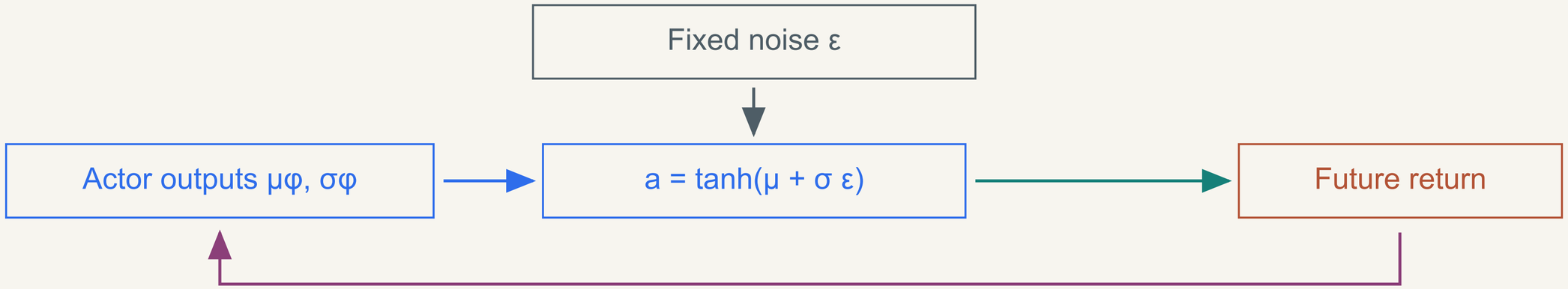


Update ϕ. Keep world-model parameters θ and critic parameters ψ fixed.

Retain input derivatives through dynamics, rewards and values.

Original implementation: detach each actor's state input, while retaining state propagation through the model.

Differentiating a sampled action

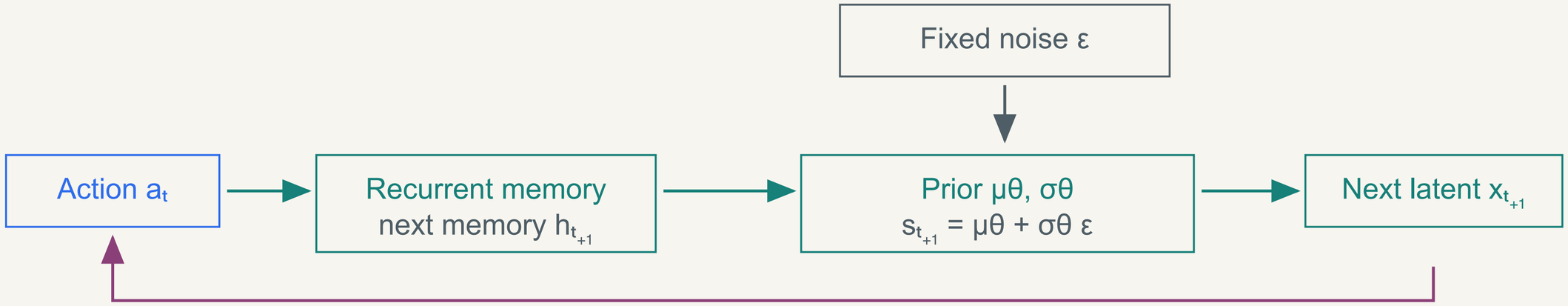


Hold ε fixed; differentiate through the actor mean and scale

$$a_t = \tanh(\mu_\phi(\mathrm{sg}(x_t)) + \sigma_\phi(\mathrm{sg}(x_t)) \odot \epsilon_t)\,, \qquad \epsilon_t \sim \mathcal{N}(0, I)$$

For a fixed noise draw, the action is differentiable with respect to the actor’s outputs.

Differentiating a stochastic latent transition



FIX θ; keep derivatives from the next latent back to the action

$$s_{t+1} = \mu_\theta(h_{t+1}) + \sigma_\theta(h_{t+1}) \odot \xi_t, \quad \xi_t \sim \mathcal{N}(0, I)$$

Full RSSM state: $x_t = (h_t, s_t)$. Reparameterization retains the action-to-future-state derivative.

Hafner et al., Dreamer, ICLR 2020

A learned Dreamer agent



Original Dreamer demonstration: a learned actor executes a visual-control task.

Training

Improve actor and critic using imagined returns.

Execution

Infer the state from real observations, then choose the actor’s action.

Training the three components

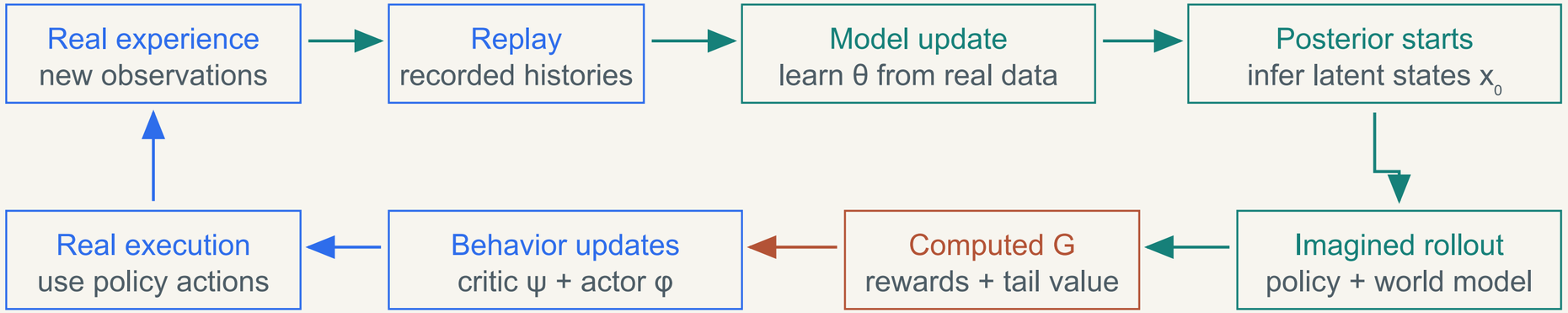
Training stage	Signal	Update
World model	Real observations and rewards	θ
Value	Imagined return target	ψ
Policy	Expected imagined return	φ

World model: fit real experience. Critic: fit fixed return targets. Actor: improve imagined return.

