

CIS 6280 · FALL 2026

WORLD MODELS

LECTURE 9

Planning and Control with World Models

Course announcements

SEP 24

Assignment 1 due

OFFICE HOURS

Today, 4:30–5:30 PM

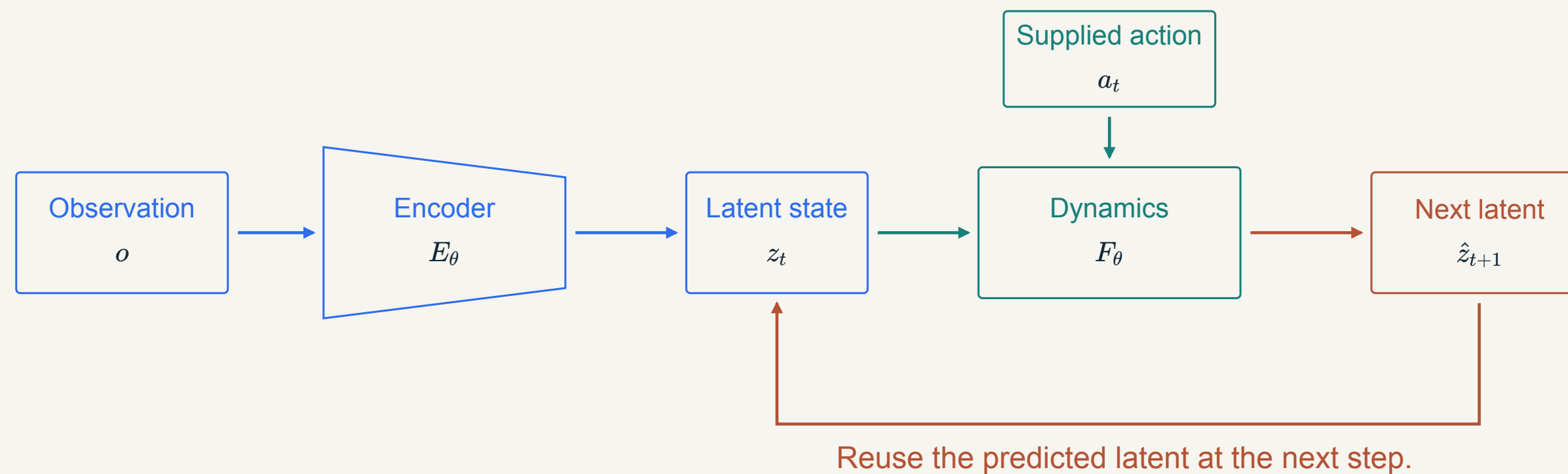
On campus · September 24

LECTURE FORMAT

Today's lecture is on Zoom

Due to travel

TD-MPC: latent dynamics



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

The observation input may include recent frames. Rollout reuses the predicted latent.

Encoder

Deterministic: $z = E_\theta(o)$

Prediction

Deterministic autoregressive latent prediction.

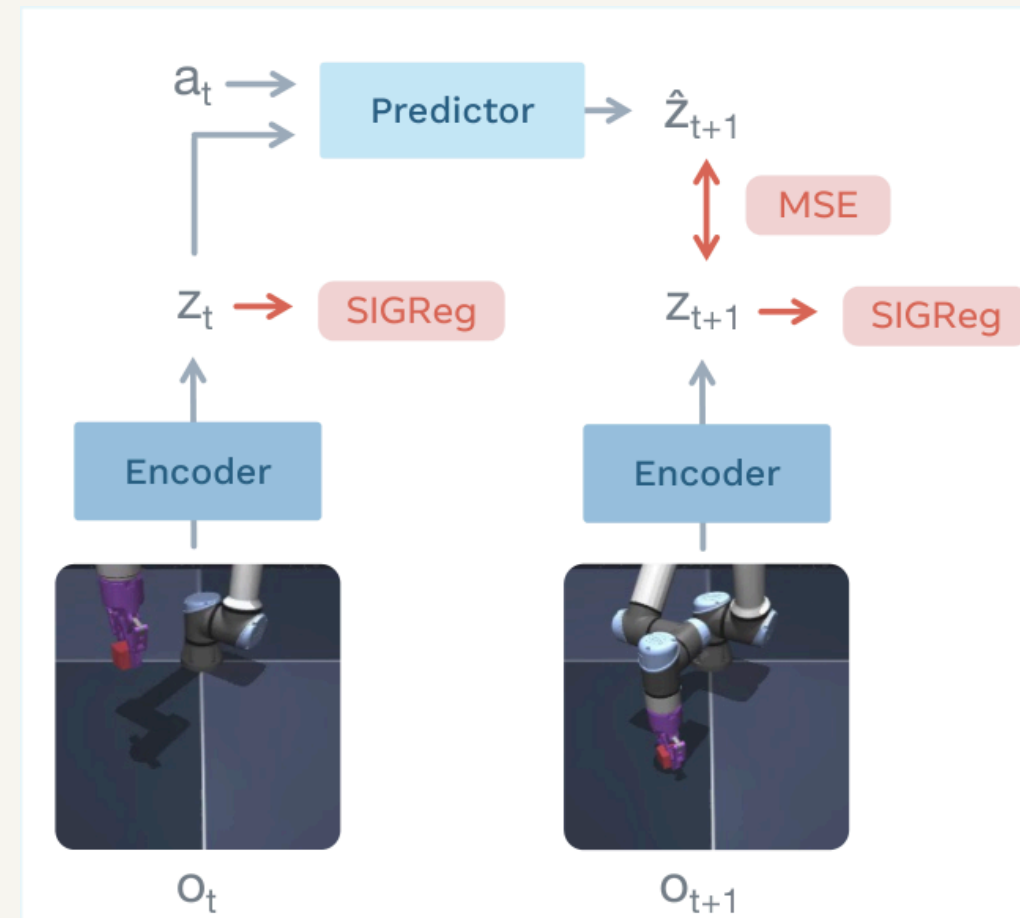
Loss

Squared L2 latent consistency.

Full method

Reward / value losses also matter; L2 consistency alone permits collapse.

LeWorldModel: joint feature prediction



Original joint encoder/predictor training pipeline

$$\hat{z}_{t+1} = F_{\psi}(z_{\text{history}}, a_{\text{history}})$$

Encoder

Deterministic continuous features, learned jointly.

State

History retained inside the predictor.

Prediction

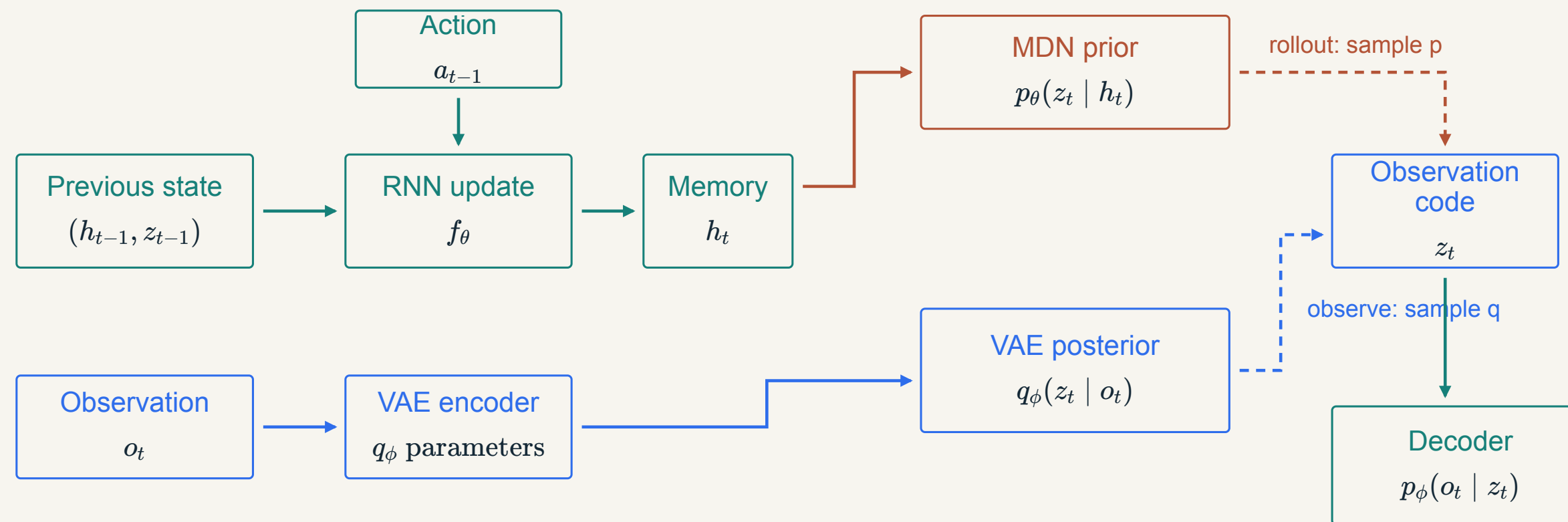
Deterministic autoregressive latent prediction.

Loss

MSE prediction + SIGReg to constrain the representations.

V-JEPA 2-AC: related feature-prediction interface; pretrained frozen encoder, with visual / pose / action context.

World Models: vision, memory, and rollouts



Encode the current frame as z ; retain earlier history in h .

$$h_t = f_\theta(h_{t-1}, z_{t-1}, a_{t-1})$$

Observe: sample the VAE posterior. Imagine: sample the MDN prior, feed the generated code back, and update memory.

Encoder

Stochastic VAE code:

$$z_t \sim q_\phi(\cdot | o_t)$$

Latent meaning

z_t encodes the current frame; h_t summarizes earlier codes and actions.

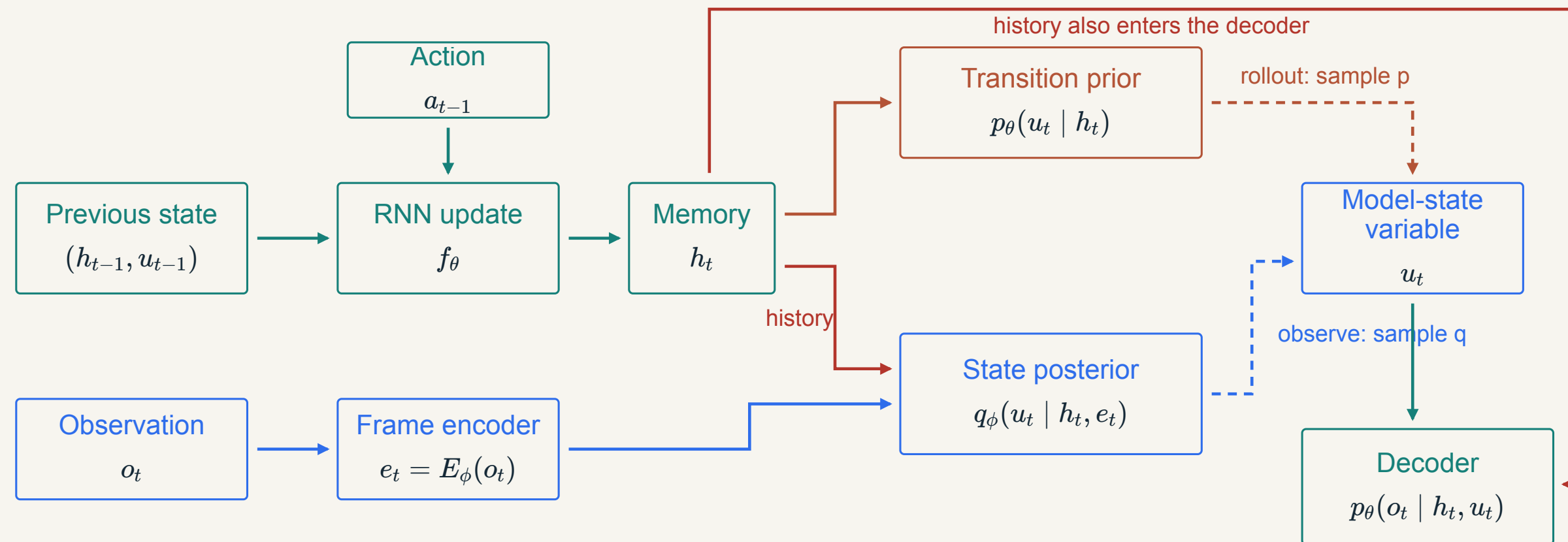
Prediction

Deterministic h update; probabilistic z update through an MDN.

