

CIS 6280 · FALL 2026

WORLD MODELS

LECTURE 10

Policy and Value Learning with World Models

L6–L9 revisit · From learned predictions to decisions

Course announcements

SEP 30

Assignment 2 releases September 30.

AFTER FALL BREAK

Final project details will be released next week, after fall break.

BEFORE OCT 14

Complete project registration before October 14. Groups of 2–4 are encouraged.

TODAY'S LECTURE

Today's revisit supports Assignment 2. Part of the planned L10 material moves to after fall break.

Flappy Bird: plan, act, and replan

Learned latent world model

Predict candidate futures for flap / no-flap sequences.

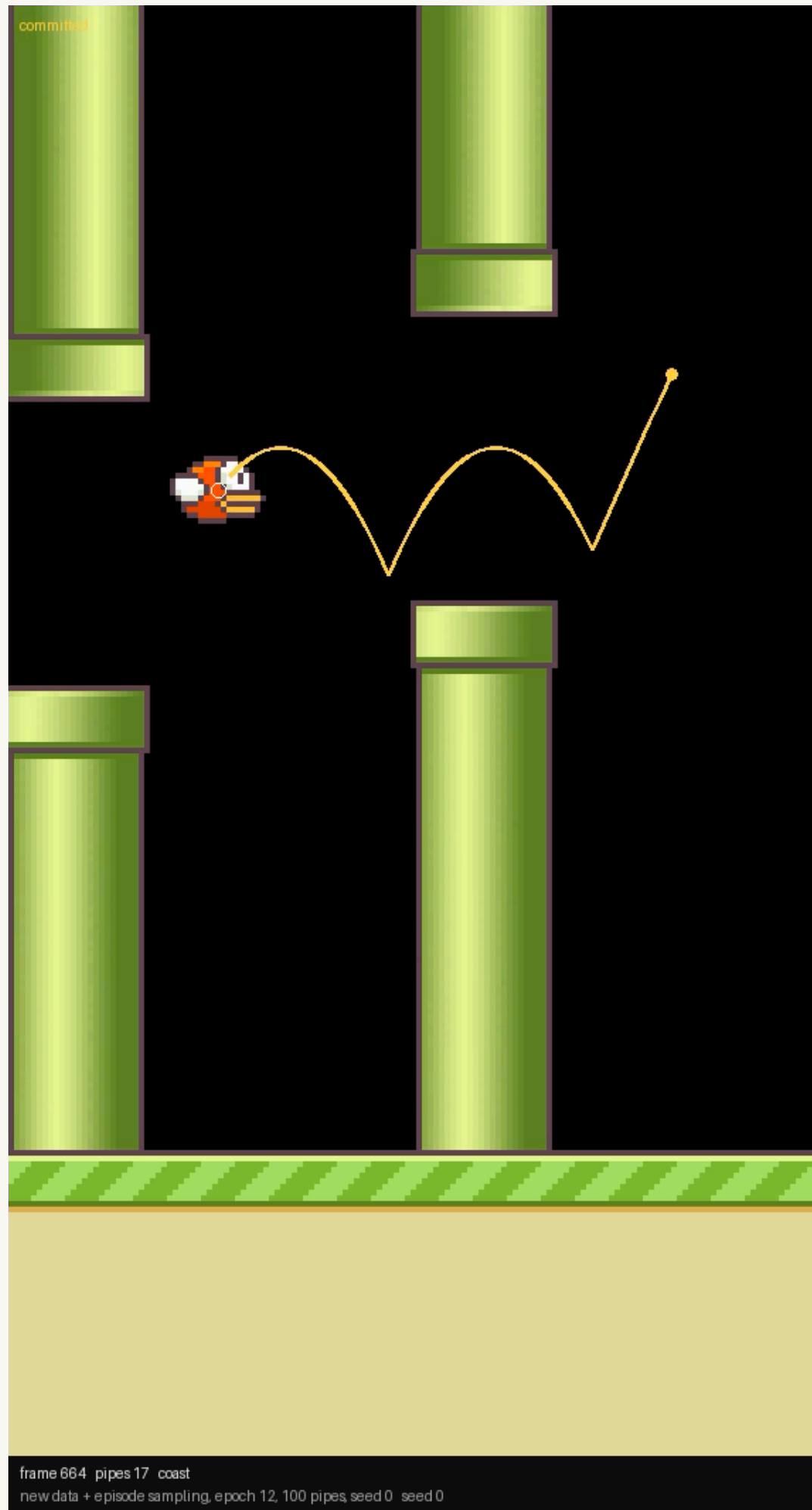
CEM selects the best predicted path

Execute only the first action of the chosen sequence.

Observe the new state and replan

New pipes keep arriving. Repeat the decision process.

Finite planning horizon. Repeated control.



Two ways to act with a world model



PushT: move the block to the goal.

First: plan with CEM

Search action sequences and compare their predicted outcomes.

Then: learn policy and value

Policy: choose actions from the current state.

Value: estimate future return under a policy.

Model-based RL can use either approach, or combine them.

Flappy Bird executes one action, then replans. This PushT clip executes the full plan.

The four interfaces

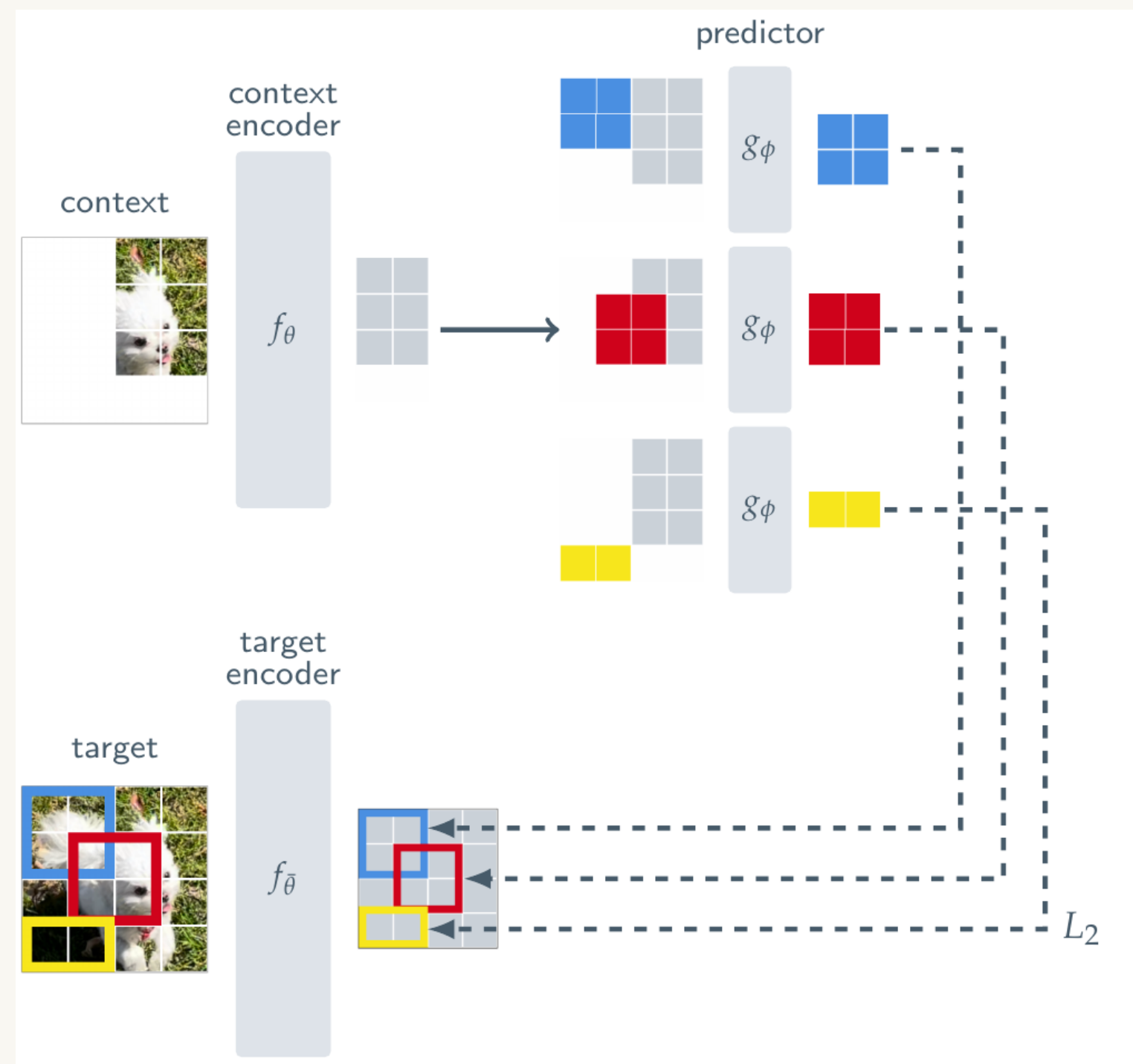
Component	Input	Output
Representation	Image or observation history	JEPA features / VAE posterior
Dynamics	Latent context and actions	Predicted future states
Objective	Predicted outcome and task	Cost or return
Action selection	Model and objective	Action plan / policy action

Three families of representation learning

Family	What supplies the target?	Examples we covered
Reconstruction	Observed data or missing pixels	Autoencoders / MAE; VAE adds a probabilistic latent model
Contrastive	The matching view among candidates	SimCLR / CLIP
Learned targets	Another network encoding of a view or region	DINO / I-JEPA / V-JEPA / LeJEPA

Today: learned targets for LeWM, and reconstruction for the RSSM route.

JEPA predicts a representation



The context encoder sees visible patches.

The predictor estimates target-block features from context and target positions.

The I-JEPA loss uses teacher features

Loss. The loss is simply the average L_2 distance between the predicted patch-level representations $\hat{\mathbf{s}}_y(i)$ and the target patch-level representation $\mathbf{s}_y(i)$; i.e.,

$$\frac{1}{M} \sum_{i=1}^M D(\hat{\mathbf{s}}_y(i), \mathbf{s}_y(i)) = \frac{1}{M} \sum_{i=1}^M \sum_{j \in B_i} \|\hat{\mathbf{s}}_{y_j} - \mathbf{s}_{y_j}\|_2^2.$$

Prediction: contextual features at the masked target locations.

Target: normalized features from a full-image teacher, with stopped gradients.

Stop-gradient and EMA have different roles

$$\theta_{\text{teacher}}^{\text{new}} = m\theta_{\text{teacher}}^{\text{old}} + (1 - m)\theta_{\text{student}}^{\text{new}}$$

$$m = 0.9, \quad 0.9 \times 2 + 0.1 \times 4 = 2.2$$

Backpropagation updates the student and predictor.

EMA then updates teacher parameters using the new student parameters.

Feature agreement alone permits collapse

$$E(x) = c \quad \text{for every input}$$

$$F(c) = c \quad \implies \quad \mathcal{L}_{\text{pred}} = 0$$

Identical embeddings match perfectly while discarding input information.

A useful training recipe must also preserve variation across examples.