Alternate updates with different objectives and parameter sets.

Hafner et al., Dreamer, ICLR 2020

The complete Dreamer loop



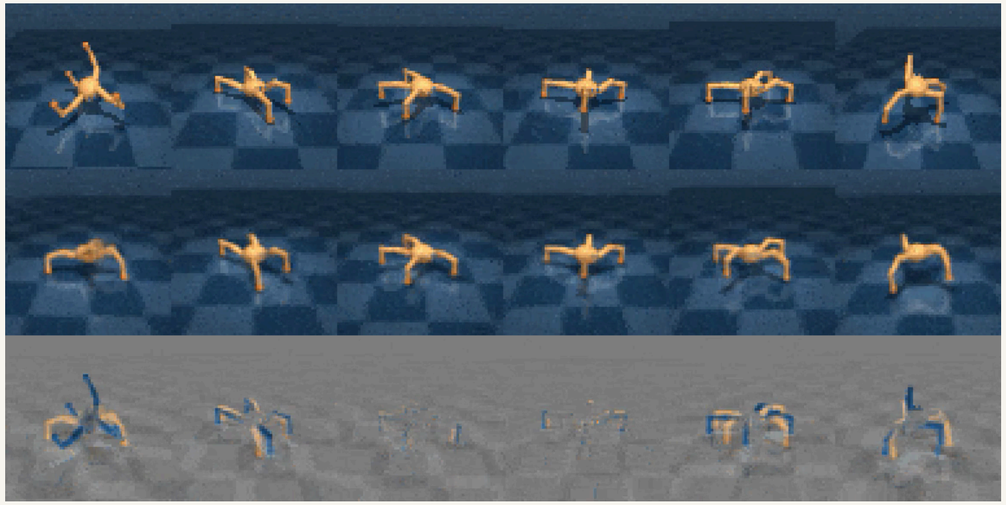
Real experience improves the model; imagination improves behavior

Real replay supplies posterior starting states. The model and actor generate imagined trajectories and returns.

After actor and critic updates, real execution supplies new evidence for replay.

Hafner et al., Dreamer, ICLR 2020

Prediction error affects behavior learning

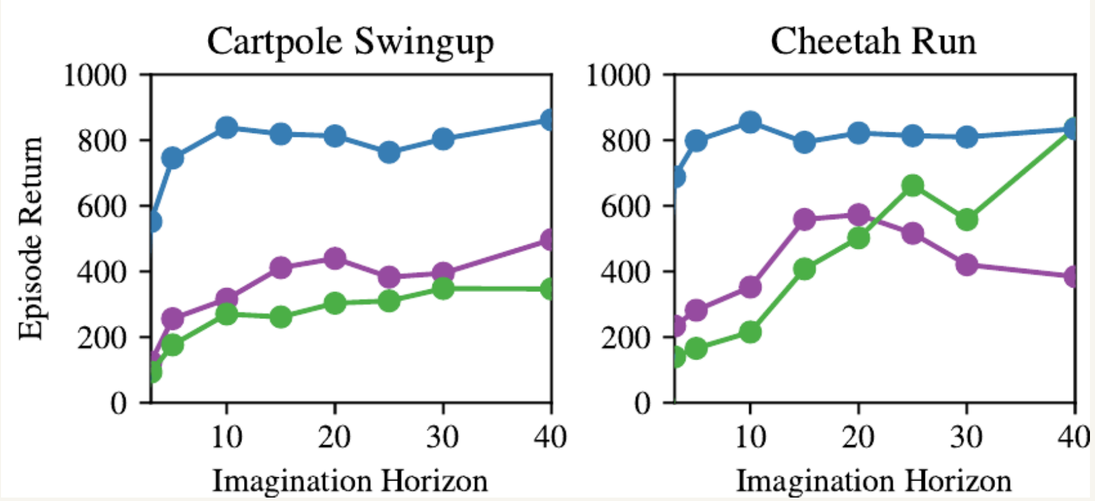


Original model evaluation with recorded future actions: held-out observations, decoded predictions and differences.

Errors in predicted states or rewards affect targets used by both critic and actor.

This evaluation uses recorded future actions. It is not a rollout of newly selected policy actions.

Value prediction and the imagination horizon

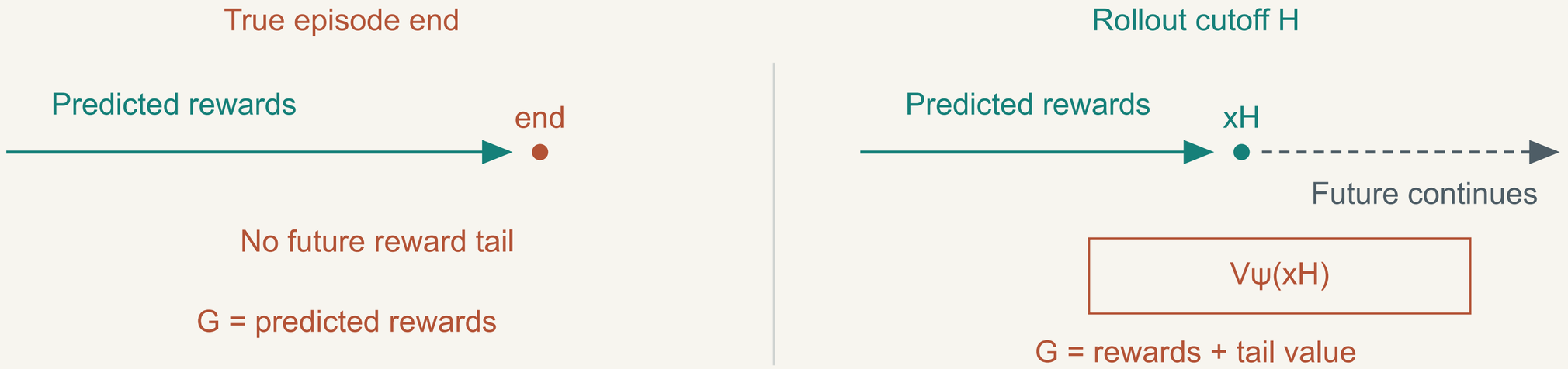


Selected panels from Dreamer Figure 4. Horizontal: imagination horizon. Vertical: episode return.