Loss

VAE objective, then next-code MDN negative log likelihood.

RSSM: PlaNet and Dreamer



Infer the latent state u from history + the current observation.

$$h_t = f_\theta(h_{t-1}, u_{t-1}, a_{t-1}), \quad \tilde{s}_t = (h_t, u_t)$$

Same RNN + probabilistic latent structure. Red arrows show RSSM's added history dependencies.

Latent meaning

u_t is inferred using history and the current observation.

Posterior

$q_\phi(u_t | h_t, E(o_t))$ uses the temporal context.

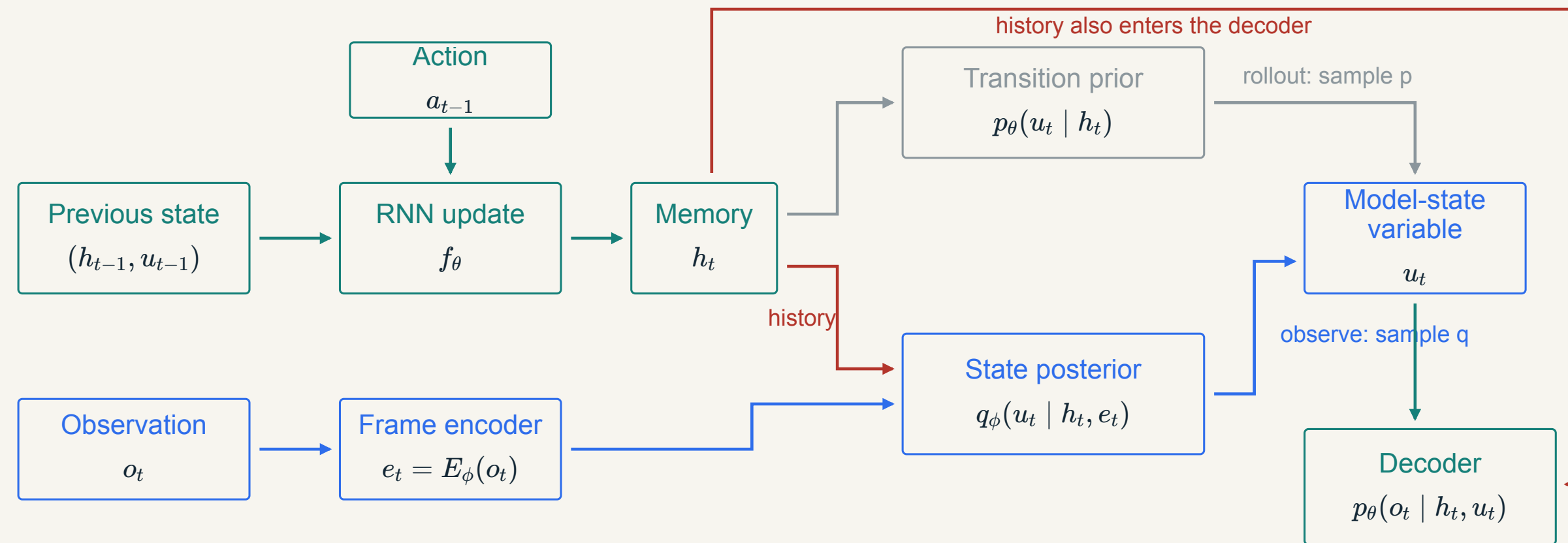
Shared structure

Deterministic RNN memory + probabilistic latent transitions.

Difference: training

Joint observation NLL + KL to the temporal prior.

RSSM: updating with observations



Infer the latent state u from history + the current observation.

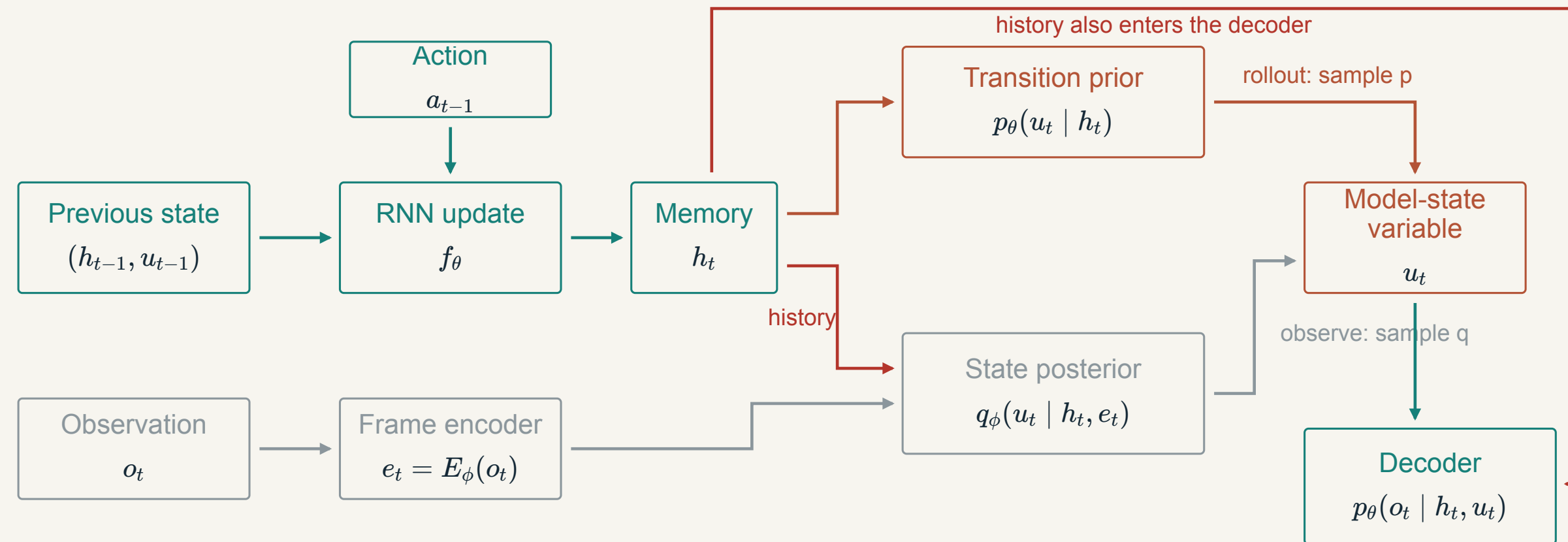
World Models 2018: current frame

$$z_t \sim q_\phi(\cdot | o_t)$$

RSSM: history + current observation

$$u_t \sim q_\phi(\cdot | h_t, E_\phi(o_t))$$

RSSM: imagining without observations

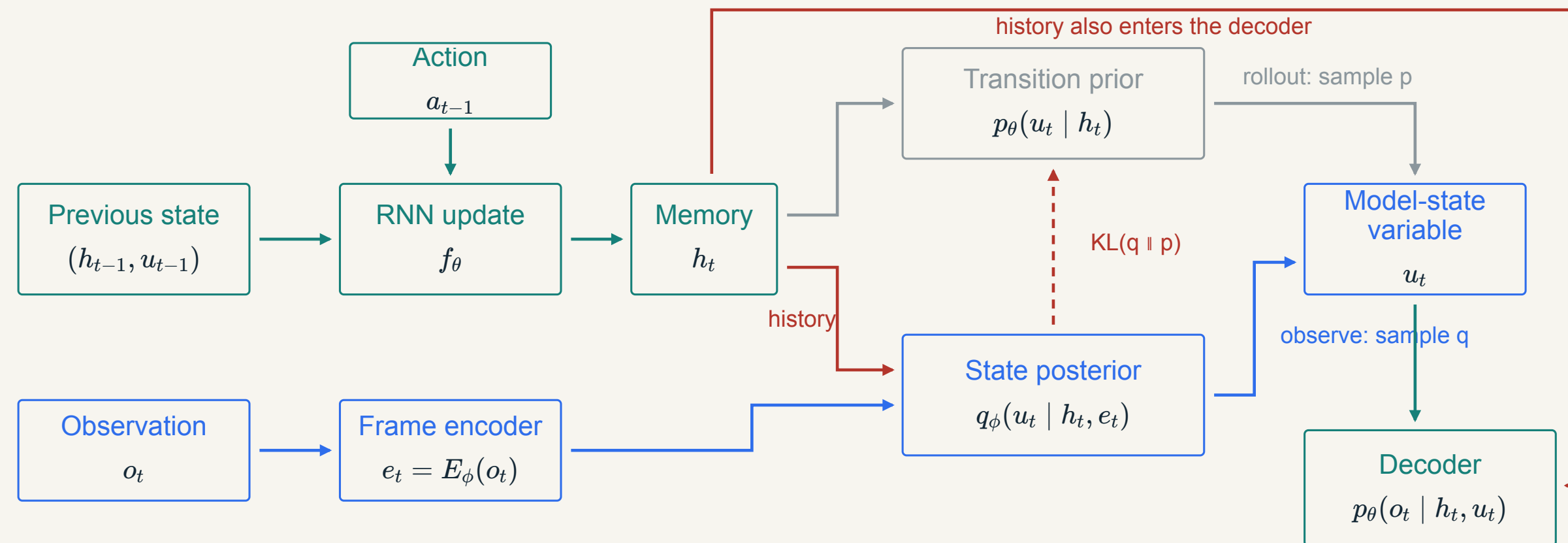


Infer the latent state u from history + the current observation.

$$u_t \sim p_{\theta}(u_t | h_t), \quad h_{t+1} = f_{\theta}(h_t, u_t, a_t)$$

During imagination, sample the prior and continue. The observation branch is unavailable.

RSSM: learning the model

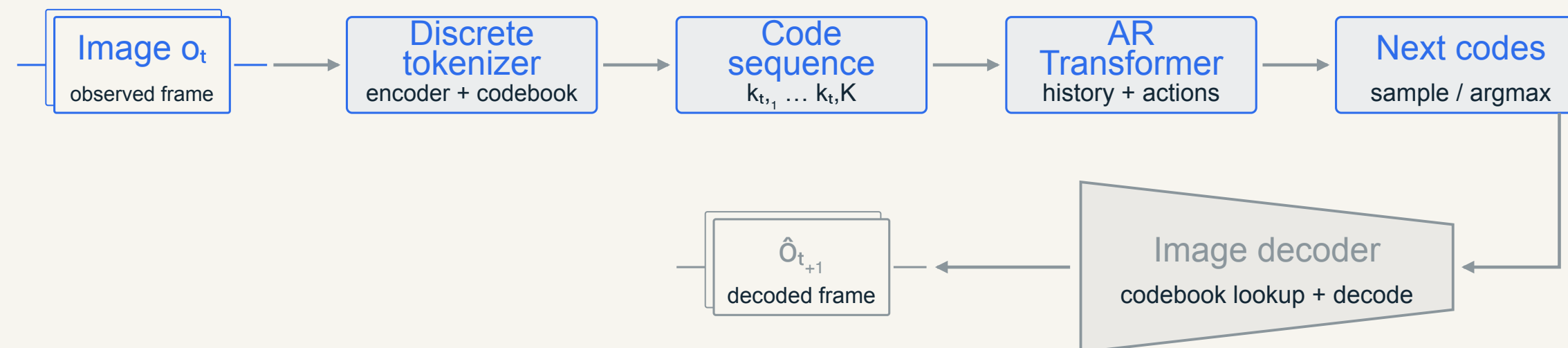


Infer the latent state u from history + the current observation.

$$\mathcal{L}_t = \mathbb{E}_q[-\log p_\theta(o_t | h_t, u_t)] + \beta D_{\text{KL}}(q_\phi(u_t | h_t, e_t) || p_\theta(u_t | h_t))$$

Joint temporal training: the posterior is matched to a history-dependent prior.