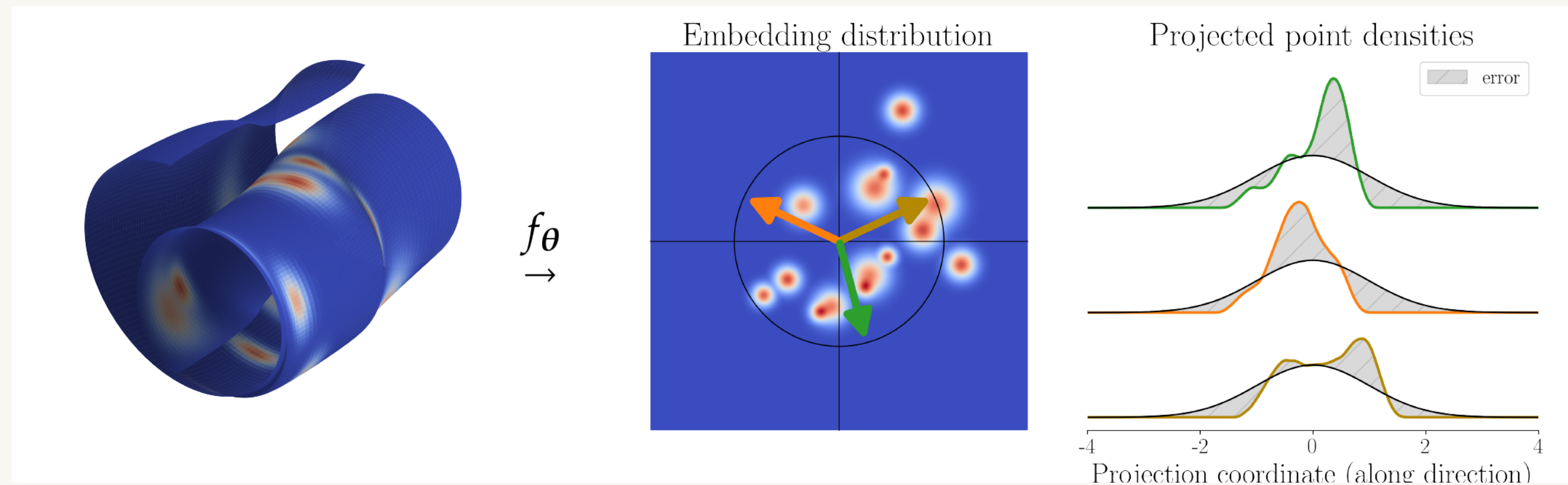
LeJEPA adds a distribution constraint

$$\mathcal{L} = (1 - \lambda) \mathcal{L}_{\text{agreement}} + \lambda \mathcal{L}_{\text{SIGReg}}$$

View agreement keeps corresponding examples close.

SIGReg encourages a prescribed distribution across examples.

SlGReg constrains embeddings across examples



Collect a batch Z of encoded views.

Encourage its distribution to resemble $N(0, I)$.

Random projections reduce the matching problem

$$\mathbf{h}^{(m)} \triangleq \mathbf{Z}\mathbf{u}^{(m)}, \mathbf{u}^{(m)} \in \mathbb{S}^{D-1}, \quad (6)$$

Sample unit directions $\mathbf{u}$ and compute $\mathbf{h} = \mathbf{Z}\mathbf{u}$.

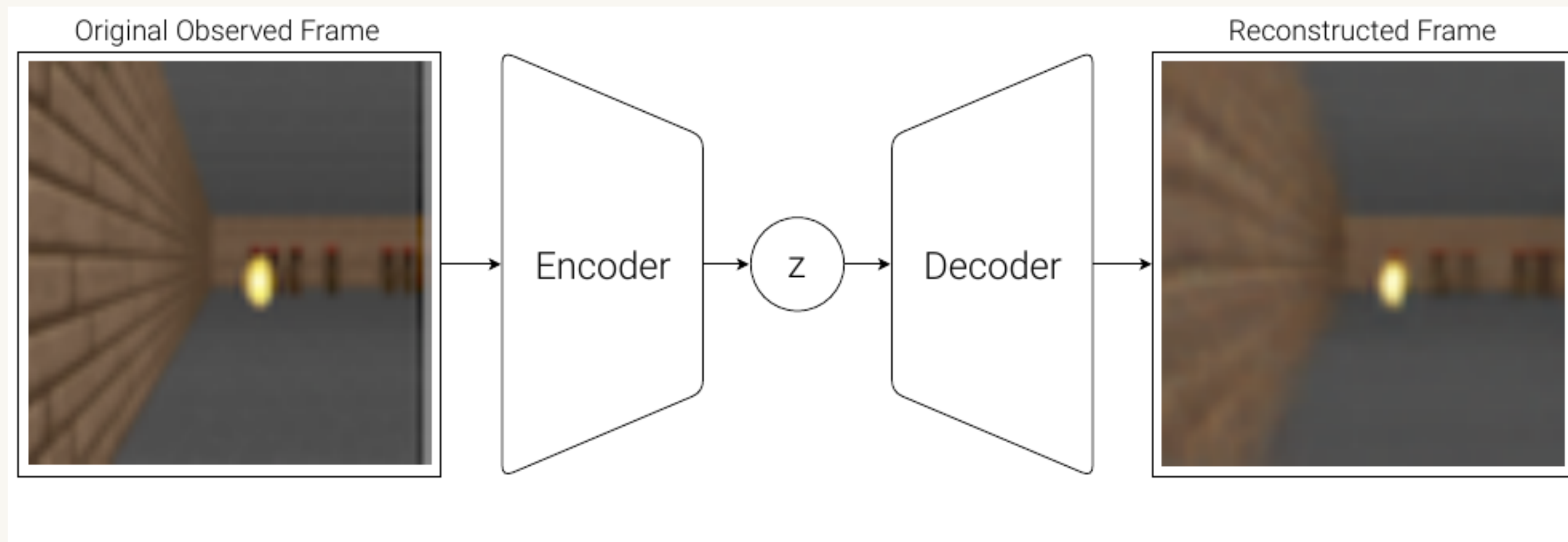
Each direction produces B scalar values to compare with $N(0,1)$.

JEPA and VAE learn latents for different targets

Question	JEPA / LeJEPA	VAE
What must the code preserve?	Information needed to predict target features	Information needed to reconstruct the image
What does one image produce?	A feature vector $z = E(o)$	A distribution $q(z \mid o)$
Where is the Gaussian?	SIGReg: embeddings across different examples	A posterior for each image, compared with a shared prior
Why revisit it today?	Prepares us for LeWM feature prediction	Prepares us for RSSM inference and its training loss

Two representation routes. Next: VAE, to prepare for RSSM.

VAE learns to reconstruct an image from a latent



For one fixed image, the encoder outputs a mean and diagonal variance.

Sample z from $q(z | o)$, then reconstruct that image with the decoder.

Sampling still permits encoder gradients

$$z = \mu(o) + \sigma(o) \odot \epsilon, \quad \epsilon \sim \mathcal{N}(0, I)$$

ϵ supplies randomness independently of encoder parameters.

The decoder loss differentiates through μ and σ via the sampled z .

The VAE loss has two jobs

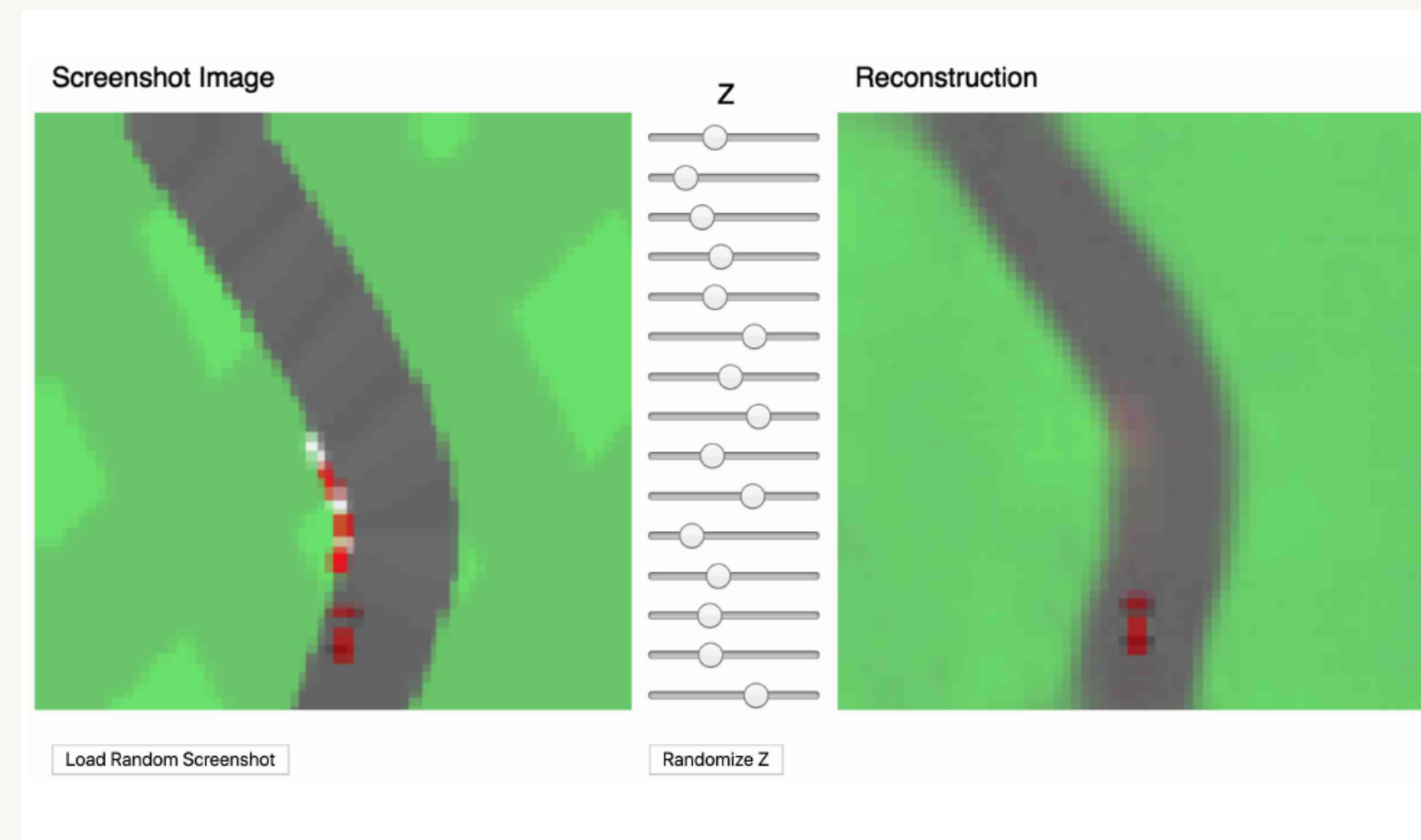
$$\mathcal{L}_{\text{VAE}} = \mathbb{E}_q[-\log p(o \mid z)] + D_{\text{KL}}(q(z \mid o) \parallel p(z))$$

$$p(z) = \mathcal{N}(0, I)$$

Reconstruction rewards information about the image.

KL limits how far each image posterior departs from the prior.

Why reconstruct the observed image?



For a fixed-variance Gaussian decoder, negative log likelihood is proportional to squared error.

The target is the same image that entered the encoder.

Why the VAE needs a KL term

$$D_{\text{KL}}(q || \mathcal{N}(0, I)) = \frac{1}{2} \sum_j [\mu_j^2 + \sigma_j^2 - \log \sigma_j^2 - 1]$$

The penalty is zero when $\mu = 0$ and $\sigma^2 = 1$.

It balances reconstruction against an image-dependent code.

VAE KL compares distributions for each observation.