Dreamer with value Actor without value PlaNet planning

In these tasks, tail-value prediction reduces sensitivity to the rollout horizon.

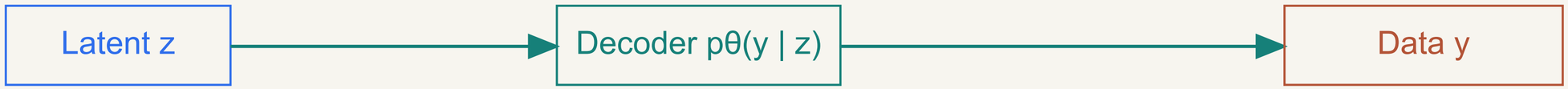
Task termination and rollout cutoff



The task ends: no future reward remains.

Only prediction stops: use the value model to estimate the future that remains.

Variational autoencoders



Sample a latent, then generate the observation

Inference

$$q_\phi(z \mid y)$$

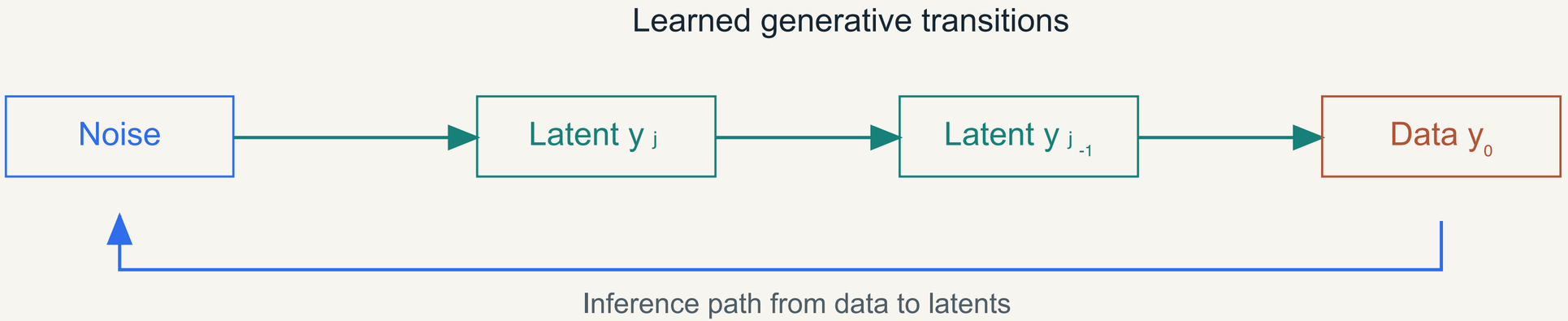
Generation

$$p(z)\, p_\theta(y \mid z)$$

A latent variable connects a simple prior to the data distribution.

Kingma and Welling, VAE (2014)

A hierarchy of latent variables



$$p_\theta(y_{0:J}) = p(y_J) \prod_{j=1}^{J} p_\theta(y_{j-1} \mid y_j)$$

The data are y_0. The intermediate $y_1, \ldots, y_J$ are latent variables.

Generate through a sequence of conditional distributions.

Progressive image corruption



Data | Less signal | Mixed | Mostly noise | Noise

Choose an inference process that gradually removes the data signal.

A Gaussian forward transition



Fixed forward corruption, adapted from the CIS 7000 diffusion lecture.

$$q(y_j \mid y_{j-1}) = \mathcal{N}\left(\sqrt{1-\beta_j}\, y_{j-1},\ \beta_j I\right)$$

Retained signal

$\sqrt{1-\beta_j}\, y_{j-1}$ scales the previous sample.

The schedule $\beta_1, \ldots, \beta_J$ is fixed before training.

Added noise

$\beta_j I$ sets the Gaussian covariance.

The index j counts corruption steps, not environment time.

Ho et al., DDPM (2020) · CVPR diffusion tutorial (credited in CIS 7000)

Sampling a chosen noise level

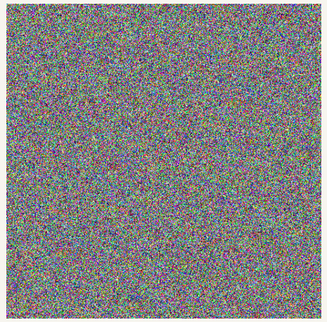
$$\alpha_j = 1 - \beta_j, \qquad \bar{\alpha}_j = \prod_{i=1}^{j} \alpha_i$$

$$y_j = \sqrt{\bar{\alpha}_j}\, y_0 + \sqrt{1 - \bar{\alpha}_j}\, \epsilon, \qquad \epsilon \sim \mathcal{N}(0, I)$$