IRIS: discrete codes and future generation



Separate losses; repeated updates of both autoencoder and Transformer.

Within a frame: generate the codes. Across time: append the next action and continue. Decoding only visualizes the rollout.

Encoder

Deterministic assignment to discrete image codes.

State

Code and action history inside the sequence model.

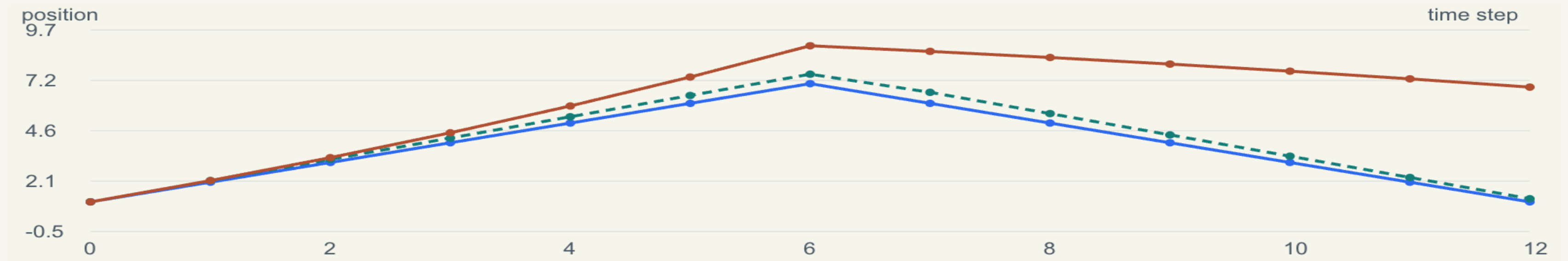
Prediction

Probabilistic AR: categorical next-token distribution.

Loss

Autoencoder reconstruction / codebook terms; token cross-entropy for dynamics.

Compounding error and observation updates



True system One-step prediction with true-state input Free-running rollout

$$x_0 = \hat{x}_0 = 1; g = 1.08; a_0 \dots a_5 = +1; a_6 \dots a_{11} = -1; e_t = \hat{x}_t - x_t.$$

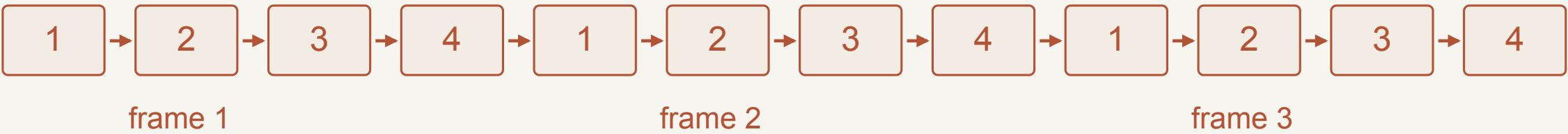
Step 12/12 · one-step error: 0.160 · rollout error: 5.823

$$x_{t+1} = x_t + a_t, \quad \hat{x}_{t+1} = g\hat{x}_t + a_t \quad \Rightarrow \quad e_{t+1} = ge_t + (g - 1)x_t$$

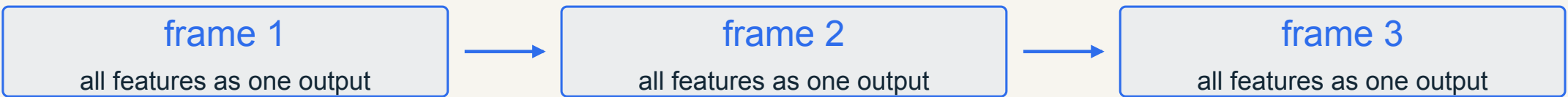
Remedies: multi-step consistency, a validated rollout horizon, and state updates when real observations arrive.

Sequential generation cost

Token by token



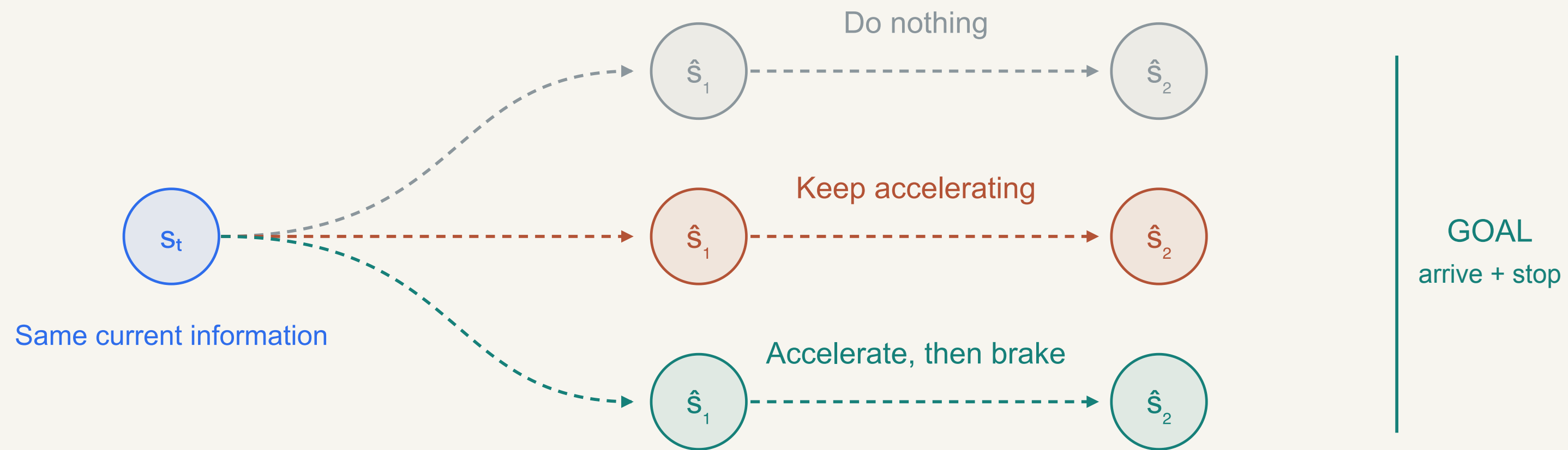
Frame / block prediction



Cost	Possible mitigation	Tradeoff
Many serial token steps	Fewer tokens; frame/block prediction	Compression may lose information
Repeated computation / long context	Caching; context compression	Memory use or forgotten history
Many iterative sampling steps	Distillation / shortcut methods	Additional training; later lectures

H frames $\times$ K serial codes $\approx HK$ prediction steps. Compare actual latency too: steps have different costs.

How can a world model help an agent choose actions?



L8: given actions, predict. L9: use predictions to choose actions.

Goal and task cost: arrive and stop

No action



$v = 0$

At target, still moving



$v = 2$



At target, stopped



$v = 0$

target position $p = 1$

$$J = (p_H - 1)^2 + v_H^2 + 0.1 \sum_{j=0}^{H-1} a_j^2$$

position error

remaining motion

action effort

Prediction loss and task cost

Learn to predict

$$\min_{\theta} \mathcal{L}(f_{\theta}(s_t, a_t), s_{t+1}^{\text{observed}})$$

Compare with what actually happened.

Change model parameters θ .

Plan for a task

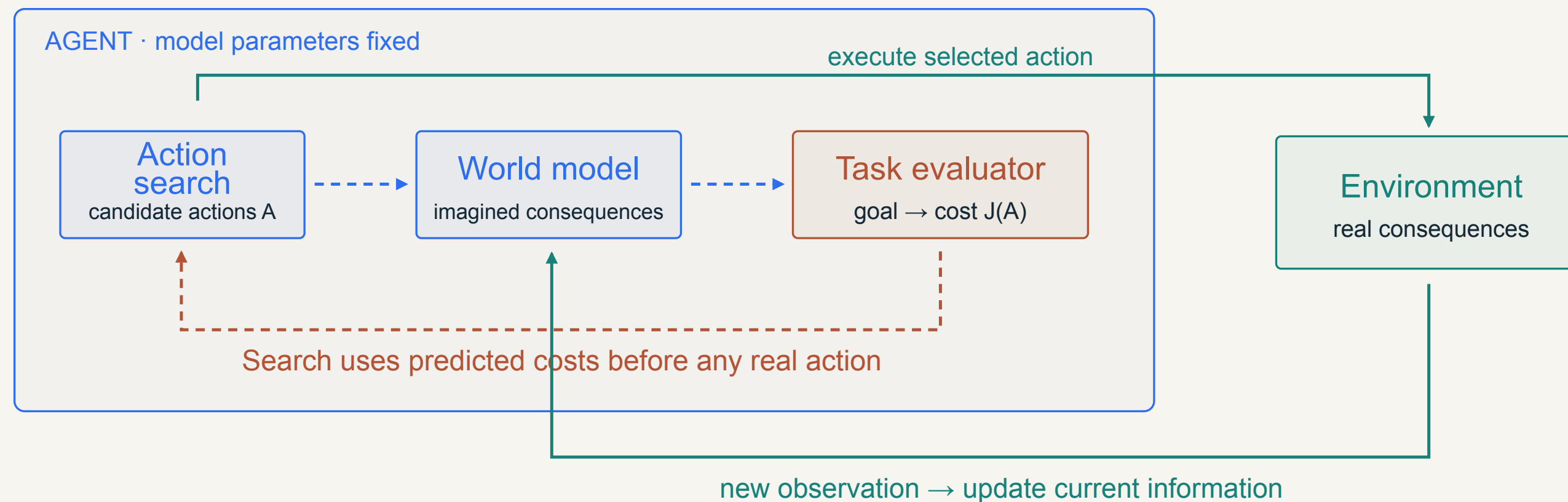
$$\min_{\mathbf{A}} J(\text{Rollout}_{\theta}(\mathcal{H}_t, \mathbf{A}), g)$$

Compare with what we want to happen.

Change candidate actions $\mathbf{A}$. Keep θ fixed.

An accurate prediction is useful only after we decide which outcomes to prefer.

Inside the planner and the real feedback loop



Dashed arrows: internal computation. Solid arrows: real action and observation.

Planning with an LGSSM

$$s_{k+1} = As_k + Ba_k + w_k, \quad w_k \sim \mathcal{N}(0, Q)$$

$$o_k = Cs_k + \nu_k, \quad \nu_k \sim \mathcal{N}(0, R)$$

Current information

$$p(s_t \mid \mathcal{H}_t) = \mathcal{N}(\mu_t, P_t)$$

Planning variables

$$\mathbf{A} = (a_t, \dots, a_{t+H-1}) \in \mathcal{A}^H$$

Choose a sequence that minimizes expected task cost over the next H steps.

The model and current belief are given. The action sequence is the optimization variable.

Predictive distributions under candidate actions

Condition on the current history and a fixed candidate action sequence.

$$p(s_{t+j} \mid \mathcal{H}_t, \mathbf{A}) = \mathcal{N}(\mu_j, P_j), \quad \mu_0 = \mu_t, P_0 = P_t$$

Prediction recursion

$$\mu_{j+1} = A\mu_j + Ba_{t+j}$$

$$P_{j+1} = AP_jA^\top + Q$$

Repeat for $j = 0, \dots, H - 1$. No future observations are available during this rollout.

Actions change the predicted mean. With fixed additive noise, the covariance is independent of the candidate actions.

Expected quadratic task cost

g is the target state. The scalar $\lambda > 0$ controls the penalty on action effort.

$$J(\mathbf{A}) = \sum_{j=1}^H \mathbb{E}[\|s_{t+j} - g\|_2^2] + \lambda \sum_{j=0}^{H-1} \|a_{t+j}\|_2^2$$

Expected squared distance

$$\mathbb{E}[\|s - g\|_2^2] = \|\mu - g\|_2^2 + \text{tr}(P), \quad s \sim \mathcal{N}(\mu, P)$$

$$J(\mathbf{A}) = \sum_{j=1}^H [\|\mu_j - g\|_2^2 + \text{tr}(P_j)] + \lambda \sum_{j=0}^{H-1} \|a_{t+j}\|_2^2$$

The expectation is exact. Here the covariance term is constant with respect to the action sequence.