Two representation losses, still no action dynamics

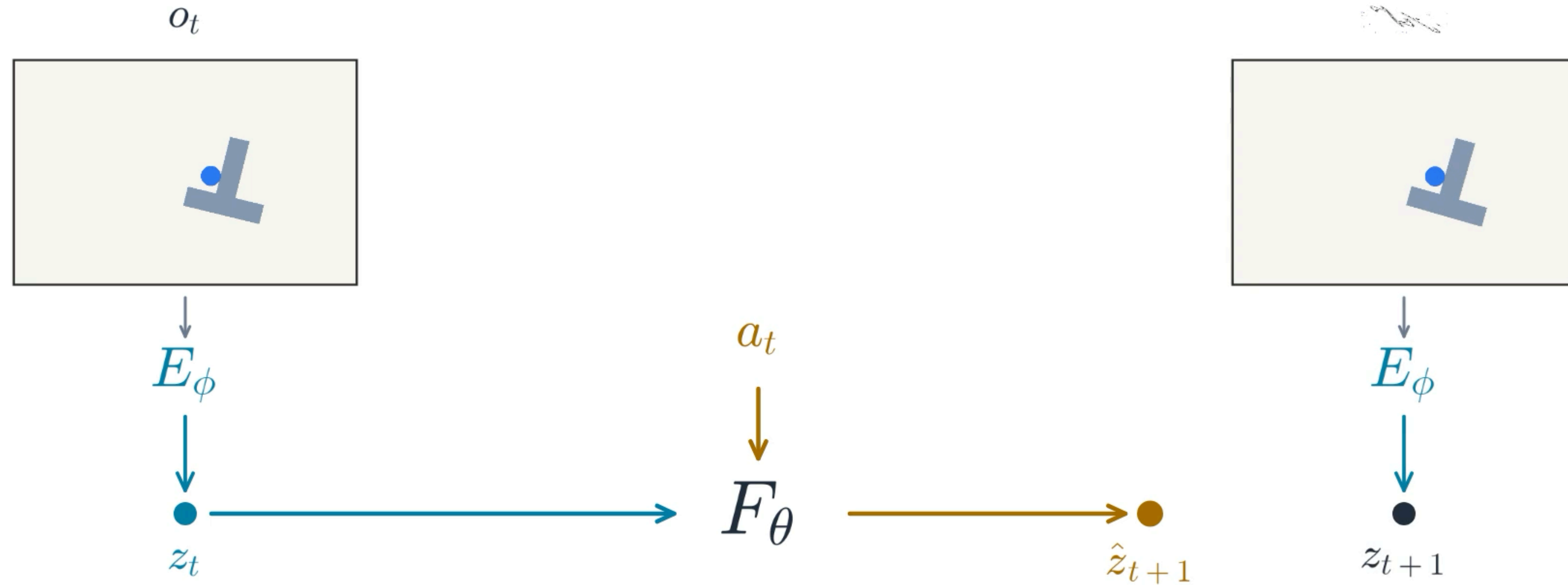
	JEPA / LeJEPA	VAE
Compared target	Learned feature of another view / region	Observed input under a decoder likelihood
Latent output	Usually a deterministic embedding	A conditional distribution $q(z \mid o)$
Distribution constraint	Recipe-dependent, e.g. aggregate SIGReg	Per-observation KL to $p(z)$
Main interpretation	Learn useful predictable features	Learn inference and a generative latent model

Next, condition future-state prediction on actions.

Training and planning optimize different variables

	Model training	Planning at time t
Inputs	Recorded images and actions	Current context and goal image
Compared quantities	Predicted and observed next embeddings	Predicted outcome and desired goal
Updated variables	Encoder and predictor parameters	Action-proposal mean and variance
Real environment	Data already collected	Advances only when actions execute

Predict the next latent from this state and action.



$$\hat{z}_{t+1} = F_\theta(z_t, a_t)$$

A1 physics · illustrative latents

Predict future features from observed history and actions; train encoder and predictor jointly.

The tensors in one training batch

Tensor	Shape	Meaning
obs	$B \times T \times 3 \times 224 \times 224$	Recorded images
actions	$B \times T \times A_{\text{block}}$	Recorded action blocks
emb	$B \times T \times D$	Encoded observed images
pred / target	$B \times (T-1) \times D$	Aligned next-step pairs

Actions condition a causal predictor

The predictor is a transformer with 6 layers, 16 attention heads, and 10% dropout ($\sim 10\text{M}$ parameters). Actions are incorporated into the predictor through Adaptive Layer Normalization (AdaLN) [37] applied at each layer. The AdaLN parameters are initialized to zero to stabilize training and ensure that action conditioning impacts the predictor training progressively. The predictor takes as input a

history of N frame representations and predicts the next frame representation auto-regressively with temporal causal masking to avoid looking at future embeddings. The predictor is also followed by a projector network with the same implementation as the one used for the encoder. All components of our world model are learned jointly using the loss described in the following paragraph.

AdaLN introduces the recorded action into the transformer.

The history window must exclude future observations.

Time alignment for four observed frames

Predictor output	Allowed observed context	Recorded input	Target
pred[:,0]	z0	u0	z1
pred[:,1]	z0, z1	u0, u1	z2
pred[:,2]	z0, z1, z2	u0, u1, u2	z3
pred[:,3]	Observed history through z3	History through u3	No z4 in this slice

The prediction loss compares next embeddings

$$\mathcal{L}_{\text{pred}} \triangleq \|\hat{\mathbf{z}}_{t+1} - \mathbf{z}_{t+1}\|_2^2, \quad \hat{\mathbf{z}}_{t+1} = \text{pred}_\phi(\mathbf{z}_t, \mathbf{a}_t). \quad (1)$$

Prediction: the model output under the recorded action.

Target: the encoder output for the real next image.

A prediction MSE by hand

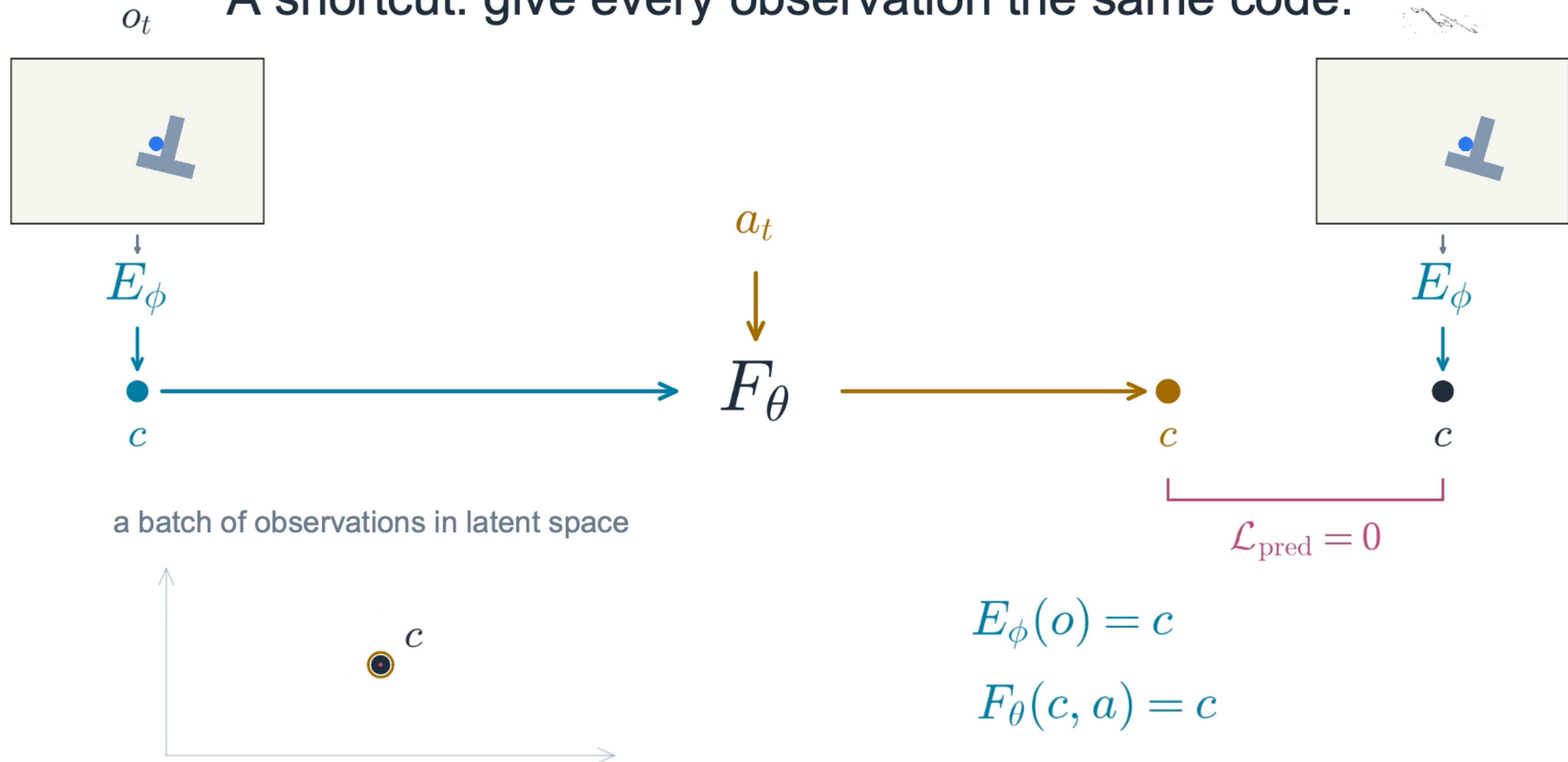
$$\hat{z} = (0.2, -0.1), \quad z = (0.5, 0.3)$$

$$\|\hat{z} - z\|_2^2 = 0.3^2 + 0.4^2 = 0.25$$

$$D = 2 : \quad \text{MSE} = 0.25/2 = 0.125$$

The loss is a scalar that contributes gradients to both branches.

A shortcut: give every observation the same code.



Different images. The same code. Zero prediction error.

A1 physics · illustrative latents

A constant encoder can make prediction error zero while discarding the scene.