Clean data y_0

+



Sampled noise ε

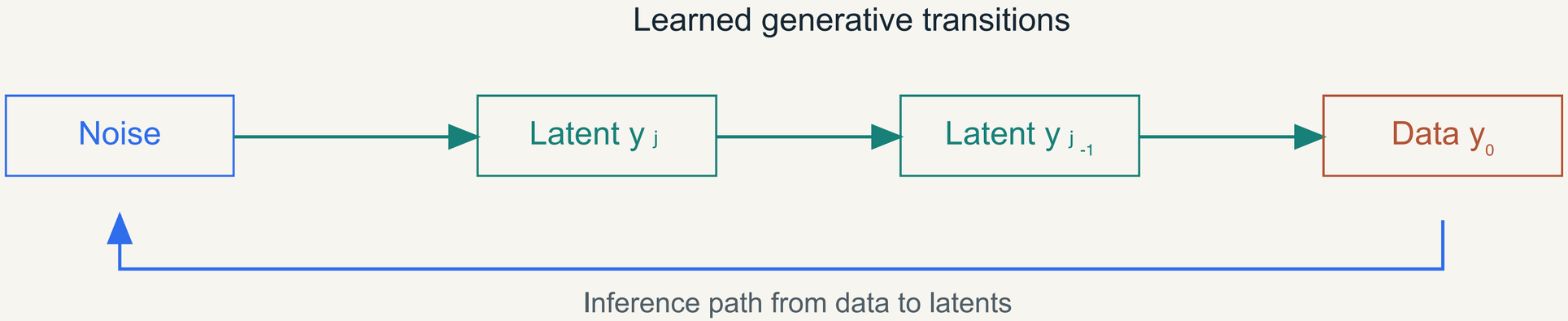
→



Noisy training input y_j

Training can sample any noise level directly.

Learning the reverse process



$$p_\theta(y_{j-1} \mid y_j) = \mathcal{N}\big(\mu_\theta(y_j, j),\, \sigma_j^2 I\big)\,, \qquad j \geq 2$$

Start from $y_J \sim \mathcal{N}(0, I)$. A shared network predicts the reverse mean at each noise level.

Learning supplies the information missing from a noisy sample.

VAE recap: the ELBO

$$\log p_\theta(y) \geq \mathrm{ELBO}(y)$$

$$\mathrm{ELBO}(y) = \mathbb{E}_{q_\phi(z|y)}[\log p_\theta(y \mid z)] - D_{\mathrm{KL}}(q_\phi(z \mid y) \,\|\, p(z))$$

Reconstruction

Use the inferred latent to explain the observed data.

KL to the prior

Compare the inferred latent distribution with the generative prior.

Maximize the ELBO. Equivalently, minimize the negative ELBO.

The ELBO for a latent chain

$$\log p_\theta(y_0) \geq \mathrm{ELBO}(y_0)$$

$$\mathrm{ELBO}(y_0) = \mathbb{E}_{q(y_{1:J}|y_0)}[\log p_\theta(y_{0:J}) - \log q(y_{1:J} \mid y_0)]$$

Inference distribution

The fixed forward process $q(y_{1:J} \mid y_0)$.

Generative model

The prior and learned reverse conditionals $p_\theta(y_{0:J})$.

$$\mathcal{L}_{\mathrm{negELBO}}(y_0) = -\mathrm{ELBO}(y_0)$$

The same ELBO principle applies. Training minimizes its negative.

Substituting the diffusion model into the ELBO

Generative model: learned reverse transitions

$$p_\theta(y_{0:J}) = p(y_J)\prod_{j=1}^{J} p_\theta(y_{j-1} \mid y_j)$$

Inference: fixed forward transitions

$$q(y_{1:J} \mid y_0) = \prod_{j=1}^{J} q(y_j \mid y_{j-1})$$

$$\mathrm{ELBO}(y_0) = \mathbb{E}_q\left[\log p(y_J) + \sum_{j=1}^{J} \log p_\theta(y_{j-1} \mid y_j) - \sum_{j=1}^{J} \log q(y_j \mid y_{j-1})\right]$$

Each term now corresponds to a transition in one of the two chains.

Only the reverse model depends on the learned parameters θ.

Matching the reverse transitions

$$q(y_{1:J} \mid y_0) = q(y_J \mid y_0) \prod_{j=2}^{J} q(y_{j-1} \mid y_j, y_0)$$

$$\mathcal{L}_{\text{negELBO}} = \mathcal{L}_{\text{rec}} + \sum_{j=2}^{J} \mathcal{L}_j + \mathcal{L}_{\text{prior}}$$

$$\mathcal{L}_j = \mathbb{E}_{q(y_j|y_0)}[D_{\text{KL}}(q(y_{j-1} \mid y_j, y_0) \,\|\, p_\theta(y_{j-1} \mid y_j))]$$

Target during training

The posterior $q(y_{j-1} \mid y_j, y_0)$ uses the known clean sample.

Prediction during generation

The model $p_\theta(y_{j-1} \mid y_j)$ has only the current noisy sample.

$\mathcal{L}_{\text{rec}}$: data reconstruction. $\mathcal{L}_{\text{prior}}$: final-noise prior matching.

Given the clean training sample, the Gaussian reverse target is tractable.

The Gaussian target for one reverse step

$$q(y_{j-1} \mid y_j, y_0) \propto q(y_j \mid y_{j-1})\, q(y_{j-1} \mid y_0), \qquad j \geq 2$$

Multiply the Gaussian forward transition by the Gaussian marginal. Complete the square.

$$q(y_{j-1} \mid y_j, y_0) = \mathcal{N}(\tilde{\mu}_j, \tilde{\beta}_j I)$$

$$\tilde{\mu}_j = \underbrace{\frac{\sqrt{\bar{\alpha}_{j-1}}\beta_j}{1-\bar{\alpha}_j}}_{\text{clean-sample coefficient}} y_0 + \underbrace{\frac{\sqrt{\alpha_j}(1-\bar{\alpha}_{j-1})}{1-\bar{\alpha}_j}}_{\text{noisy-sample coefficient}} y_j$$

$$\tilde{\beta}_j = \frac{1-\bar{\alpha}_{j-1}}{1-\bar{\alpha}_j}\,\beta_j$$

The target mean is computable from the clean and noisy training pair.

Ho et al., DDPM (2020)

Gaussian KL becomes mean prediction error

Training target

$$q = \mathcal{N}(\tilde{\mu}_j, \tilde{\beta}_j I)$$

Learned reverse model

$$p_\theta = \mathcal{N}(\mu_\theta, \sigma_j^2 I)$$

$$D_{\mathrm{KL}}(q \,\|\, p_\theta) = \frac{\|\tilde{\mu}_j - \mu_\theta(y_j, j)\|^2}{2\sigma_j^2} + \underbrace{\frac{d}{2}\left[\frac{\tilde{\beta}_j}{\sigma_j^2} - 1 - \log\frac{\tilde{\beta}_j}{\sigma_j^2}\right]}_{\text{independent of } \theta \text{ for fixed variances}}$$

Assume fixed positive reverse variances σ_j^2. d is the sample dimension.