Choosing an action sequence

Each candidate action sequence determines the predicted means:

$$\mu_{j+1} = A\mu_j + Ba_{t+j}, \quad \mu_0 = \mu_t$$

Choose the sequence with the lowest task cost

$$\mathbf{A}^* = \arg \min_{\mathbf{A} \in \mathcal{A}^H} \left[\sum_{j=1}^H \|\mu_j(\mathbf{A}) - g\|_2^2 + \lambda \sum_{j=0}^{H-1} \|a_{t+j}\|_2^2 \right]$$

The covariance term is the same for every candidate, so it does not affect which sequence we choose.

$\mathcal{A}^H$ is the set of valid action sequences. The model, current belief, and goal remain fixed.

Linear dynamics make the predicted means affine in the actions. The planning objective is therefore quadratic.

Model predictive control

Plan at time t

$$\mathbf{A}_t^* = \arg \min_{\mathbf{A} \in \mathcal{A}^H} J(\mathbf{A}; \mathcal{H}_t, g)$$

Execute only the first action

$$a_t = [\mathbf{A}_t^*]_0$$

Receive the real observation o_{t+1} . Update the Gaussian belief using the observation model.

$$p(s_{t+1} \mid \mathcal{H}_{t+1}) = \mathcal{N}(\mu_{t+1}, P_{t+1})$$

MPC repeats the planner after new feedback. At time $t + 1$, optimize a new action sequence.

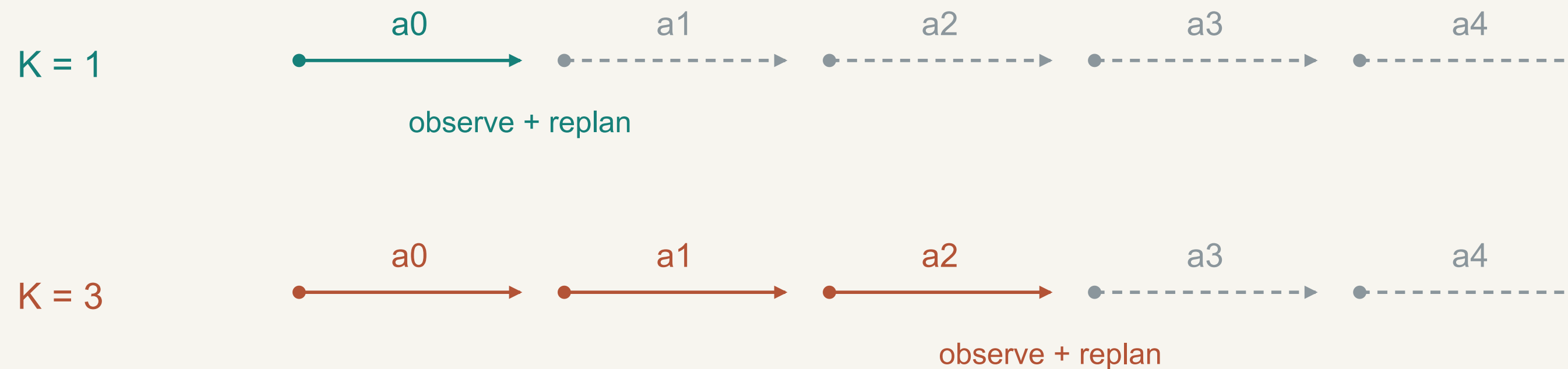
Planning with a learned latent world model

Planning step	Linear-Gaussian dynamics + quadratic cost	Learned latent world model
Initialize	Current Gaussian belief	Inferred latent state or retained history
Predict	Exact mean and covariance recursion	Learned rollout; sample futures if needed
Evaluate	Exact expected quadratic cost	Task cost on predicted outcomes
Optimize	Quadratic objective in the actions	Generally nonlinear objective in the actions

$$\mathbf{A}_t^* \approx \arg \min_{\mathbf{A} \in \mathcal{A}^H} \mathbb{E}_{p_\theta(S|\mathcal{H}_t, \mathbf{A})} [\text{Cost}(S, \mathbf{A}; g)]$$

Keep the planning objective and feedback loop. Replace the prediction and optimization calculations.

Prediction horizon H and execution interval K



Predict $H = 5$ model steps in both cases

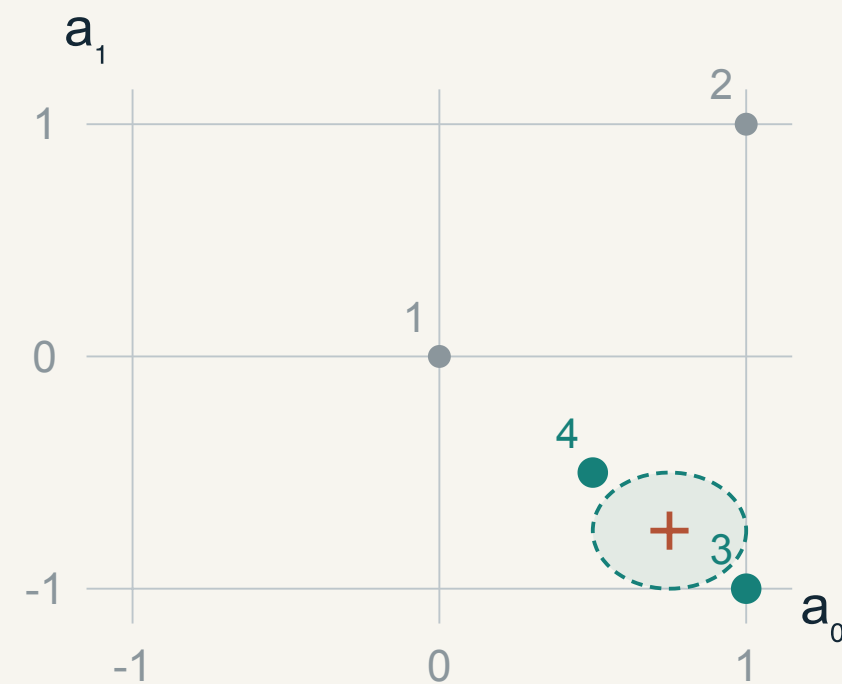
H · prediction horizon

How far do we look ahead?

K · execution interval

How long until new feedback?

CEM: a distribution over action sequences



Cross-Entropy Method

$$\mathbf{A} = (a_{t|t}, \dots, a_{t+H-1|t}) \in \mathbb{R}^{Hd_a}$$

$$q_k(\mathbf{A}) = \mathcal{N}(\boldsymbol{\mu}^{(k)}, \text{diag } \mathbf{v}^{(k)})$$

Sample N candidates from a proposal over actions.

H: rollout steps · N_e : elites · l: search rounds

k is a search iteration. Physical decision time t has not advanced.

CEM: roll out, score, and select elites

Illustrative action-search batch: take these four rollout costs as given.

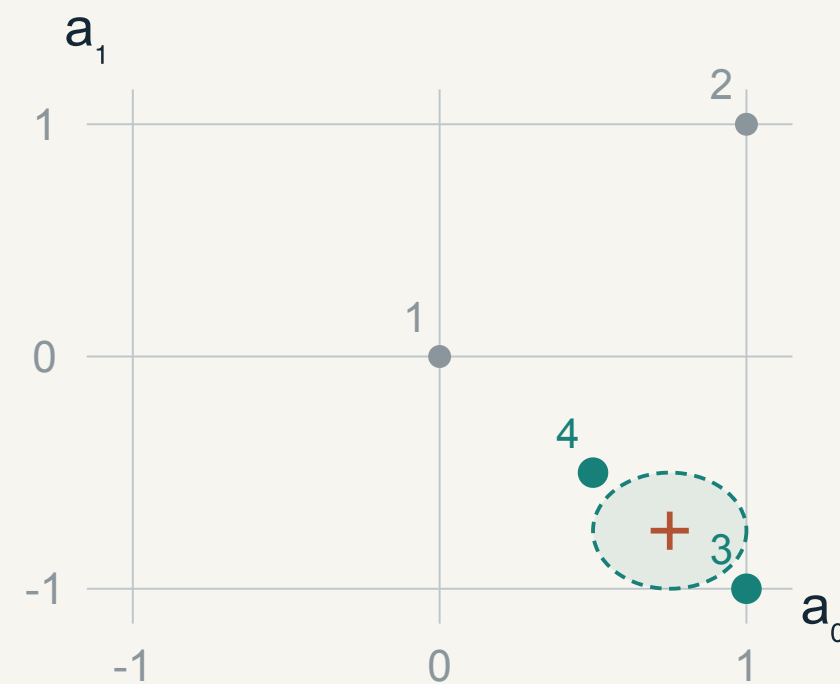
Candidate A_i	Predicted cost	Elite?
(0, 0)	1.00	—
(1, 1)	4.20	—
(1, -1)	0.20	✓ lowest two
(0.5, -0.5)	0.30	✓ lowest two

$$J_i = J(\text{Rollout}_\theta(\mathcal{H}_t, \mathbf{A}_i), g), \quad \mathcal{E}_k = \text{Bottom}_{N_e} \{J_i\}_{i=1}^N$$

Every candidate starts from its own copy of the same current information.

CEM: update the mean and variance

$$\hat{\boldsymbol{\mu}}_k = \frac{1}{N_e} \sum_{i \in \mathcal{E}_k} \mathbf{A}_i^{(k)}, \quad \hat{\mathbf{v}}_k = \frac{1}{N_e} \sum_{i \in \mathcal{E}_k} (\mathbf{A}_i^{(k)} - \hat{\boldsymbol{\mu}}_k)^{\odot 2}$$



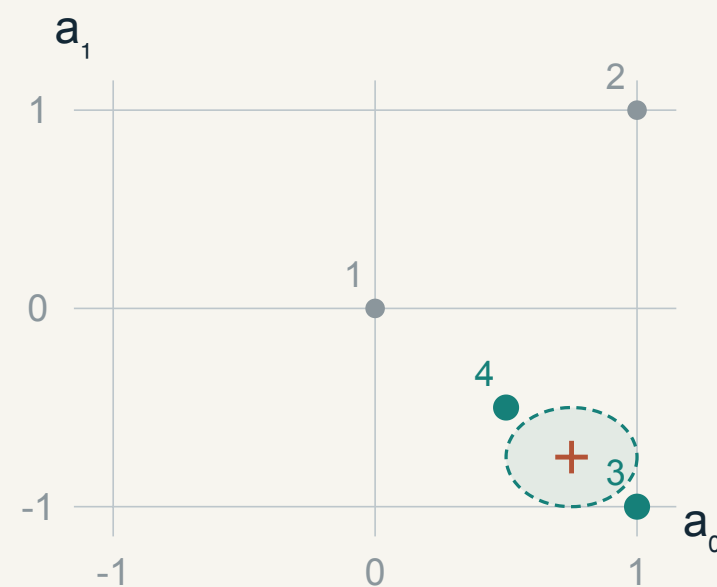
$$\begin{aligned} \hat{\boldsymbol{\mu}}_0 &= (0.75, -0.75) \\ \hat{\mathbf{v}}_0 &= (0.0625, 0.0625) \\ \boldsymbol{\mu}^{(k+1)} &= \hat{\boldsymbol{\mu}}_k, \quad \mathbf{v}^{(k+1)} = \hat{\mathbf{v}}_k \end{aligned}$$

Optional: smoothing and a variance floor.

Refit the action proposal. The world model does not change.