Gradients pass through the target encoder

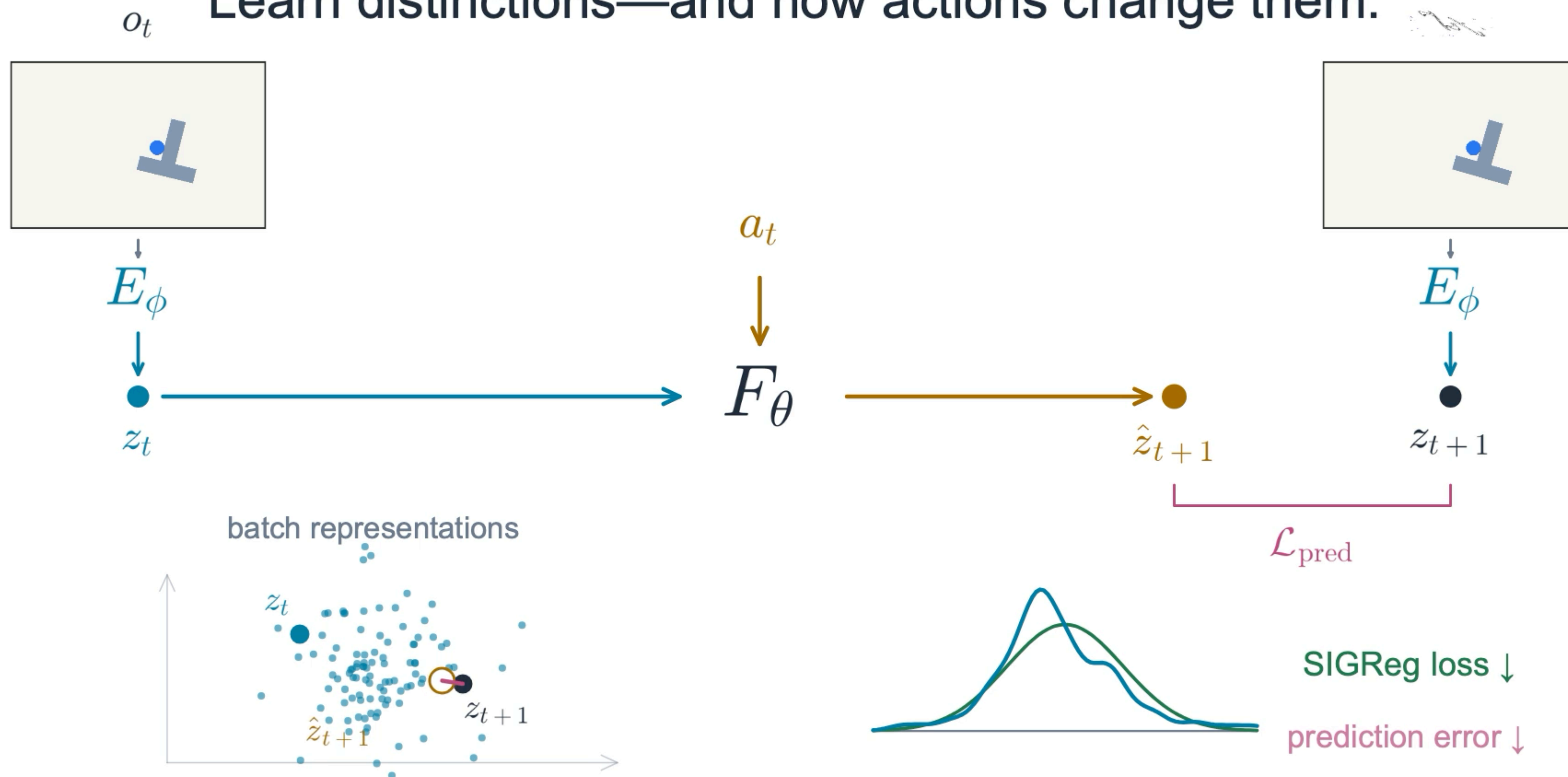
$$\mathcal{L} = \|F_{\phi}(E_{\theta}(o), u) - E_{\theta}(o_{next})\|_2^2$$

$\nabla_{\theta}\mathcal{L}$: prediction path + target path

The recorded action and raw images stay fixed.

LeWM updates θ and ϕ jointly, with no target detach.

Learn distinctions—and how actions change them.



The full objective rewards both structure and prediction.

A1 physics · illustrative latents

$L = L_{\text{prediction}} + \lambda L_{\text{SIGReg}}$; apply SIGReg across the batch at each time slice.

LeWM learns dynamics in feature space

TRAIN THE WORLD MODEL

Predict the next embedding; regularize embeddings with SIGReg.

VISUALIZE ITS PREDICTIONS

Fit a separate decoder afterward to visualize predicted features as images.

The visualization decoder is separate from the world-model training objective.

PUSHT ABLATION · PAPER TABLE 7

	Push-T (SR ↑)
LeWM w/o decoder loss	96.0 ± 2.83
LeWM with decoder loss	86.0 ± 7.54

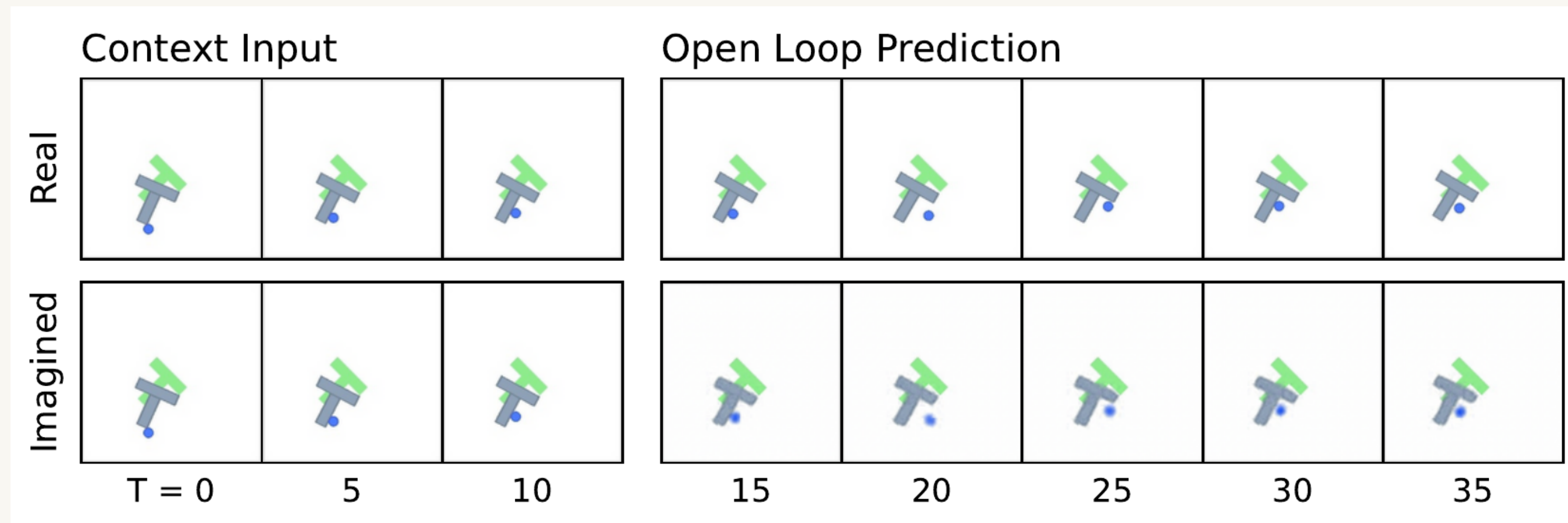
Adding a pixel loss did not improve planning in this PushT ablation.

This is specific to the experiment. RSSM also uses image reconstruction.

A training step with aligned targets

```
emb = encoder(obs)                # B, T, D
pred = predictor(emb, actions)     # causal history
L_pred = mse(pred[:, :-1], emb[:, 1:])
L_reg = mean([SIGReg(emb[:, t]) for t in range(T)])
loss = L_pred + 0.1 * L_reg
optimizer.zero_grad(); loss.backward(); optimizer.step()
```

Teacher forcing and open-loop prediction



Training inputs use observed context at each supervised step.

Planning feeds predicted latents back into later predictions.

Each candidate needs its own rollout history

Stage	Candidate A	Candidate B
Start	Same observed latent history	Same observed latent history
First transition	Use action block $u_{A,0}$	Use action block $u_{B,0}$
Next input	Append predicted $z_{A,1}$	Append predicted $z_{B,1}$
Later transition	Use A history and actions	Use B history and actions

Task rules produce reward labels

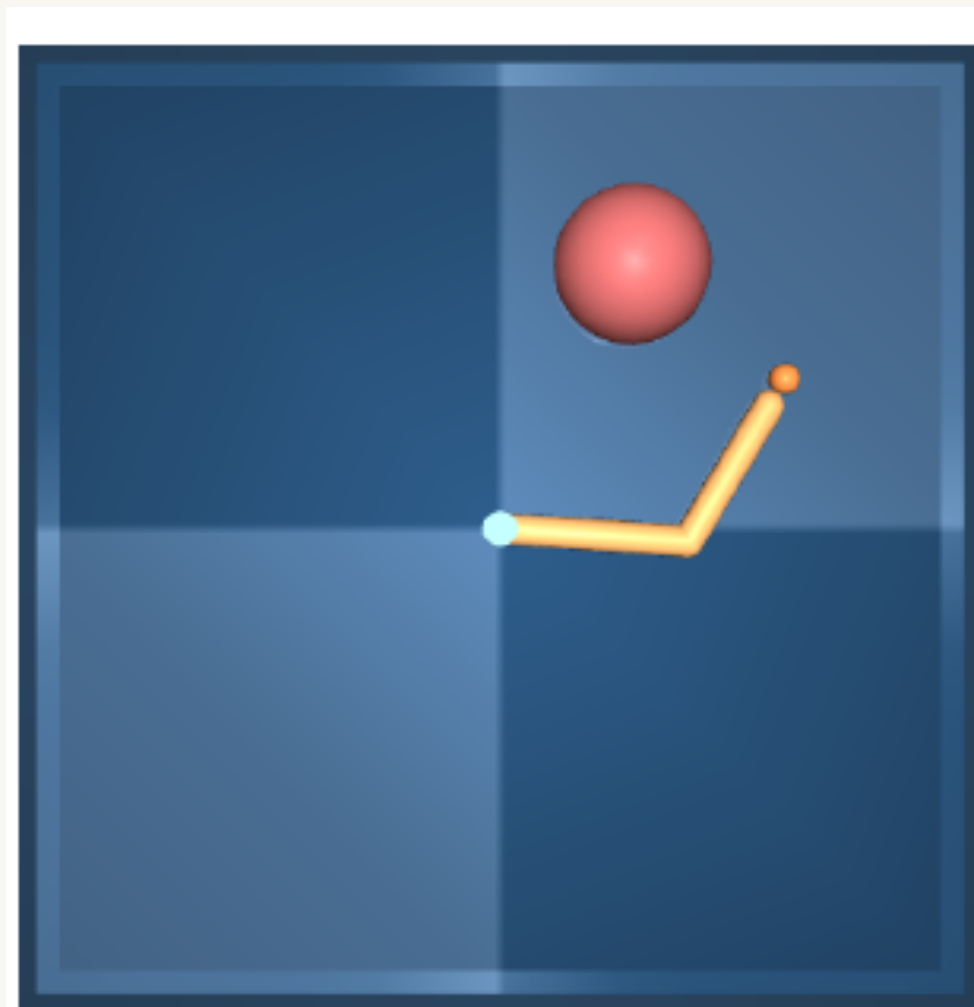
```
13   for action repeat  $k = 1..R$  do  
14   |    $r_t^k, o_{t+1}^k \leftarrow \text{env.step}(a_t)$   
15   |    $r_t, o_{t+1} \leftarrow \sum_{k=1}^R r_t^k, o_{t+1}^k$   
16    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(o_t, a_t, r_t)_{t=1}^T\}$ 
```

env.step supplies a real reward label together with the next observation.

Imagined states have no env.step call, so PlaNet learns a reward predictor.

With action repeat R, the training label sums R primitive rewards.