Learning the reverse distribution reduces to predicting its mean.

Ho et al., DDPM (2020)

The ELBO gives a weighted regression loss

$$\mathcal{L}_{\text{negELBO}} = \mathcal{L}_{\text{rec}} + \sum_{j=2}^{J} \mathbb{E}_{q(y_j|y_0)} \left[\underbrace{\frac{1}{2\sigma_j^2}}_{\text{noise-level weight}} \|\tilde{\mu}_j - \mu_\theta(y_j, j)\|^2 \right] + C$$

Exact negative ELBO

Keep the variance-dependent weights and the reconstruction endpoint.

C contains terms independent of θ.

Reweighted training objective

Changing the level weights changes the emphasis of the loss.

The weighted mean error comes from the ELBO; reweighting is a separate choice.

Ho et al., DDPM (2020)

What should the network predict?

$$y_j = a_j y_0 + b_j \epsilon, \quad a_j = \sqrt{\bar{\alpha}_j}, \quad b_j = \sqrt{1 - \bar{\alpha}_j}$$

Output	Training target	Meaning
μ_θ	$\tilde{\mu}_j(y_j, y_0)$	Mean of the next reverse step
ϵ_θ	ϵ	Noise added to the clean sample
$\hat{y}_{0,\theta}$ (x_0 prediction)	y_0	The clean data sample
Diffusion v_θ	$a_j \epsilon - b_j y_0$	A signal–noise combination

The algorithms below use ϵ-prediction.

$$\mu_\theta = \frac{1}{\sqrt{\alpha_j}} \left(y_j - \frac{\beta_j}{\sqrt{1 - \bar{\alpha}_j}} \epsilon_\theta \right)$$

Equivalent output parameterizations. Plain MSE gives different level weights.

Ho et al., DDPM (2020) · Salimans and Ho, Progressive Distillation (2022)

One DDPM training iteration

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

Ho et al., DDPM, Algorithm 1.

One supervised pair

Draw y_0, j, ϵ.

Construct the noisy input y_j.

Predict ϵ_θ, compare with ϵ, and update θ.

In the original algorithm, x_t corresponds to our y_j.

This algorithm uses the simplified noise MSE: a reweighting of the ELBO-derived terms.

A training iteration samples one noise level.

The DDPM sampling loop

Algorithm 2 Sampling

$\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for $t = T, \dots, 1$ **do**
  $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
  $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
end for
return $\mathbf{x}_0$

Ho et al., DDPM, Algorithm 2.

Repeated generation steps

Start with Gaussian noise y_J.

Evaluate the same network for $j = J, \dots, 1$.

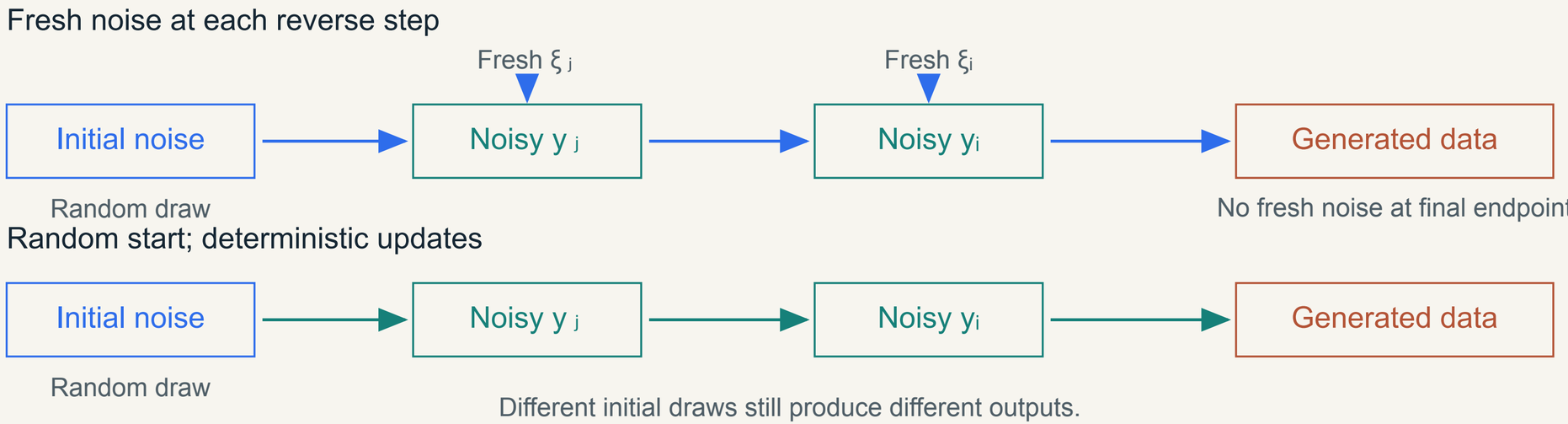
Sample each reverse transition and return y_0.

$$y_{j-1} = \mu_\theta(y_j, j) + \sigma_j \xi_j$$

Use fresh $\xi_j \sim \mathcal{N}(0, I)$ except at the final step.

Sampling starts with noise and repeatedly applies the learned reverse transitions.

Do reverse steps need fresh noise?