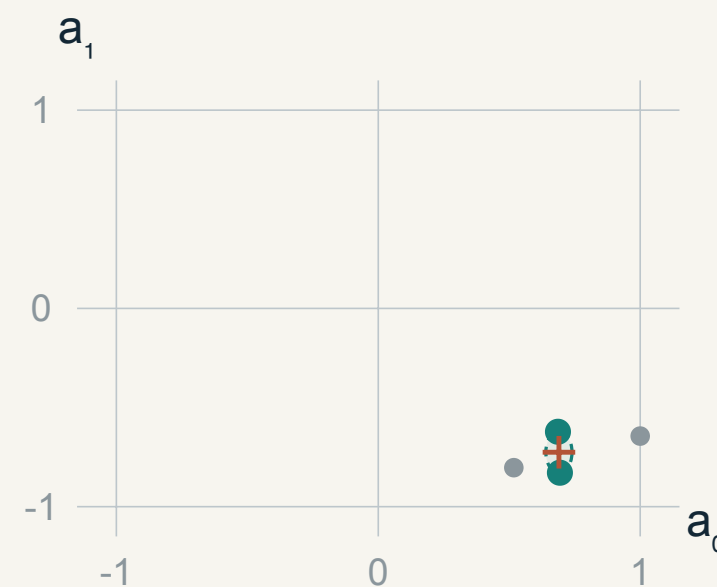
CEM: repeat the search before acting

Round 1



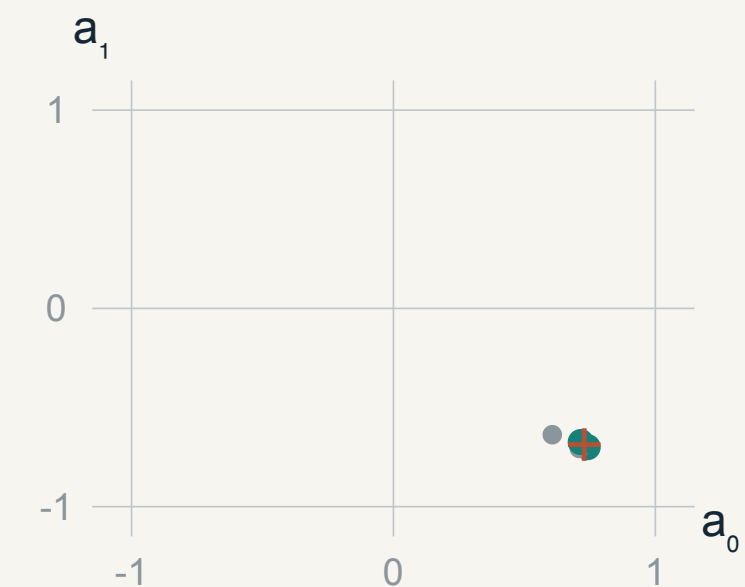
Cost of updated mean: **0.175**

Round 2



Cost of updated mean: **0.19773**

Round 3



Cost of updated mean: **0.17575**

Sample → score → keep elites → refit → sample again.

Green: elites · cross: refitted mean · ellipse: one standard deviation. Constructed first batch; seeded later draws.

Repeat before acting. Small batches need not improve the mean every round.

CEM: the complete planning algorithm

```
initialize mu, v
for k = 0, ..., I-1:
    A = clip(sample_gaussian(mu, v, N), action_bounds)
    J = rollout_and_score(A, current_information, goal)
    E = argsort(J)[:N_e]
    mu_hat = mean(A[E], axis=candidate)
    v_hat = mean((A[E] - mu_hat)^2, axis=candidate)
    mu = (1-alpha)*mu + alpha*mu_hat
    v = maximum(v_min, (1-alpha)*v + alpha*v_hat)
return clip(mu, action_bounds)
```

Inputs

Current information, goal, fixed model, action bounds.

N candidates · N_e elites · I rounds

$\alpha \in (0, 1]$: refit rate

$v_{\min}$: variance floor

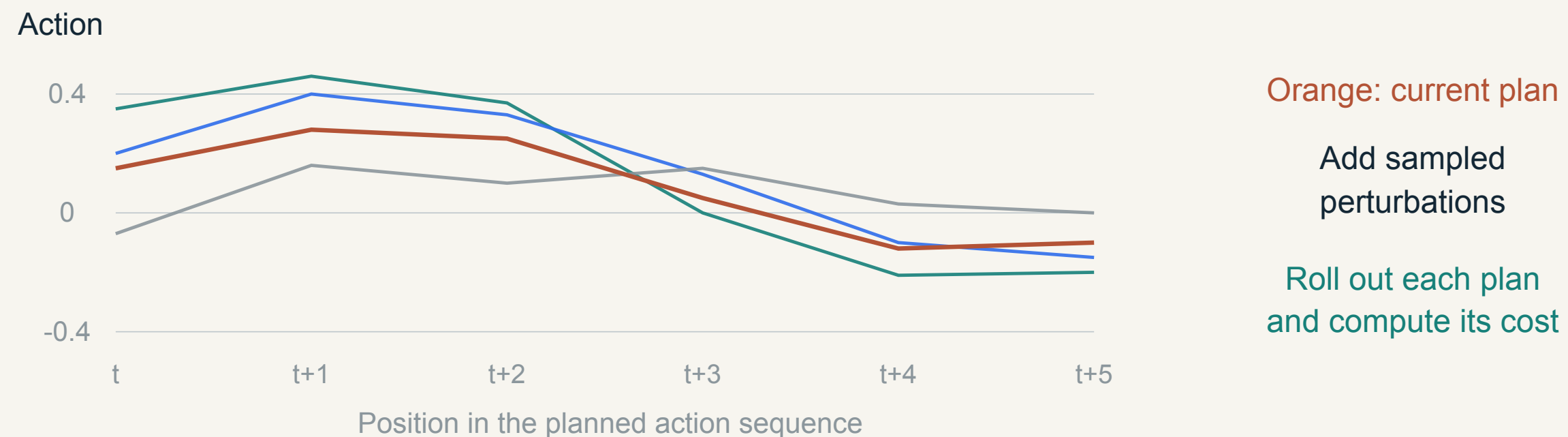
After the search

Score and check the returned mean plan.

MPC executes its first action, then observes and replans.

MPPI: perturb a current action plan

MPPI = Model Predictive Path Integral control. Start with a current action plan $\bar{\mathbf{A}}$.



$$\mathbf{A}_i = \bar{\mathbf{A}} + \boldsymbol{\epsilon}_i, \quad \boldsymbol{\epsilon}_i \sim \mathcal{N}(0, \Sigma_a)$$

Σ_a sets how widely we perturb the planned actions.

Try changes to the current plan. Use the world model to evaluate their predicted consequences.

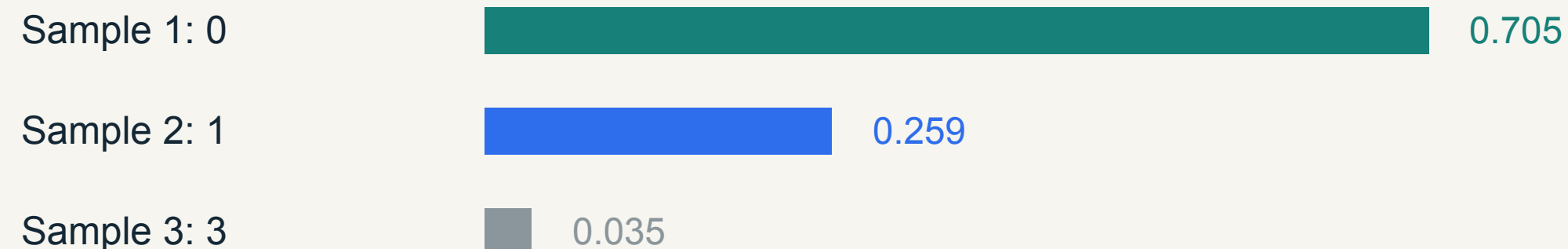
MPPI: turn rollout costs into weights

Give lower-cost sampled futures more influence on the action update.

$$w_i = \frac{\exp[-(\tilde{S}_i - \beta)/\tau]}{\sum_{\ell} \exp[-(\tilde{S}_{\ell} - \beta)/\tau]}, \quad \beta = \min_i \tilde{S}_i$$

Corrected rollout cost

Normalized influence on the update



Illustrative corrected costs, temperature $\tau = 1$. Weights sum to 1.

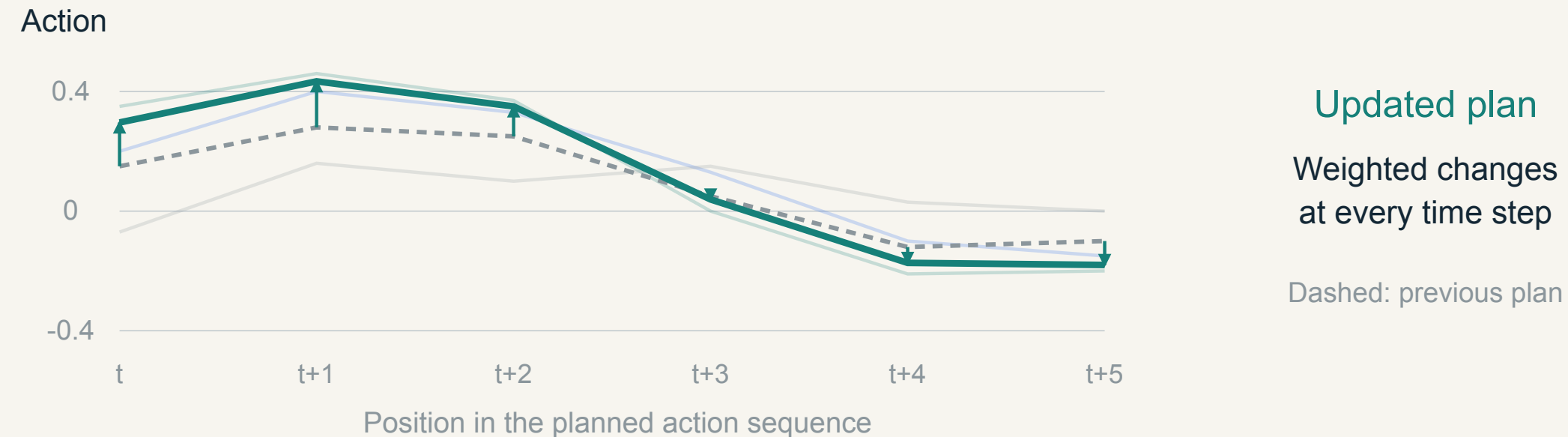
Smaller τ : concentrate weight on the best sampled costs.

Larger τ : spread weight across more samples.

$\tau > 0$ is the temperature. $\tilde{S}$ includes task cost and the control / sampling corrections of the MPPI formulation.

MPPI: update the plan and act

$$\Delta \mathbf{A} = \sum_i w_i \boldsymbol{\epsilon}_i, \quad \bar{\mathbf{A}}_{\text{new}} = \bar{\mathbf{A}} + \Delta \mathbf{A}$$



Repeat around the updated plan if time allows. Execute its first action, observe, and replan.

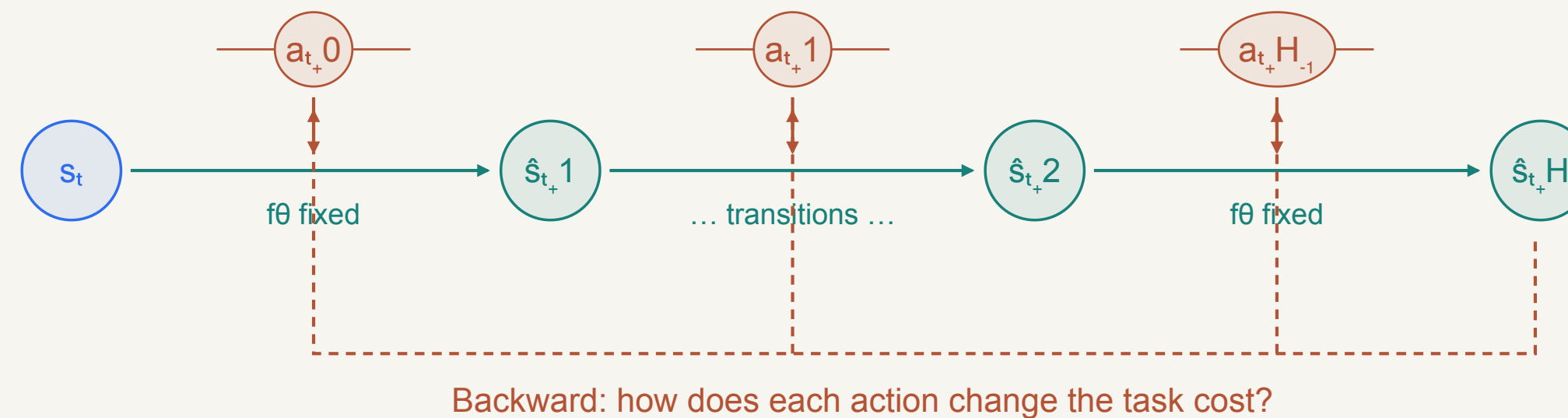
Shift the remaining plan forward to initialize the next decision. Check the updated plan and action bounds before execution.