Example: a sparse Reacher reward



(b) Reacher

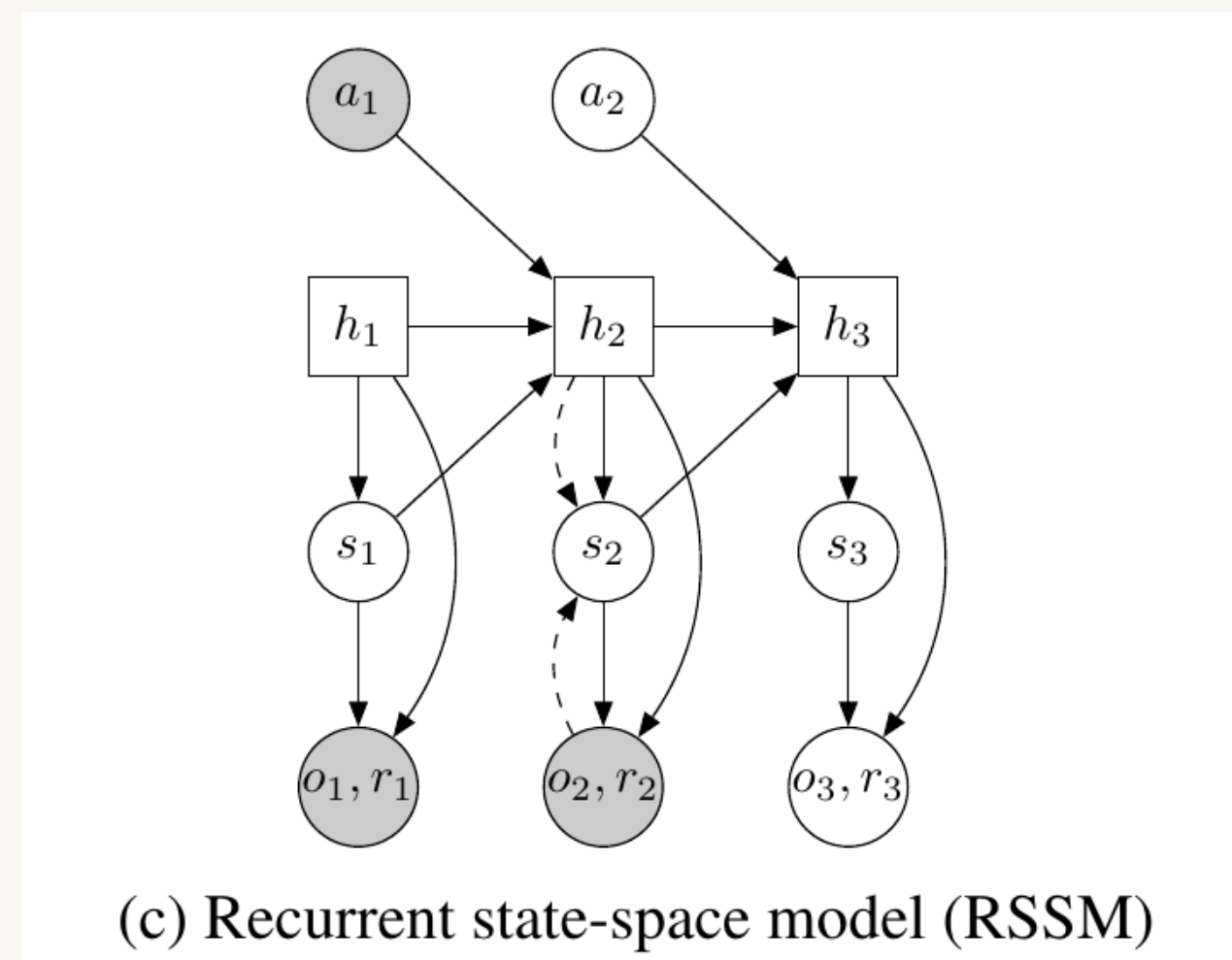
$$r = \mathbf{1}[d \leq R_{goal} + R_{tip}]$$

Suppose $R_{goal} + R_{tip} = 0.06$

At $d = 0.04$, $r = 1$. At $d = 0.08$, $r = 0$.

Both $d = 0.08$ and $d = 0.20$ receive zero: this signal is sparse.

RSSM combines memory with a stochastic state



h_t carries deterministic recurrent memory.

s_t represents latent alternatives conditioned on that memory.

Posterior inference and prior imagination

$$\begin{aligned} \text{Deterministic state model:} \quad & h_t = f(h_{t-1}, s_{t-1}, a_{t-1}) \\ \text{Stochastic state model:} \quad & s_t \sim p(s_t \mid h_t) \\ \text{Observation model:} \quad & o_t \sim p(o_t \mid h_t, s_t) \\ \text{Reward model:} \quad & r_t \sim p(r_t \mid h_t, s_t), \end{aligned} \tag{4}$$

With an image: sample s_t from $q(s_t \mid h_t, o_t)$.

Without the future image: sample s_t from $p(s_t \mid h_t)$.

RSSM learns a reward predictor

the observation model is Gaussian with mean parameterized by a deconvolutional neural network and identity covariance, and the reward model is a scalar Gaussian with mean parameterized by a feed-forward neural network and unit variance. Note that the log-likelihood under a Gaussian distribution with unit variance equals the mean squared error up to a constant.

$$\hat{r}_t = \rho(h_t, s_t), \quad s_t \sim q(s_t \mid h_t, o_t)$$

$$\begin{aligned} \mathcal{L}_{reward} &= \mathbb{E}_q[-\log \mathcal{N}(r_t; \hat{r}_t, 1)] \\ &= \frac{1}{2} \mathbb{E}_q[(r_t - \hat{r}_t)^2] + \text{constant} \end{aligned}$$

Observed $r = 1$ and prediction $= 0.4$ give a squared-error term of 0.18.

This loss fits recorded rewards; it does not maximize predicted reward.

Reward loss shapes the shared latent state

$$\hat{r} = ws = 2 \times 0.2 = 0.4, \quad r = 1$$

$$\frac{\partial \mathcal{L}}{\partial w} = (\hat{r} - r)s = -0.12$$

$$\frac{\partial \mathcal{L}}{\partial s} = (\hat{r} - r)w = -1.2$$

Reward error trains the head and the shared inference/recurrent path.

The observed reward is a label, not an extra input to $q(s_t | h_t, o_t)$.

The KL term trains the transition prior to match these inferred states.

The RSSM loss connects inference and prediction

$$\mathcal{L} = \sum_t \mathbb{E}_q[-\log p(o_t \mid h_t, s_t) - \log p(r_t \mid h_t, s_t)] \\ + \sum_t \mathbb{E}_{history}[D_{\text{KL}}(q(s_t \mid h_t, o_t) \parallel p(s_t \mid h_t))]$$

Observation loss preserves scene information. Reward loss preserves task relevance.

KL aligns the future prior with states inferred from observed data.

Predicted rewards score imagined plans

```

3   for candidate action sequence  $j = 1..J$  do
4        $a_{t:t+H}^{(j)} \sim q(a_{t:t+H})$ 
5        $s_{t:t+H+1}^{(j)} \sim q(s_t \mid o_{1:t}, a_{1:t-1}) \prod_{\tau=t+1}^{t+H+1} p(s_\tau \mid s_{\tau-1}, a_{\tau-1}^{(j)})$ 
6        $R^{(j)} = \sum_{\tau=t+1}^{t+H+1} \mathbb{E}[p(r_\tau \mid s_\tau^{(j)})]$ 

```

$$\hat{G}(U) = \sum_{j=1}^H \gamma^{j-1} \hat{r}_{t+j}$$

$$\hat{r}_{t+j} = \rho(\hat{h}_{t+j}, \hat{s}_{t+j})$$

PlaNet uses $\gamma = 1$: predicted rewards (0.2, 0.8, 1.0) give $G_{\text{hat}} = 2.0$.

Future states come from the prior. CEM can minimize $J(U) = -G_{\text{hat}}(U)$.

LeWM goal cost and PlaNet reward

Quantity	How it is computed	How the agent uses it
Observed reward r_t	Task rule on a real transition	Supervised label in PlaNet replay
Predicted reward $\hat{r}_t$	Reward head on (h_t, s_t)	Score an imagined step
Predicted return $\hat{G}(U)$	Sum predicted step rewards	PlaNet prefers high-return plans
LeWM terminal cost $J(U)$	$\ z_{\hat{}}(t+H) - z_{\text{goal}}\ ^2$	Prefer low cost, using a goal image

LeWM learns without reward labels and receives the goal at planning time.

A goal image defines the planning cost

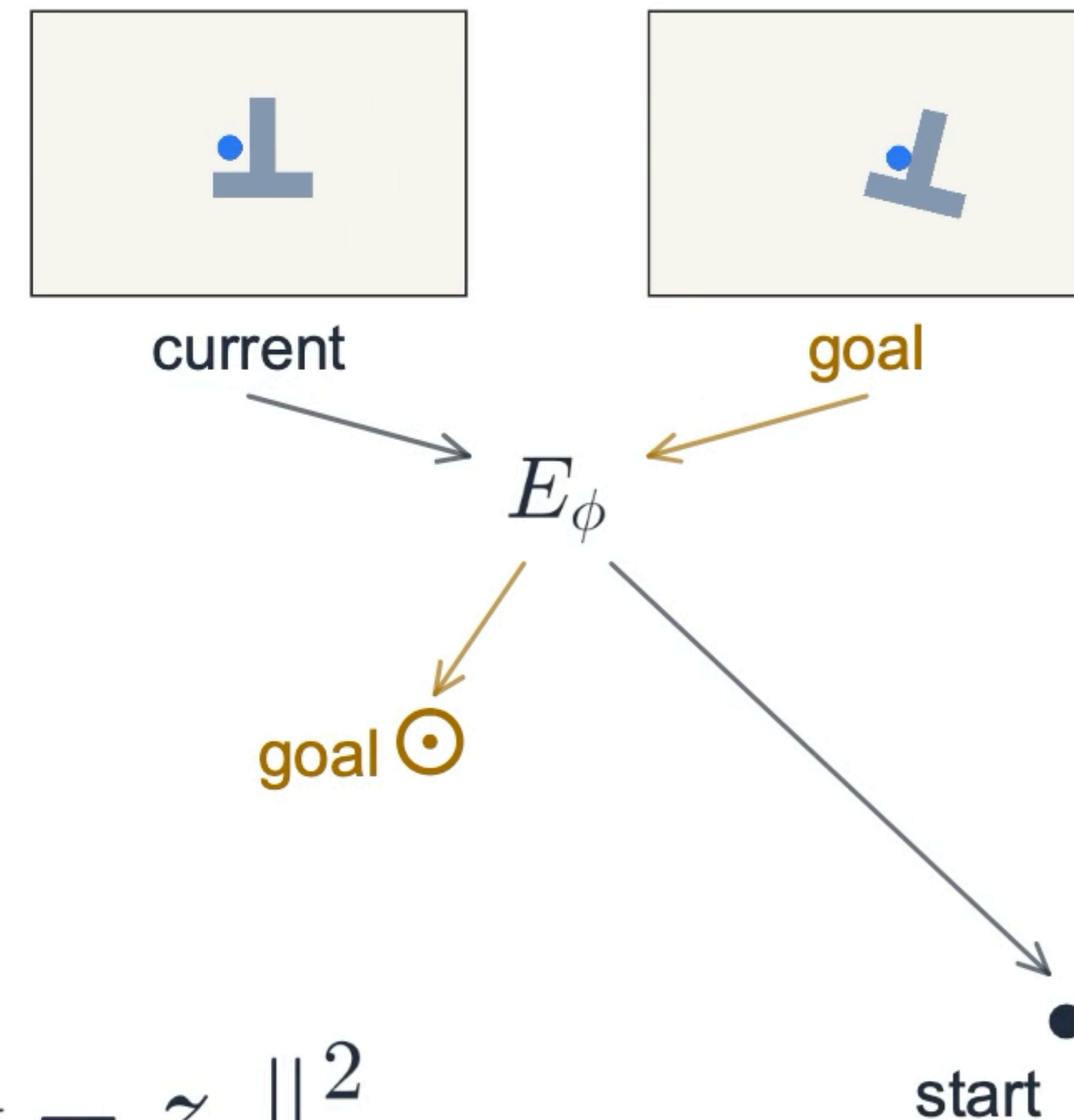
$$\mathcal{C}(\hat{\mathbf{z}}_H) = \|\hat{\mathbf{z}}_H - \mathbf{z}_g\|_2^2, \quad \mathbf{z}_g = \text{enc}_\theta(\mathbf{o}_g), \quad (4)$$

Encode the desired scene once as $\mathbf{z}_{\text{goal}}$.

Compare each predicted terminal embedding with that target.

Which actions would bring us to this goal?

One frozen encoder
Two observations. Two latent states.



$$J(a_0 : H - 1) = \|\hat{z}_H - z_g\|_2^2$$

Compare futures to the goal in the full latent space.