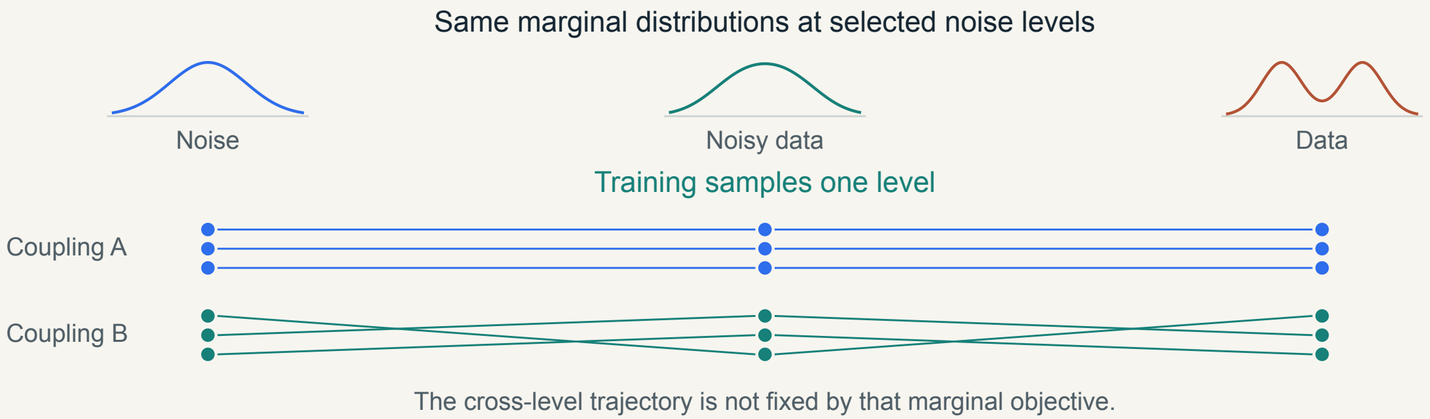


$$y_{j-1} = \mu_\theta(y_j, j) + \sigma_j \xi_j \qquad \text{(DDPM)}$$

Is fresh randomness at every step necessary to generate a distribution?

DDIM asks whether randomness can come from the initial sample alone.

The denoising task does not fix a unique path



$$q(y_j \mid y_0) = \mathcal{N}\left(\sqrt{\bar{\alpha}_j}\, y_0,\ (1 - \bar{\alpha}_j) I\right)$$

Training constructs one noisy sample at a chosen level. It does not run a whole generation path.

DDIM preserves these noisy marginals while changing how levels are connected.

Estimate the clean and noise components

$$y_j = \sqrt{\bar{\alpha}_j}\, y_0 + \sqrt{1 - \bar{\alpha}_j}\, \epsilon \qquad \text{(training construction)}$$

$$\hat{\epsilon} = \epsilon_\theta(y_j, j), \qquad \hat{y}_0 = \frac{y_j - \sqrt{1 - \bar{\alpha}_j}\, \hat{\epsilon}}{\sqrt{\bar{\alpha}_j}}$$

Estimated clean component

$\hat{y}_0$ describes the data signal.

Estimated noise component

$\hat{\epsilon}$ describes the residual noise.

Use the current noisy input to estimate both components—not a known clean image.

Move to a lower noise level without fresh noise

$$y_j = \sqrt{\bar{\alpha}_j}\,\hat{y}_0 + \sqrt{1-\bar{\alpha}_j}\,\hat{\epsilon}$$

Keep the estimates. Change the signal–noise proportions.

$$y_i = \sqrt{\bar{\alpha}_i}\,\hat{y}_0 + \sqrt{1-\bar{\alpha}_i}\,\hat{\epsilon}, \qquad i < j$$

Clean component

Scale $\hat{y}_0$ using the signal coefficient at level i.

Noise component

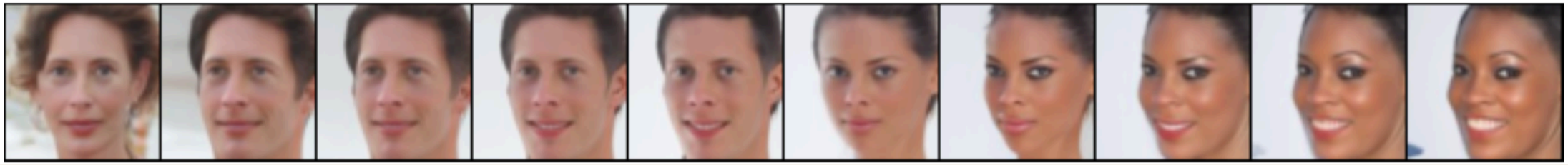
Reuse the predicted noise with the noise coefficient at level i.

This is deterministic DDIM ($\eta = 0$). Re-evaluate the network after moving.

Song et al., DDIM (2020)

Deterministic updates can still generate diverse images

$$y_J \sim \mathcal{N}(0, I) \quad \longrightarrow \quad y_0 = F_\theta(y_J)$$



Original DDIM: interpolate the initial noise, then generate a finished image from each starting point.

Same starting noise

The same deterministic sampler follows the same path.

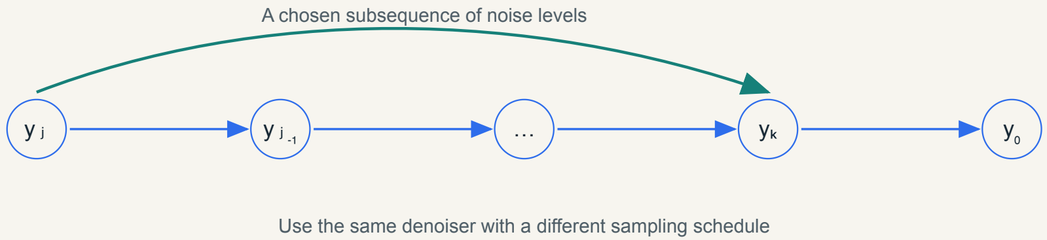
Different starting noise

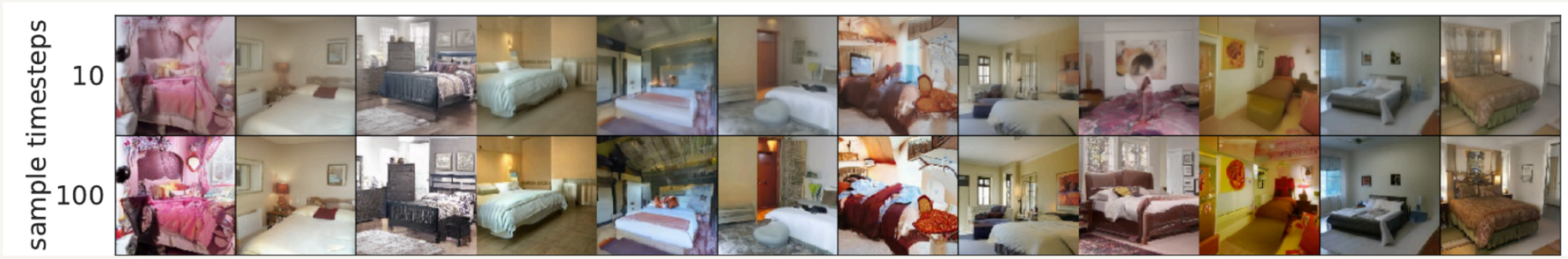
Different starts can produce different images.