The weighted perturbations change the action plan. The world model stays fixed.

Gradient-based planning

CEM and MPPI use sampled costs to improve an action plan.

When the rollout and task cost are differentiable, their gradient gives a local update direction.



$$\mathbf{A} \leftarrow \mathbf{A} - \eta \nabla_{\mathbf{A}} J(\mathbf{A}; \theta) \quad \text{with } \theta \text{ fixed}$$

The gradient supplies a local direction for improving the action sequence.

Gradients through a predicted trajectory

For a deterministic rollout and a terminal task cost:

$$s_{j+1} = f_{\theta}(s_j, a_j), \quad J(\mathbf{A}) = \ell(s_H, g) + \sum_{j=0}^{H-1} r(a_j)$$

$$\nabla_{a_j} J = \nabla r(a_j) + \left(\frac{\partial s_H}{\partial a_j} \right)^{\top} \nabla_{s_H} \ell(s_H, g)$$

An early action influences every later state

$$\frac{\partial s_H}{\partial a_j} = \frac{\partial f_{H-1}}{\partial s_{H-1}} \dots \frac{\partial f_{j+1}}{\partial s_{j+1}} \frac{\partial f_j}{\partial a_j}$$

Backpropagate through the imagined rollout. Keep the initial state and model parameters fixed.

Projected gradient updates

$$\mathbf{A}^{(k+1)} = \Pi_{\mathcal{A}^H} \left[\mathbf{A}^{(k)} - \eta_k \nabla_{\mathbf{A}} J(\mathbf{A}^{(k)}) \right]$$

Gradient update

Change all actions in the planned sequence using the current cost gradient.

η_k controls the update size.

Action constraints

Project the updated sequence into the valid action set.

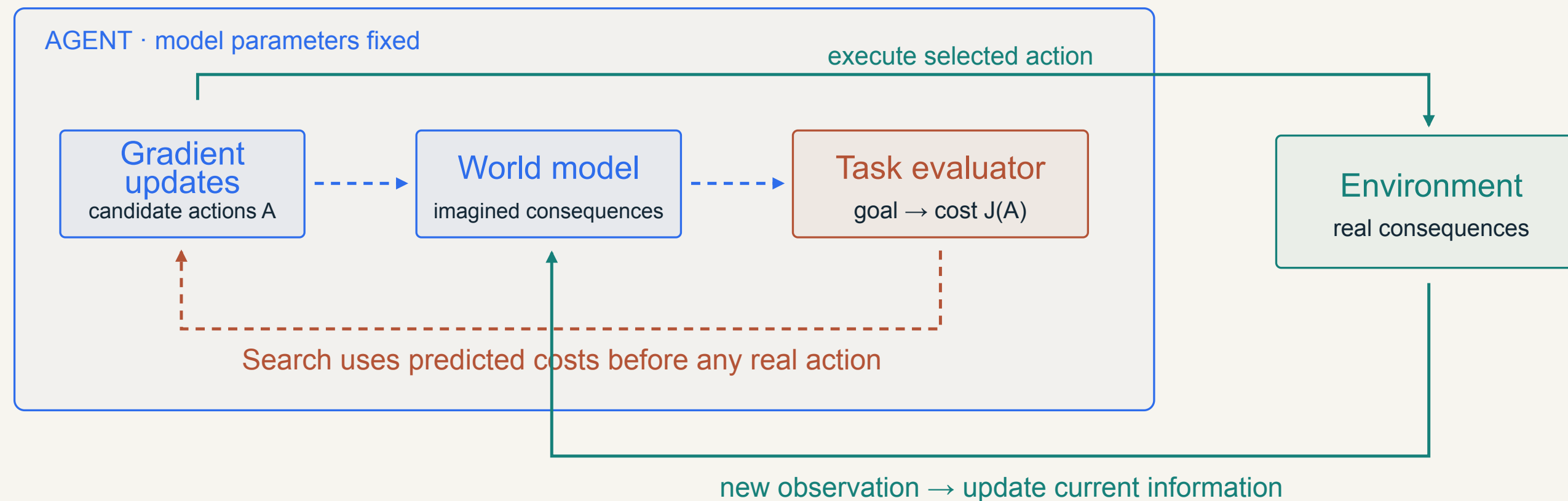
For box bounds, projection clips each coordinate.

$$[\Pi_{\mathcal{A}^H}(\mathbf{u})]_i = \min(a_{\max,i}, \max(a_{\min,i}, u_i))$$

Repeat within the planning budget, then evaluate the final sequence before executing its first action.

Projection enforces action bounds. A fixed step size does not guarantee lower cost.

Put gradient updates inside MPC



Initialize → roll out → differentiate → update → repeat.

Optional warm start: shift the previous plan and append one valid action.

Stochastic futures: reparameterization returns

$$\hat{s}_{j+1} = f_{\theta}(\hat{s}_j, a_j, \epsilon_j), \quad \epsilon_j \sim p(\epsilon) \text{ independent of } \mathbf{A}$$

$$\nabla_{\mathbf{A}} \mathbb{E}[J] \approx \frac{1}{M} \sum_{m=1}^M \nabla_{\mathbf{A}} J\left(\text{Rollout}_{\theta}(\mathcal{H}_t, \mathbf{A}, \epsilon^{(m)})\right)$$

From L7 to planning

Hold base noise fixed while differentiating each sampled future.

Check the full path

Categorical sampling and MDN component choices need additional treatment.

Continuous latents alone do not guarantee a usable pathwise gradient.

Limits of action gradients

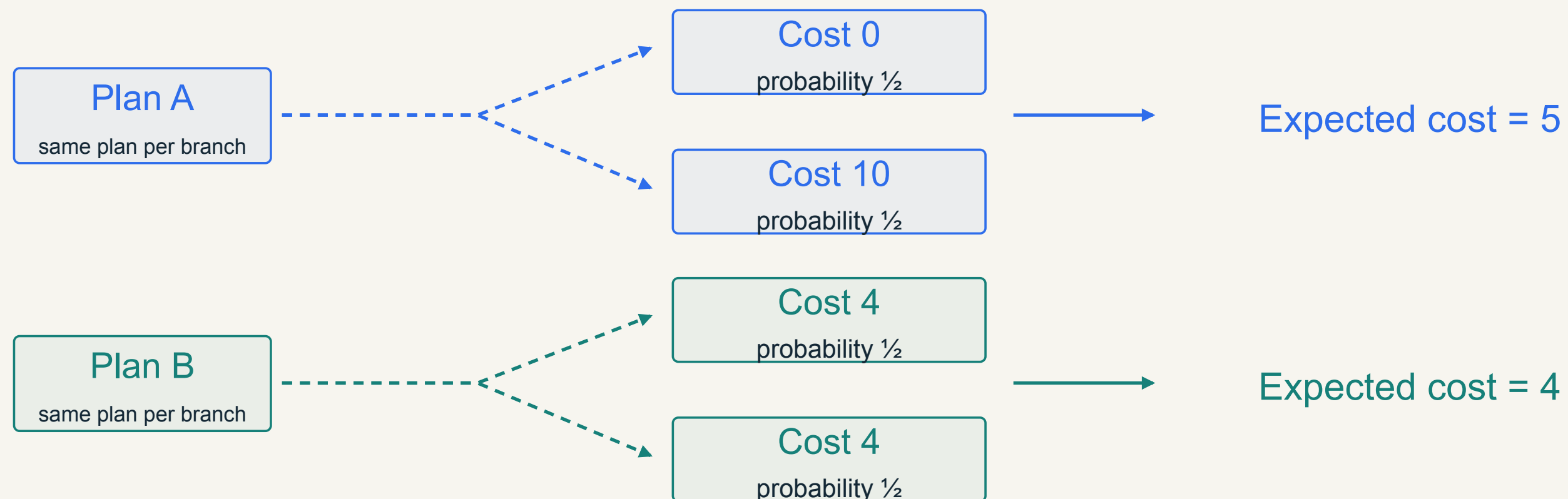
Local search. Flat regions and local minima can stall an update.

Long rollouts. Gradients can vanish or explode across many transitions.

Model error. A model's action gradient can point in the wrong direction.

Possible responses: restarts, shorter horizons with feedback, and better model coverage.

One plan, several futures: compare expected costs



$$\mathbb{E}[J(S)] \neq J(\mathbb{E}[S]) \quad \text{in general}$$

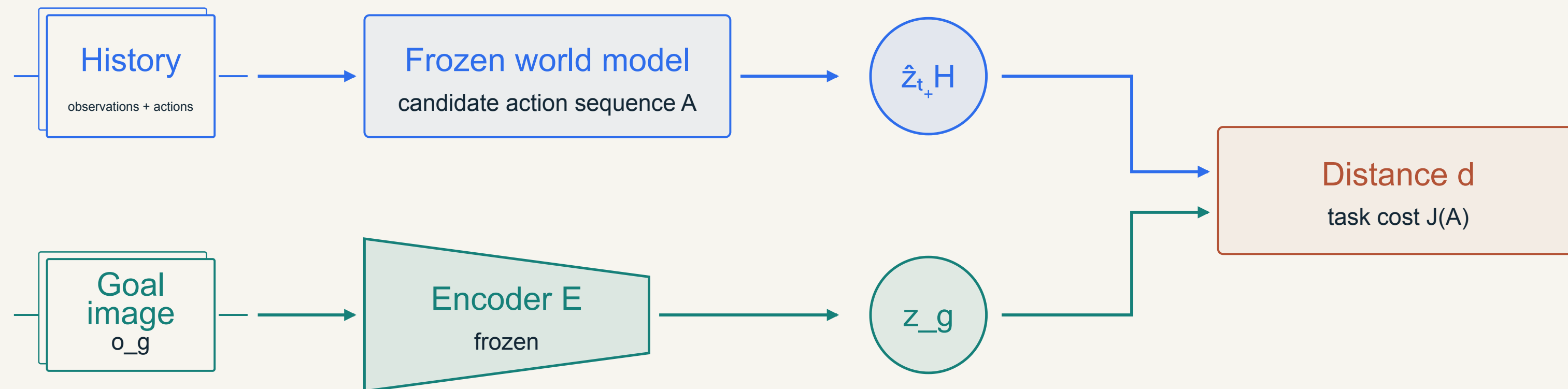
Sampling candidate actions and sampling the futures of one plan are different operations.

CEM, MPPI, and gradient-based planning

Method	How it updates a plan	Needs action gradients?
CEM	Refit the proposal to elites	No
MPPI	Average cost-weighted perturbations	No
Gradient-based	Step along the negative cost gradient	Yes

All three optimize actions inside MPC. The model parameters stay fixed.