A1 physics · illustrative latents

Score in the full feature space; the fixed PCA projection is only a visualization.

Planning searches the action sequence

$$\mathbf{a}_{1:H}^* = \arg \min_{\mathbf{a}_{1:H}} \mathcal{C}(\hat{\mathbf{z}}_H), \quad (5)$$

$\mathbf{U} = (u_0, \dots, u_{H-1})$ is the optimization variable.

Encoder and predictor parameters remain fixed.

The complete CEM algorithm

Algorithm 2 Cross-Entropy Method (CEM) for Action Sequence Optimization

Require: World model f , planning horizon H , number of samples N , number of elites K , number of iterations T

- 1: Initialize sampling distribution parameters $\mu_0 = \mathbf{0}$, $\Sigma_0 = I$
 - 2: **for** $t = 1$ to T **do**
 - 3: Sample N candidate action sequences $\{a_{1:H}^{(i)}\}_{i=1}^N \sim \mathcal{N}(\mu_{t-1}, \Sigma_{t-1})$
 - 4: **for** $i = 1$ to N **do**
 - 5: Roll out $a_{1:H}^{(i)}$ in the world model f
 - 6: Compute cost $J^{(i)}$
 - 7: **end for**
 - 8: Select the K sequences with lowest cost (elites)
 - 9: Update distribution parameters using elite set:
 - 10: $\mu_t \leftarrow \frac{1}{K} \sum_{i \in \mathcal{E}} a_{1:H}^{(i)}$
 - 11: $\Sigma_t \leftarrow \text{Var}_{i \in \mathcal{E}} \left(a_{1:H}^{(i)} \right)$
 - 12: **end for**
 - 13: **return** best action sequence found or first action of μ_T
-

A population of plans proposes outcomes. Elite plans reshape the next population.

Indices in a planning decision

Symbol	Meaning	What changes
t	Current real decision time	Only after real execution
$h = 0, \dots, H-1$	Future model step inside a plan	Predicted context for a candidate
$k = 0, \dots, K-1$	CEM search round	Proposal mean and variance
$i = 1, \dots, N$	Candidate plan in one round	Actions and predicted trajectory

The proposal is a distribution over whole plans

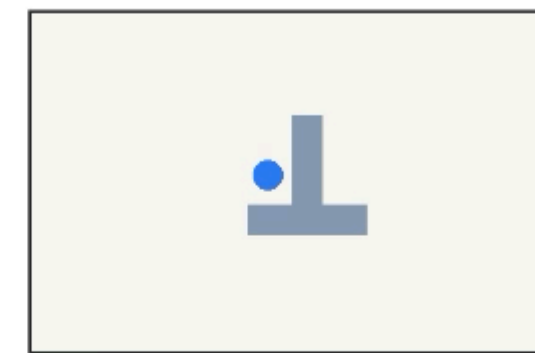
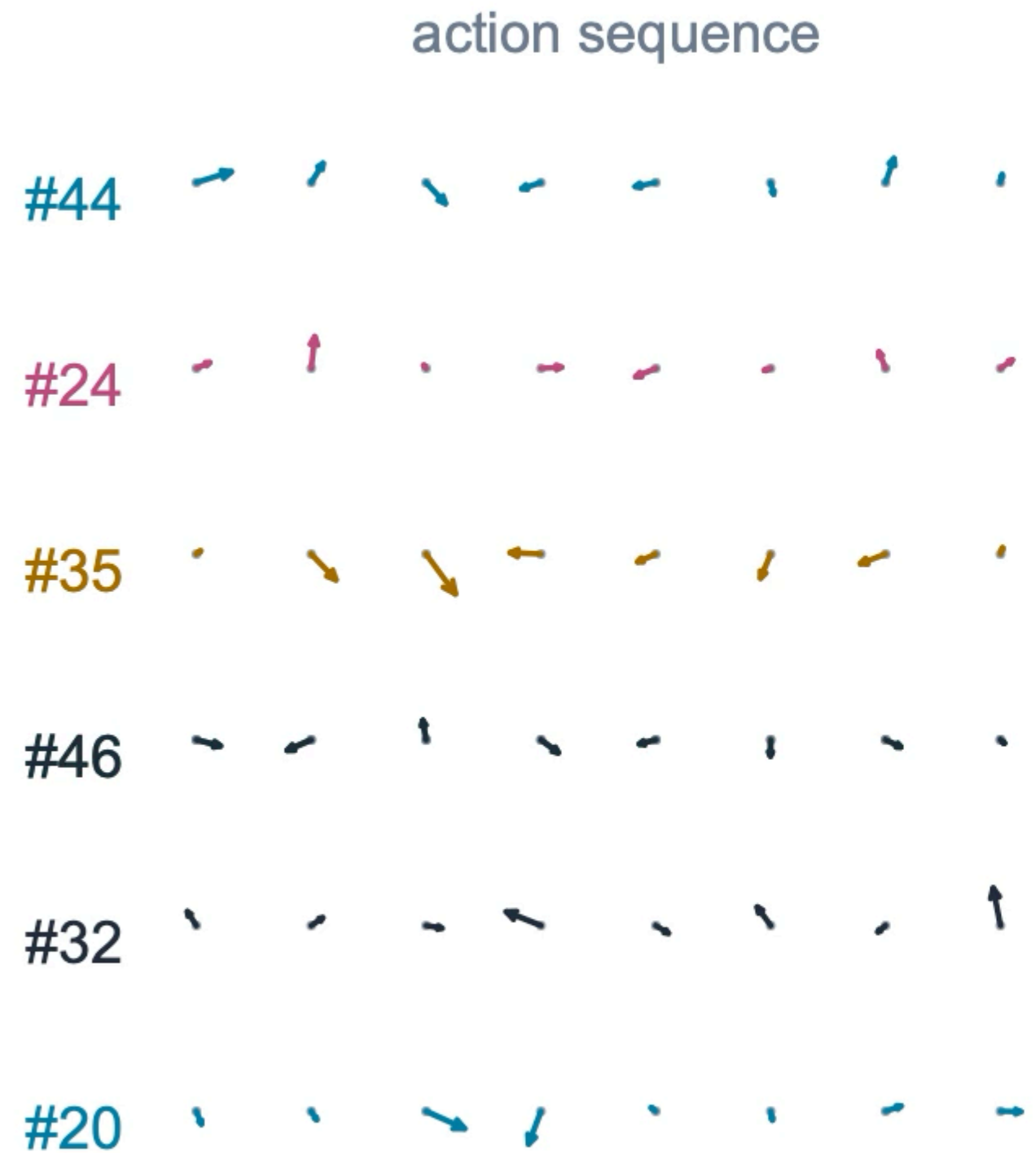
$$U \in \mathbb{R}^{H \times A_{block}}$$

$$q_k(U) = \mathcal{N}(\text{vec}(U); \mu_k, \text{diag}(v_k))$$

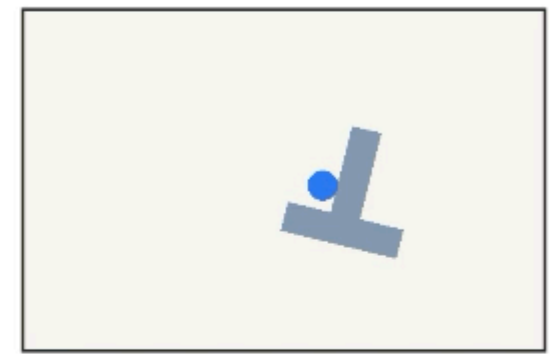
$$\text{sampled plans: } N \times H \times A_{block}$$

Each future time and action coordinate has its own mean and variance.

Sample actions. Then let the model predict.



current

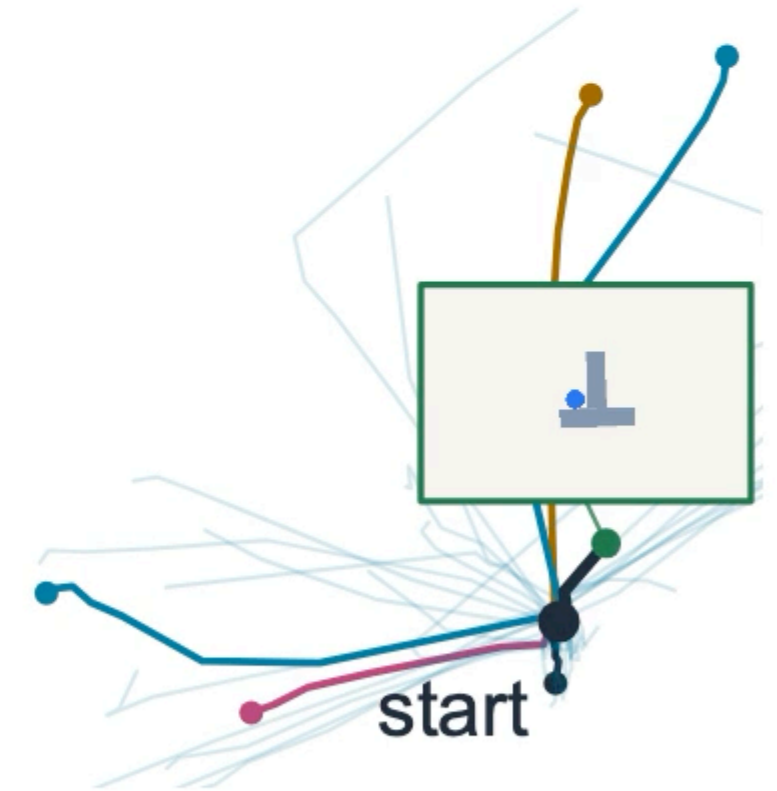


goal

goal $\odot$



same PCA · enlarged view



48 sampled futures · outcomes appear on their trajectories

A1 physics · illustrative latents

Initialize $\mu = 0$, $v = 1$. Sample $U = \mu + \sqrt{v} \odot \epsilon$, then apply action bounds.

Every candidate rolls out the frozen model

Algorithm 2 Cross-Entropy Method (CEM) for Action Sequence Optimization

Require: World model f , planning horizon H , number of samples N , number of elites K , number of iterations T

- 1: Initialize sampling distribution parameters $\mu_0 = \mathbf{0}$, $\Sigma_0 = I$
- 2: **for** $t = 1$ to T **do**
- 3: Sample N candidate action sequences $\{a_{1:H}^{(i)}\}_{i=1}^N \sim \mathcal{N}(\mu_{t-1}, \Sigma_{t-1})$
- 4: **for** $i = 1$ to N **do**
- 5: Roll out $a_{1:H}^{(i)}$ in the world model f
- 6: Compute cost $J^{(i)}$
- 7: **end for**

Copy the same observed context into N independent branches.

Apply each candidate action block and append the predicted embedding.

Scoring compares the predicted terminal outcomes

$$J_i = \sum_d (\hat{z}_{i,t+H,d} - z_{goal,d})^2$$

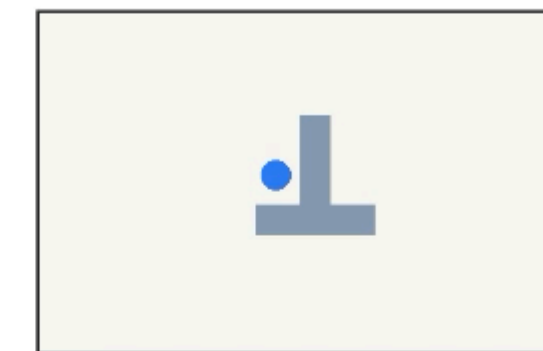
One scalar cost per candidate plan

Lower cost ranks higher in this LeWM example.