A random input—not a random step—is sufficient for output diversity.

Song et al., DDIM (2020)

An additional benefit: fewer sampling steps





Original DDIM samples: each column uses the same initial noise across different sampling step counts.

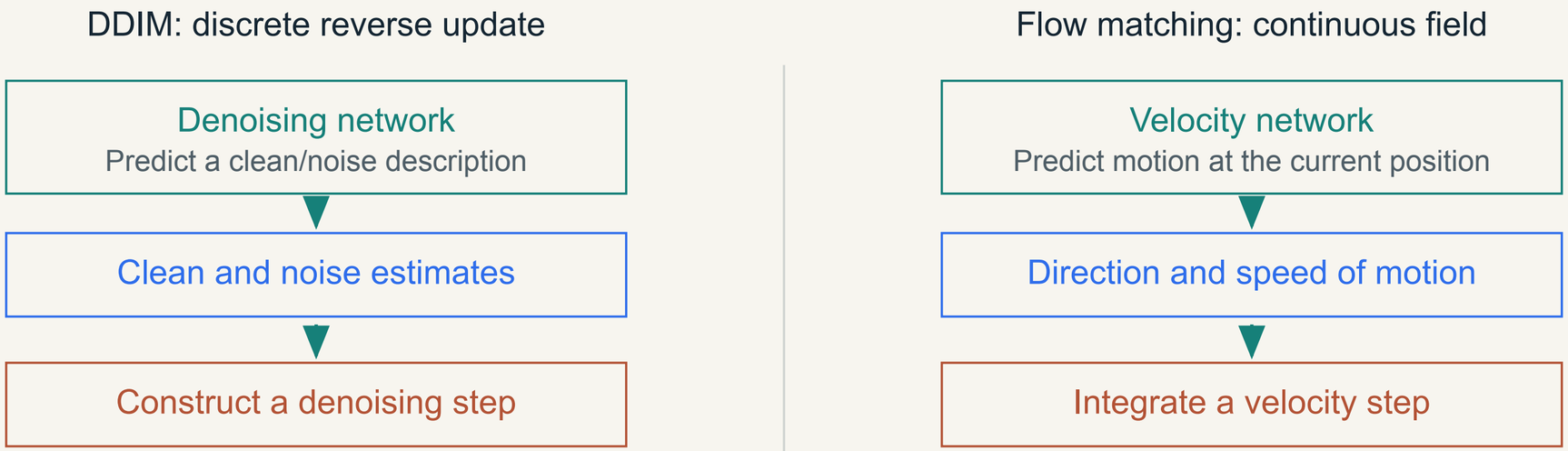
$$J = j_K > j_{K-1} > \cdots > j_0 = 0$$

Evaluate the network at the selected levels. Fewer evaluations reduce sampling cost and can change sample quality.

Choose a shorter level sequence without retraining the denoiser.

Can we learn the motion directly?

DDIM showed that a generated sample can follow a deterministic path.



Two related views of movement during generation

Move from predicting a denoised endpoint to learning how the sample moves.

Flow matching: a velocity field



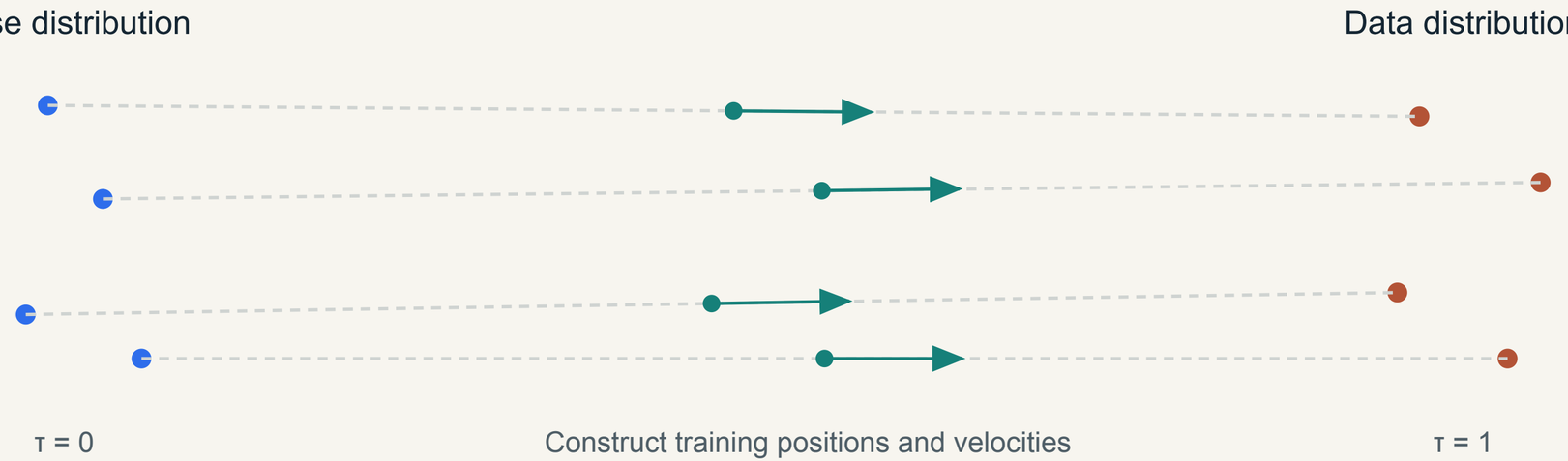
Integrate the field from the noise distribution toward the data distribution

$$\frac{dr_\tau}{d\tau} = v_\theta(r_\tau, \tau)$$

The network predicts how the current sample should move.