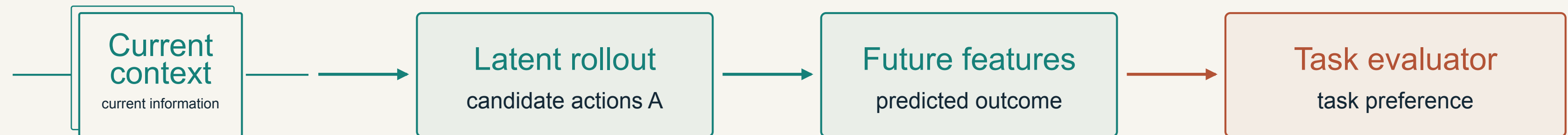
Turn a visual goal into a planning cost



$$J_{\text{goal}}(\mathbf{A}) = d(\hat{z}_{t+H}(\mathbf{A}), E(o_g)), \quad d(u, v) = \|u - v\|_2^2 \text{ is one choice}$$

The goal enters the evaluator. It need not enter the dynamics model.

A shared planning interface across model families



Check the interface

Ask before planning

Initialization

Which observations, memory, or inferred state does the model need?

Action search

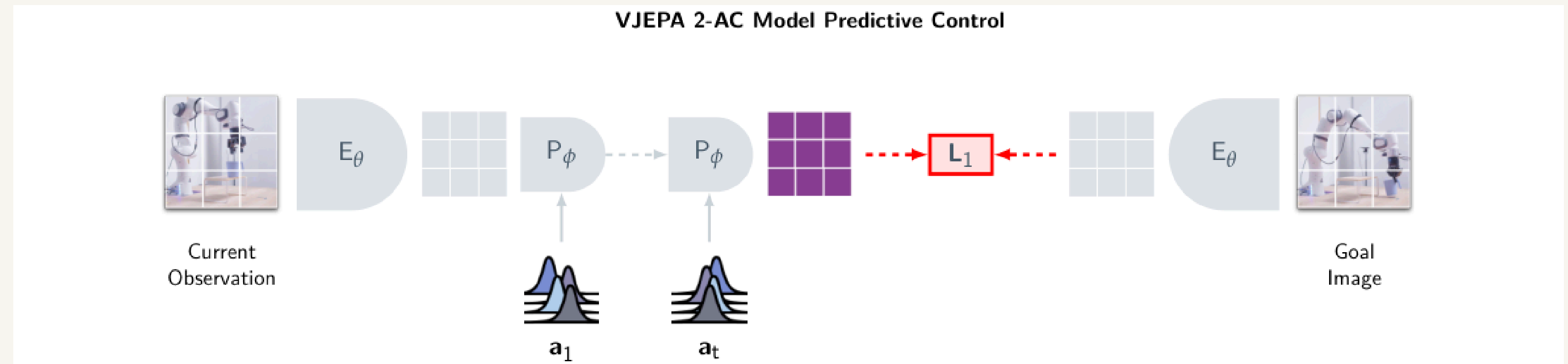
Are candidate actions valid? Are futures sampled or deterministic?

Evaluation and feedback

What score ranks plans? When does the next real observation arrive?

Initialize with enough history or inferred state to predict action consequences. Clone this context for every candidate.

V-JEPA 2-AC: planning inputs



V-JEPA 2, Figure 7 · action-conditioned visual prediction and goal comparison

Observed context

Visual features **and robot pose**.

Candidate input

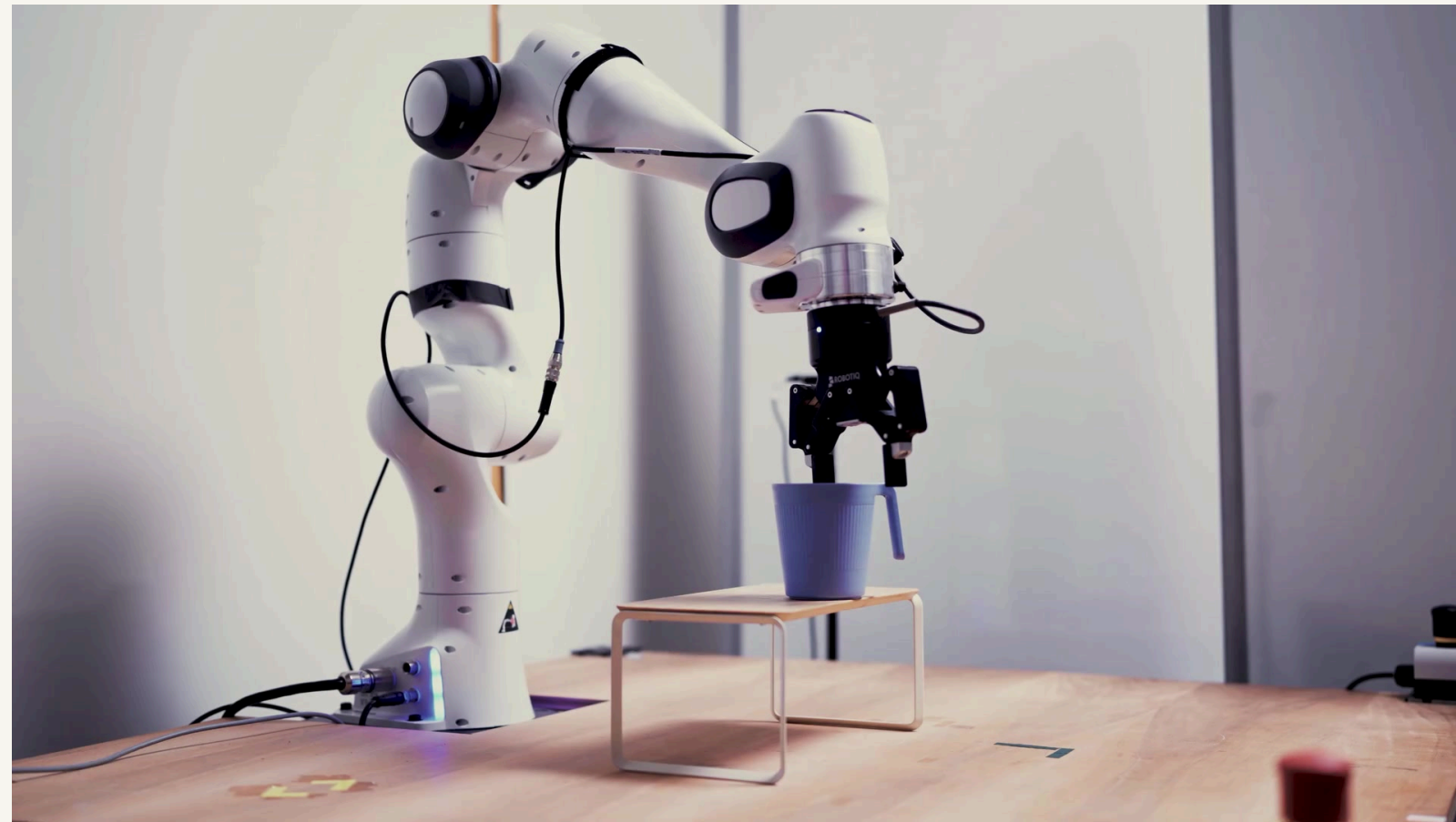
An action sequence for the frozen predictor.

Desired outcome

A goal image encoded in the same feature space.

The schematic above emphasizes visual features; the model also uses observed end-effector pose.

V-JEPA 2-AC: select and execute



Robot control · V-JEPA 2 project video

Published planning protocol

$$J(\mathbf{A}) = \|\hat{z}_{t+H}(\mathbf{A}) - E(o_g)\|_1$$

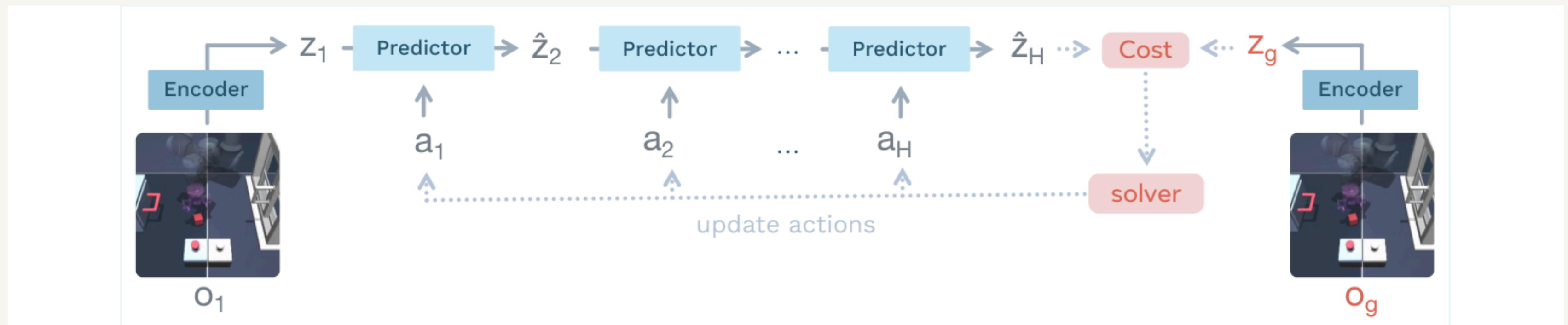
CEM searches action sequences.

Execute **one action** ($K = 1$).

Observe and replan.

Goal-feature distance is a task choice; viewpoint changes can affect it.

LeWorldModel: squared L2 distance to a visual goal



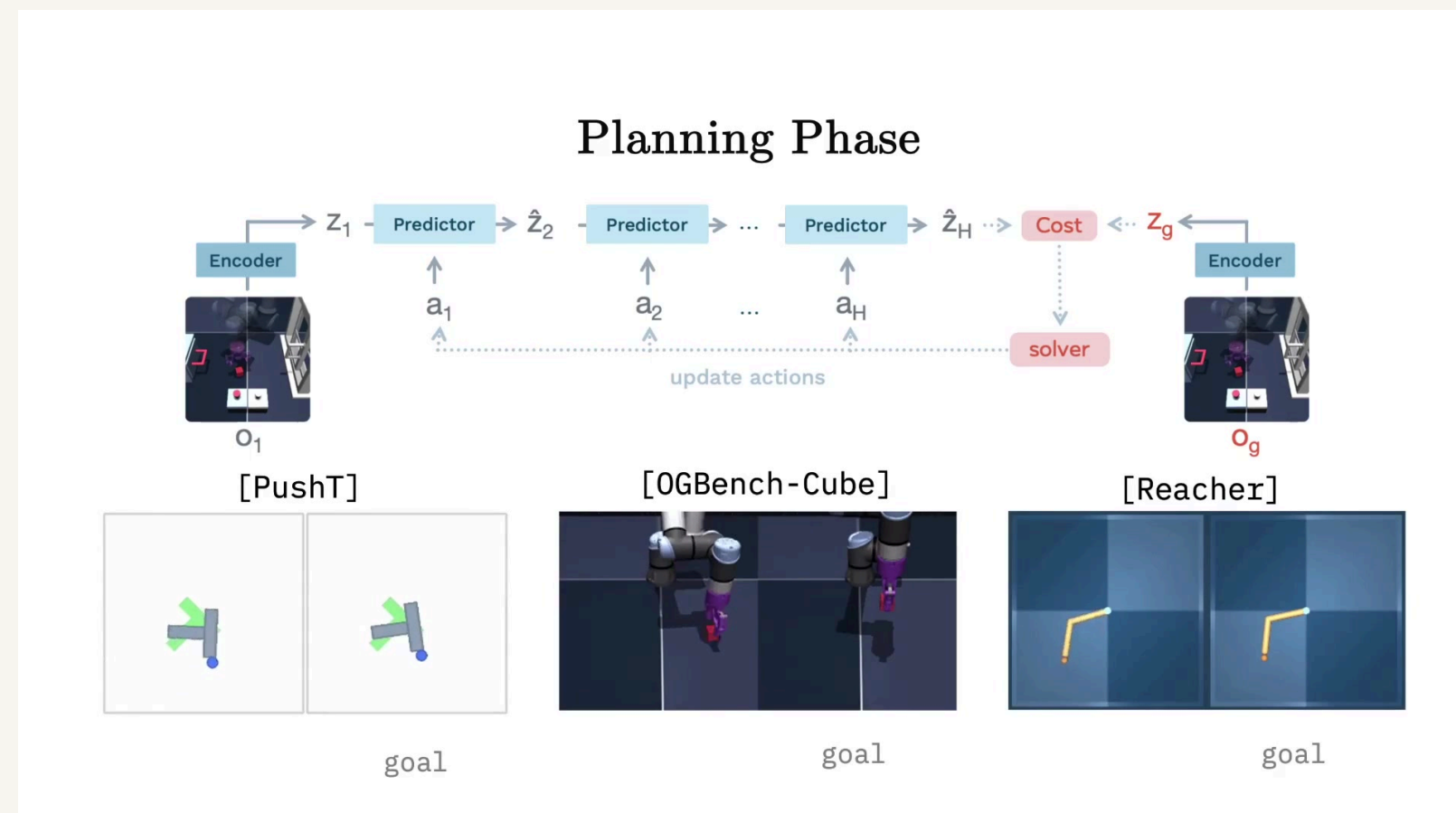
LeWorldModel, Figure 4 · goal-feature cost drives action search

$$J_{\text{LeWM}}(\mathbf{A}) = \|\hat{z}_{t+H}(\mathbf{A}) - z_g\|_2^2, \quad z_g = E(o_g)$$

$$\mathbf{A}^* \approx \arg \min_{\mathbf{A} \in \mathcal{A}^H} J_{\text{LeWM}}(\mathbf{A}) \quad \text{using CEM}$$

Compare the predicted endpoint with the goal. Keep encoder and predictor fixed.