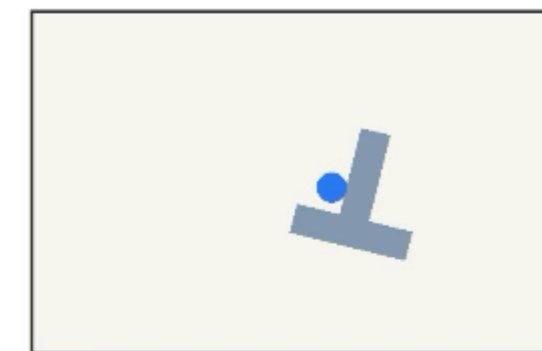
All plans use the same goal and the same starting context.

Keep the six plans with the lowest goal cost.

	action sequence	8D cost
#44		0.01814
#19		0.01826
#23		0.01844
#07		0.02536
#42		0.02642
#28		0.03313



current



goal

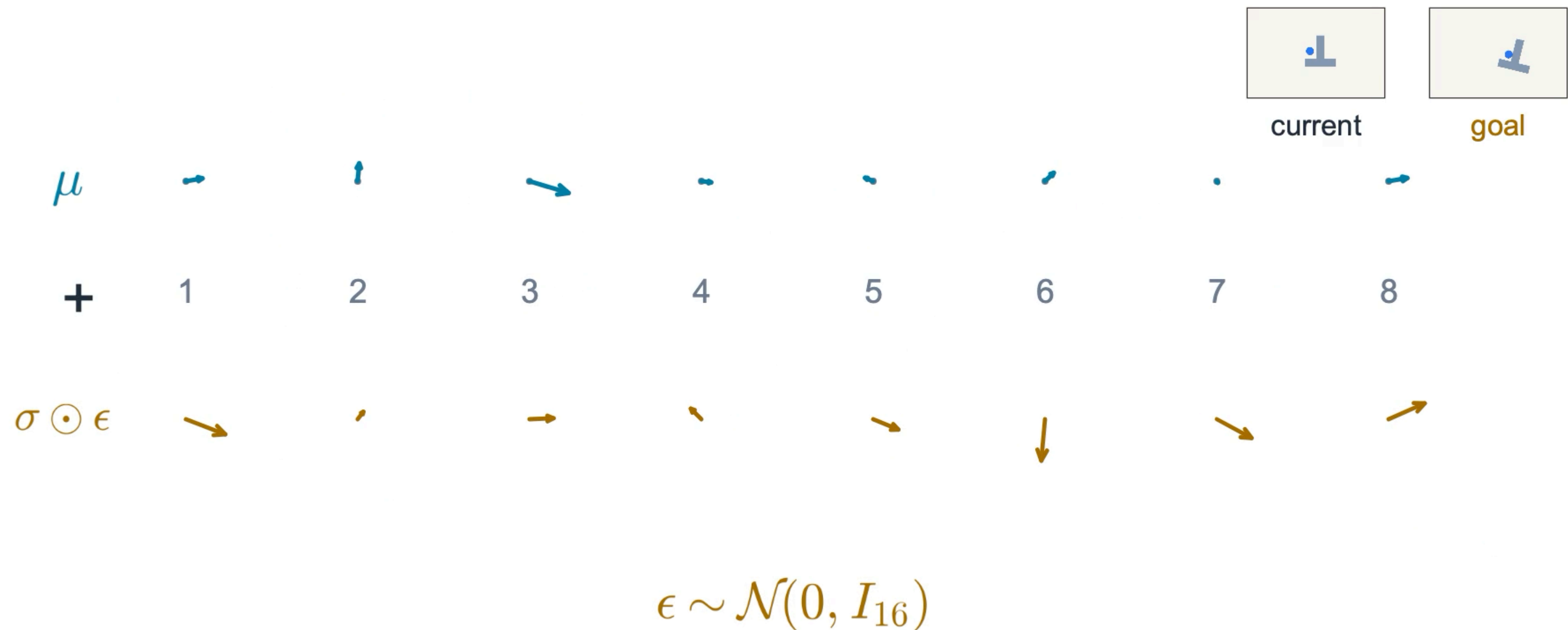


Select by the full latent cost—not by distance in this 2D picture.

A1 physics · illustrative latents

Elite selection ranks full-dimensional terminal costs, not distances in the PCA picture.

Sample a small perturbation around the fitted mean.



The fitted proposal spread sets the size of the perturbation.

A1 physics · illustrative latents

$\mu = \text{mean}(\text{elites})$; $v = \text{mean}((\text{elite} - \mu)^2)$; resample with standard deviation $\sqrt{v}$.

Why means and variances solve the refit

$$\eta_{new} = \arg \max_{\eta} \frac{1}{N_e} \sum_{i \in elite} \log q_{\eta}(U_i)$$

$$\mu_{new} = \text{mean}(U_{elite})$$

$$v_{new} = \text{mean}((U_{elite} - \mu_{new})^2)$$

The refit makes the selected plans more probable under the proposal.

A two-step system we can calculate

$$p_{next} = p + v, \quad v_{next} = v + a$$

$$p_0 = 0, \quad v_0 = 0; \quad p_2^* = 1, \quad v_2^* = 0$$

$$a_0, a_1 \in [-1, 1]$$

Each plan contains two scalar actions.

Four complete candidate rollouts

Plan (a0, a1)	State after a0: (p1,v1)	Terminal (p2,v2)
(0, 0)	(0, 0)	(0, 0)
(1, 1)	(0, 1)	(1, 2)
(1, −1)	(0, 1)	(1, 0)
(0.5, −0.5)	(0, 0.5)	(0.5, 0)

The classroom cost includes effort

$$J(a_0, a_1) = (a_0 - 1)^2 + (a_0 + a_1)^2 + 0.1(a_0^2 + a_1^2)$$

Term 1 penalizes final position error. Term 2 penalizes final velocity.

Term 3 penalizes action effort. This is an added classroom choice.

Cost calculation and elite selection

Plan	Position	Velocity	Effort	Total
(0, 0)	1	0	0	1.00
(1, 1)	0	4	0.20	4.20
(1, -1)	0	0	0.20	0.20
(0.5, -0.5)	0.25	0	0.05	0.30

Elites: (1, -1) and (0.5, -0.5).

Refitting the mean plan

$$\mu_{new} = \frac{1}{2}[(1, -1) + (0.5, -0.5)]$$

$$\mu_{new} = (0.75, -0.75)$$

$$J(\mu_{new}) = 0.0625 + 0 + 0.1125 = 0.175$$

The mean can differ from every sampled candidate.

Refitting the variance

deviations: $(+0.25, -0.25)$, $(-0.25, +0.25)$

$$v_{new} = (0.0625, 0.0625)$$

$$\sigma_{new} = \sqrt{v_{new}} = (0.25, 0.25)$$

The next Gaussian uses this mean and standard deviation.

CEM can concentrate around the wrong answer

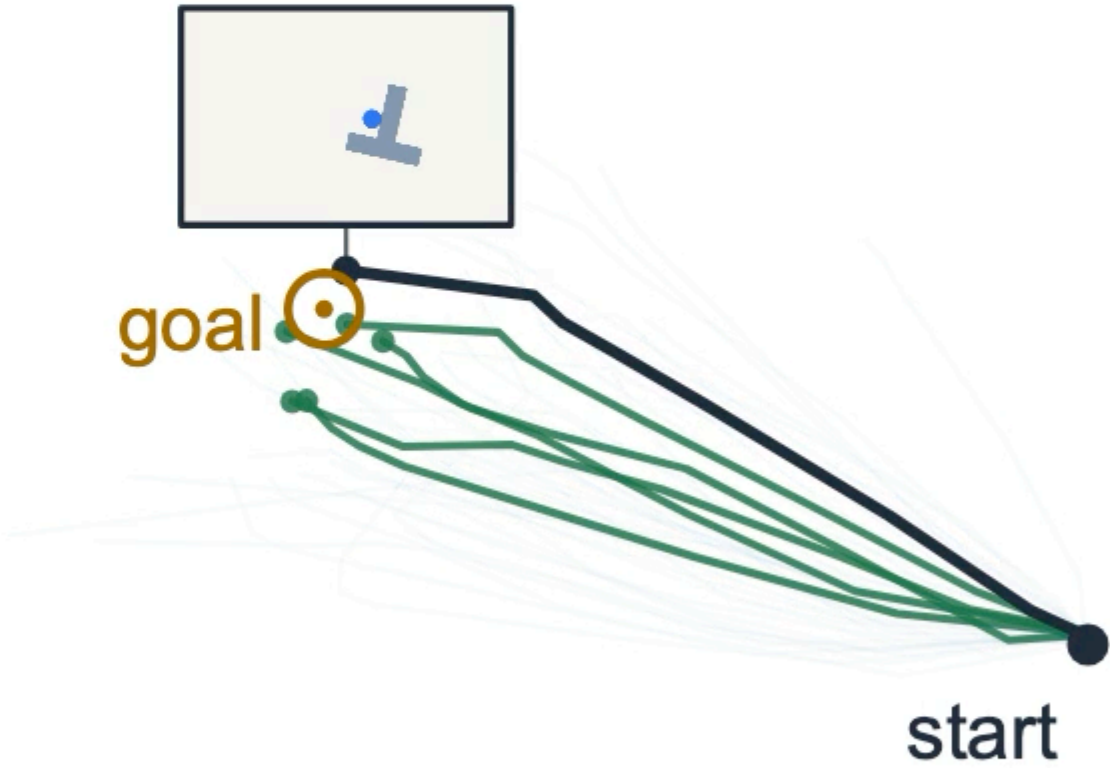
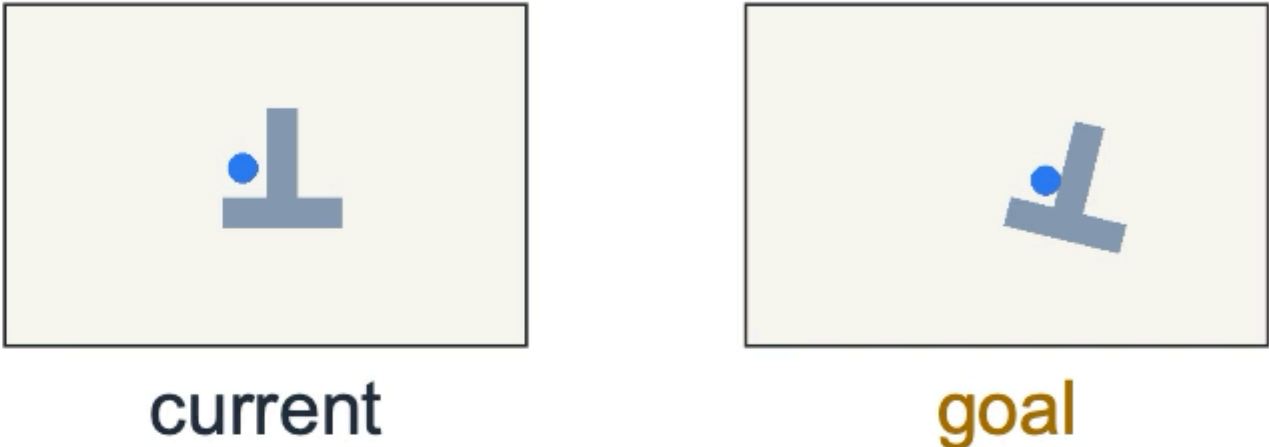
Finite samples may miss good action regions.

A Gaussian mean can lie between incompatible good plans.

An inaccurate model may reward actions that fail in the environment.