$\tau = 0$ is noise. $\tau = 1$ is data.

A path from noise to data



$$z \sim \mathcal{N}(0, I), \qquad y \sim p_{\text{data}}, \qquad r_\tau = (1-\tau)z + \tau y$$

Draw the noise and data independently. Interpolate them at a sampled $\tau \in [0, 1]$.

The path defines a training input at every generation time.

The velocity of a paired path

$$r_\tau = (1-\tau)z + \tau y$$

$$\frac{dr_\tau}{d\tau} = y - z$$

Position

Interpolate the sampled noise–data pair.

Velocity

The endpoint displacement gives the direction and speed.

The target velocity is known for each constructed training pair.

Learning the velocity field

$$\mathcal{L}_{\mathrm{CFM}} = \mathbb{E}_{z,y,\tau}\left[\left\|v_\theta(r_\tau, \tau) - (y - z)\right\|^2\right]$$

Input

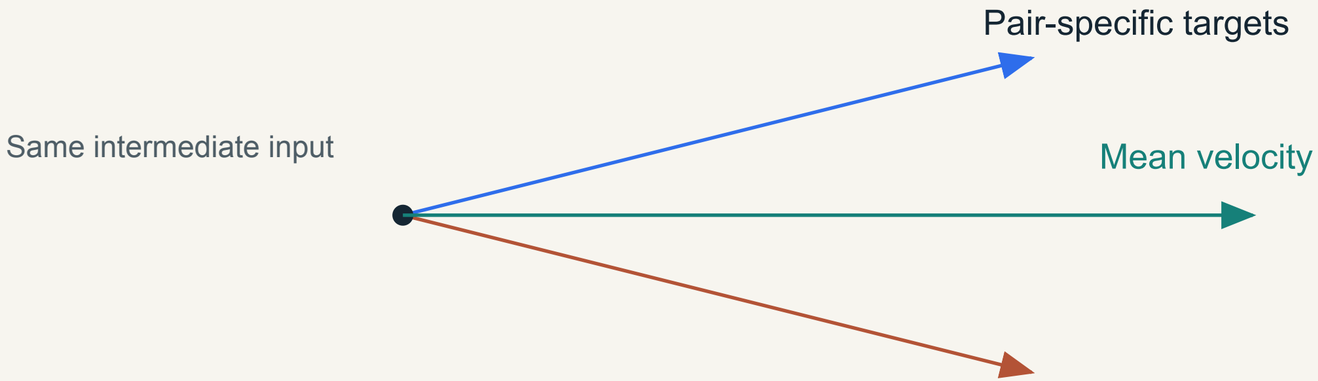
Interpolated sample r_τ and generation time τ.

Target

The paired path velocity $y - z$.

Training constructs intermediate samples directly, without integrating the learned field.

Training paths and sampling trajectories



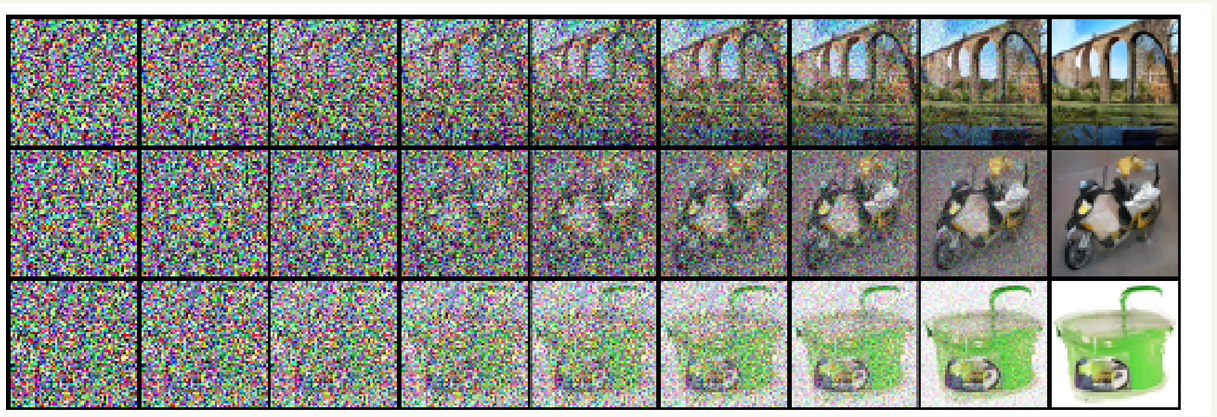
Regress different pair targets at the same position and generation time

$$v^*(r,\tau) = \mathbb{E}[\, y - z \mid r_\tau = r, \tau \,]$$

Many noise–data pairs can produce the same intermediate input.

Straight training segments can produce curved sampling trajectories.

Sampling with the velocity field



Original flow matching sample trajectories, from noise to generated images (Lipman et al., Figure 6).

$$r_0 \sim \mathcal{N}(0, I), \qquad r_{\tau+\Delta\tau} \approx r_\tau + \Delta\tau\, v_\theta(r_\tau, \tau)$$

Euler sampling: predict the current velocity, take a small step, then evaluate again.

Follow the learned field from $\tau = 0$ to $\tau = 1$.

Denoising and velocity prediction

DDPM / DDIM

$$y_j = a_j y_0 + b_j \epsilon$$

Predict noise along a prescribed Gaussian corruption schedule.

The sampler determines the reverse updates.

Flow matching

$$r_\tau = (1-\tau)z + \tau y$$

Predict velocity along a chosen noise–data path.

The sampler integrates the learned field.

Path, prediction target, and sampling rule are connected design choices.

Next

How do diffusion and flow matching fit into the world-model framework?