Resample from the updated actions · round 5

	action sequence	8D cost
#45	→ · → · · · → →	0.00029
#22	→ · → → → · · →	0.00037
#20	→ · → · · · · · →	0.00052
#31	→ · → · · · · · →	0.00063
#26	→ · → → · · · · ·	0.00117
#38	→ · → → → · · · ·	0.00127



mean proposal std 0.084

same PCA · enlarged view
best so far 0.00029

The same goal. Smaller action variation. Better sampled plans.

A1 physics · illustrative latents

Search rounds update action proposals; select the best evaluated sequence for this illustration.

LeWM reports this planning budget

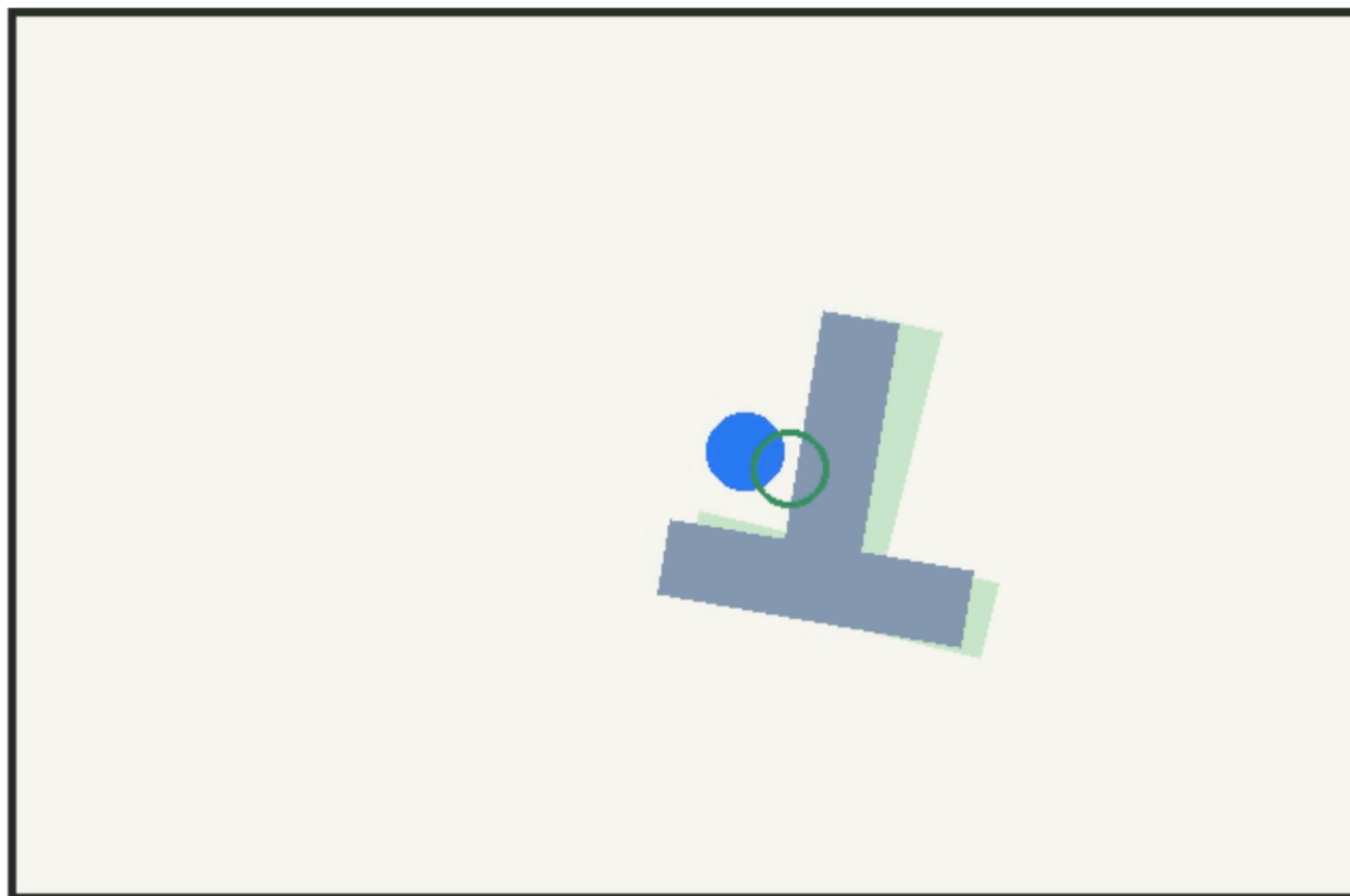
Planning solver. For planning, we use the Cross-Entropy Method (CEM). At each planning step, CEM samples 300 candidate action sequences and optimizes them for a maximum of 30 iterations in *PushT* and 10 iterations in the other environments. At each iteration, the top 30 trajectories are retained to update the sampling distribution, and the initial sampling variance is set to 1. The planning horizon is set to 5 steps, which corresponds to 25 environment timesteps due to the use of a frame skip of 5. We employ a receding-horizon Model Predictive Control (MPC) scheme with a horizon of 5, meaning that the entire optimized action sequence is executed before replanning. This configuration follows the setup used in [18].

$N = 300$ plans. $N_e = 30$ elites. $H = 5$ model steps.

PushT: up to 30 CEM rounds; execute all 5 planned model steps.

Execute the best sequence we actually evaluated.

Best sampled plan · round 6 · #42

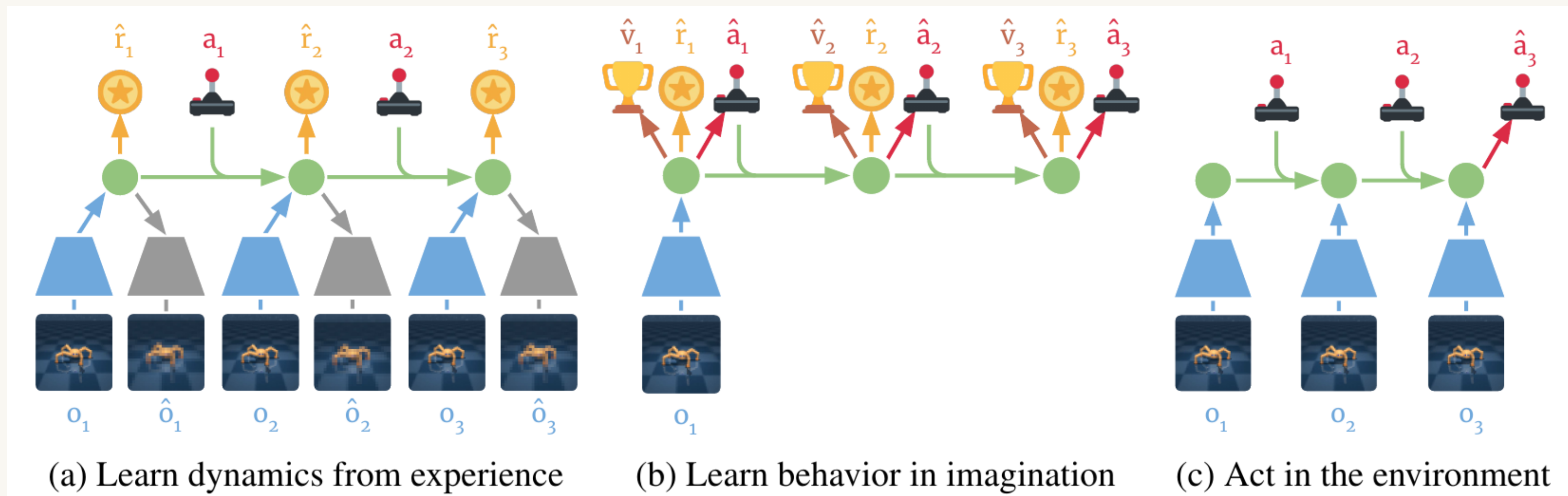


Sample → predict → score → refit → sample again

A1 physics · illustrative latents

Execute the entire selected sequence. This clip contains no feedback replanning.

Model-based RL uses predicted consequences



Model-based RL uses predicted action consequences to improve expected return.

Dreamer learns from real experience, improves behavior in imagination, then acts.

CEM is a planner inside a model-based agent

Question	Planning with CEM / PlaNet	Learning a policy / Dreamer
How is the model used?	Score candidate action sequences	Generate trajectories for actor–critic updates
What changes in behavior improvement?	A proposal over plans at this decision	Shared policy and value parameters
How is an action chosen?	Run a search using predicted returns	Evaluate the learned actor
What is shared?	An RSSM can predict states and rewards	An RSSM can predict states and rewards

CEM specifies a search method. Model-based RL specifies how prediction improves behavior.

Value estimates what happens beyond the horizon

$$x_t = (h_t, s_t), \quad V^\pi(x_t) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid x_t \right]$$

$$\hat{G}_H(U) = \sum_{k=0}^{H-1} \gamma^k \hat{r}_{t+k+1} + \gamma^H V_\psi(\hat{x}_{t+H})$$

Reward predicts one step. Value estimates the remaining return under a policy.

Example: $H = 2$, $\gamma = 0.9$. Plan A: (5, 0), tail $0 \rightarrow 5$. Plan B: (0, 0), tail $10 \rightarrow 8.1$.

The value target comes from imagined returns

$$a_t \sim \pi_\phi(\cdot \mid x_t), \quad \hat{x}_{t+1} \sim p_\theta(\cdot \mid x_t, a_t)$$

$$y_t = \hat{r}_{t+1} + \gamma(1 - d_{t+1}) \text{sg}[V_\psi(\hat{x}_{t+1})]$$

$$\mathcal{L}_V(\psi) = \frac{1}{2} \mathbb{E}[(V_\psi(x_t) - \text{sg}[y_t])^2]$$

One-step bootstrap recap:
update the critic ψ while holding
the target fixed.

Dreamer uses multi-step λ -
returns. The full derivation follows
after fall break.

Policy gradient updates policy parameters

$$\nabla_{\phi} J = \mathbb{E}_{\pi_{\phi}} \left[\sum_t \gamma^t \nabla_{\phi} \log \pi_{\phi}(a_t \mid x_t) A^{\pi_{\phi}}(x_t, a_t) \right]$$

$$\hat{A}_t = \hat{G}_t - V_{\psi}(x_t), \quad \phi \leftarrow \phi + \alpha \widehat{\nabla_{\phi} J}$$

Use the Applied ML result: better-than-expected actions become more likely.

CEM updates a plan proposal. Policy gradient updates a reusable policy.

Dreamer differentiates through imagined trajectories

$$\max_{\phi} \mathbb{E}_{q_{\theta}, q_{\phi}} \left(\sum_{\tau=t}^{t+H} V_{\lambda}(s_{\tau}) \right), \quad (7)$$

$$\min_{\psi} \mathbb{E}_{q_{\theta}, q_{\phi}} \left(\sum_{\tau=t}^{t+H} \frac{1}{2} \left\| v_{\psi}(s_{\tau}) - V_{\lambda}(s_{\tau}) \right\|^2 \right). \quad (8)$$

$$a_t = g_{\phi}(x_t, \epsilon_t), \quad x_{t+1} = f_{\theta}(x_t, a_t, \xi_t)$$

$$\phi \leftarrow \phi + \alpha \nabla_{\phi} \mathbb{E}[\hat{G}_H] \quad \text{through actions, states, rewards and values}$$

During the actor update, hold θ and ψ fixed but retain derivatives through their inputs.

RSSM supplies the imagined dynamics. Actor and critic learn how to use them.

From learned predictions to better decisions

Representation and dynamics losses make future states predictable.

CEM searches actions at a decision. A learned policy reuses behavior across decisions.

Value connects finite imagination to longer-term return.