LeWorldModel: when does feedback arrive?



LeWorldModel control · local video

Cited implementation

$$H = K = 5 \text{ model steps}$$

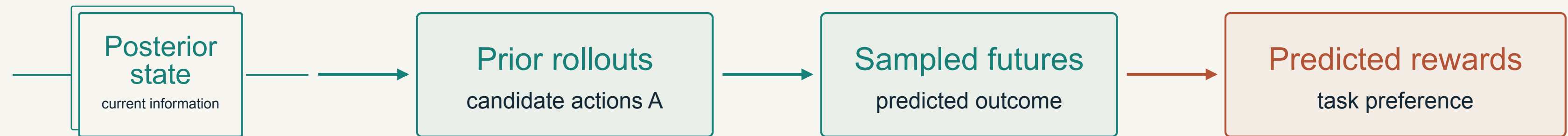
Execute the full planned chunk before replanning.

$$5 \text{ model steps} \times 5 \text{ frame skip} = 25 \text{ environment steps}$$

Appendix D. The general formulation allows executing a prefix.

Larger K means fewer planning calls—and a longer wait before correction.

RSSM: posterior initialization, prior rollouts



PlaNet · published CEM

Infer the current RSSM state from observations.

Imagine candidate futures with the prior; aggregate scores.

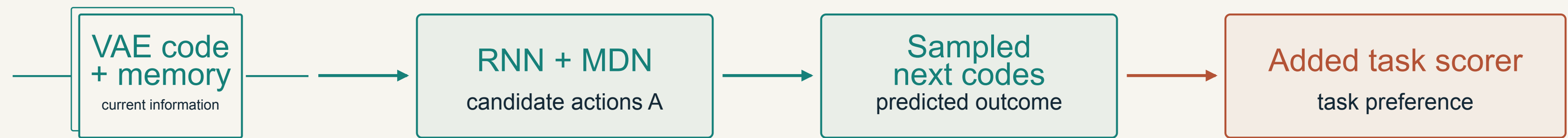
A new goal-distance scorer would be an added design choice, not PlaNet's original objective.

Dreamer · same model family

Learns an actor and a value function in imagination.

Its behavior learning comes in L10.

World Models (2018): memory inside each rollout



Expose the predictor to search

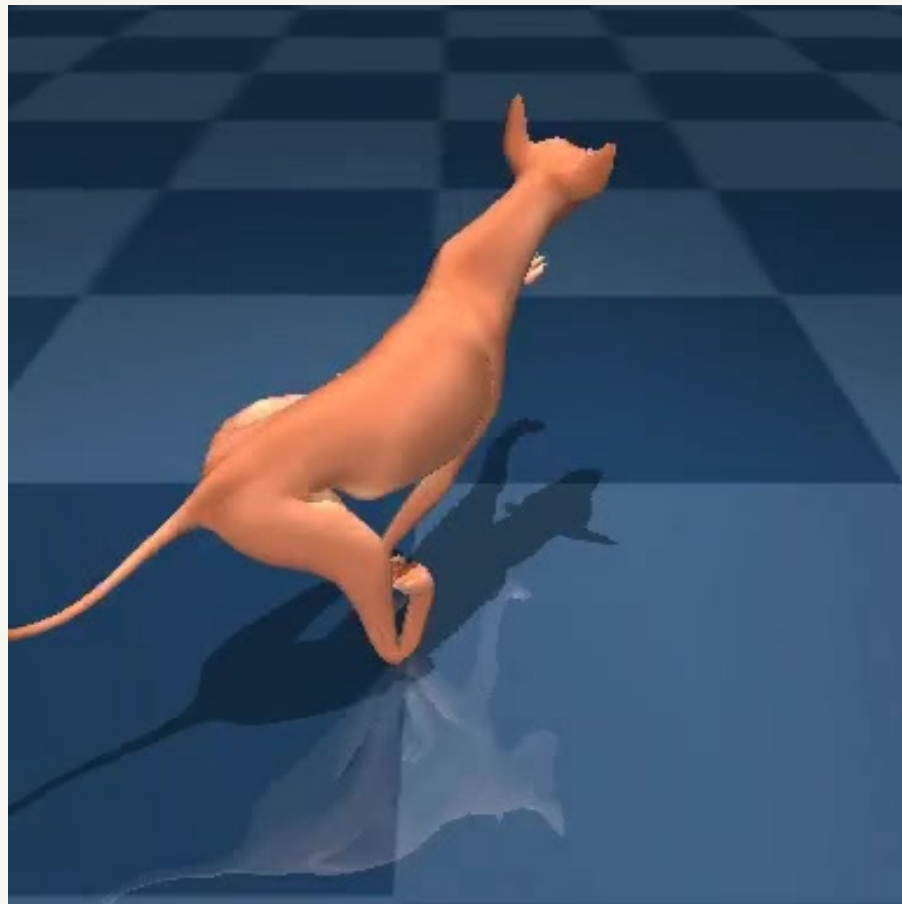
Each candidate carries its own recurrent memory.
Actions drive code prediction and memory updates.

Original World Models (2018)

A compact controller is optimized with CMA-ES.
Online CEM / MPPI here would be an extension.

The VAE code alone does not provide the whole predictive state.

TD-MPC: MPPI with a learned task evaluator



TD-MPC · model-based continuous control

Published MPPI-family planner

Roll out deterministic latent dynamics.

Use importance-weighted elite trajectories.

Include policy-assisted action proposals.

$$\text{trajectory score} = \sum_{j=0}^{H-1} \gamma^j \hat{r}_j + \gamma^H \hat{Q}_H$$

L10 modules: learned reward, terminal value, and policy. This score is maximized; cost would use its negative.

IRIS: discrete actions and token rollouts



IRIS · discrete observation tokens and Atari actions

Possible search extension

Append a valid action to token history.

Generate the next frame's codes; supply a task scorer.

Enumerate or sample categorical action sequences.

Original IRIS uses an actor-critic in imagination, not online lookahead search.

Gaussian perturbations should not be applied unchanged to discrete action IDs.

Showing success can be easier than specifying it



One successful trajectory: move a full cup to the target.

"This is what success looks like" can be easier than writing a complete objective.

A successful action sequence and its observed outcome convey the task together.

One demonstration, different objectives

The same successful cup demonstration is consistent with several objectives.

Candidate behavior	Match the final goal image	Arrive without spilling
Carry the full cup carefully	Accept	Accept
Spill, then arrive upright	May accept	Reject
Arrive safely at another acceptable placement	May reject	May accept

One example does not specify every constraint, acceptable variation, or tradeoff.

Constructed example. More demonstrations, comparisons, constraints, or feedback can supply missing task information.

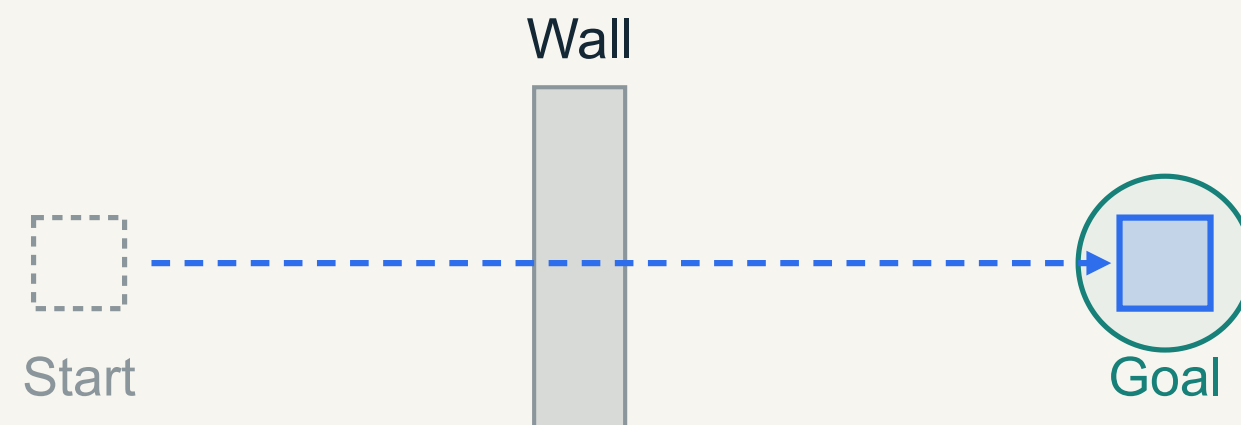
If rare failures matter, specify an acceptable failure probability as well as average cost.

World model error: predicted motion vs. reality

Task: move the box to the goal. **Cost:** final distance to the goal.

World model prediction

Same action: push the box right

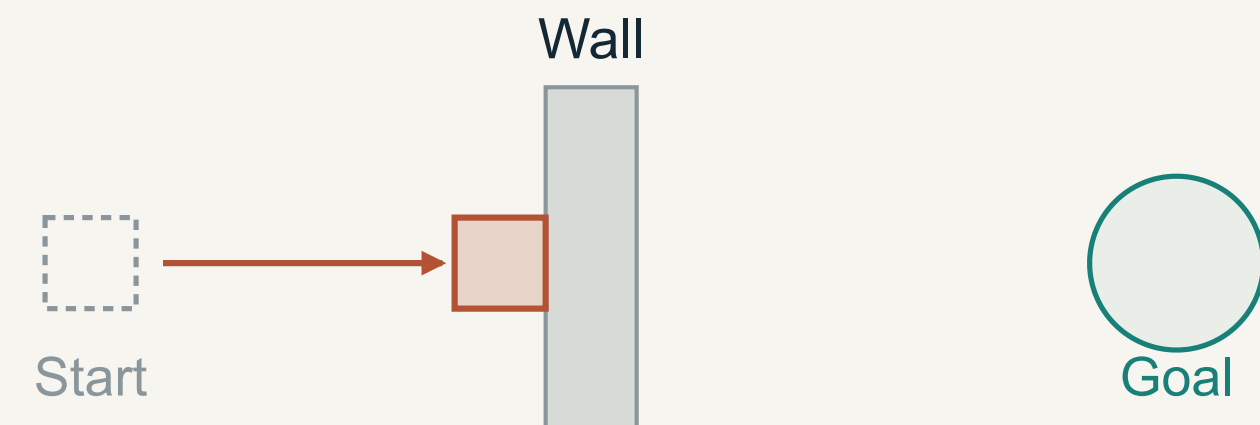


Passes through the wall and reaches the goal

Predicted distance to goal: near zero

Real execution

Same action: push the box right



Hits the wall and stops short of the goal

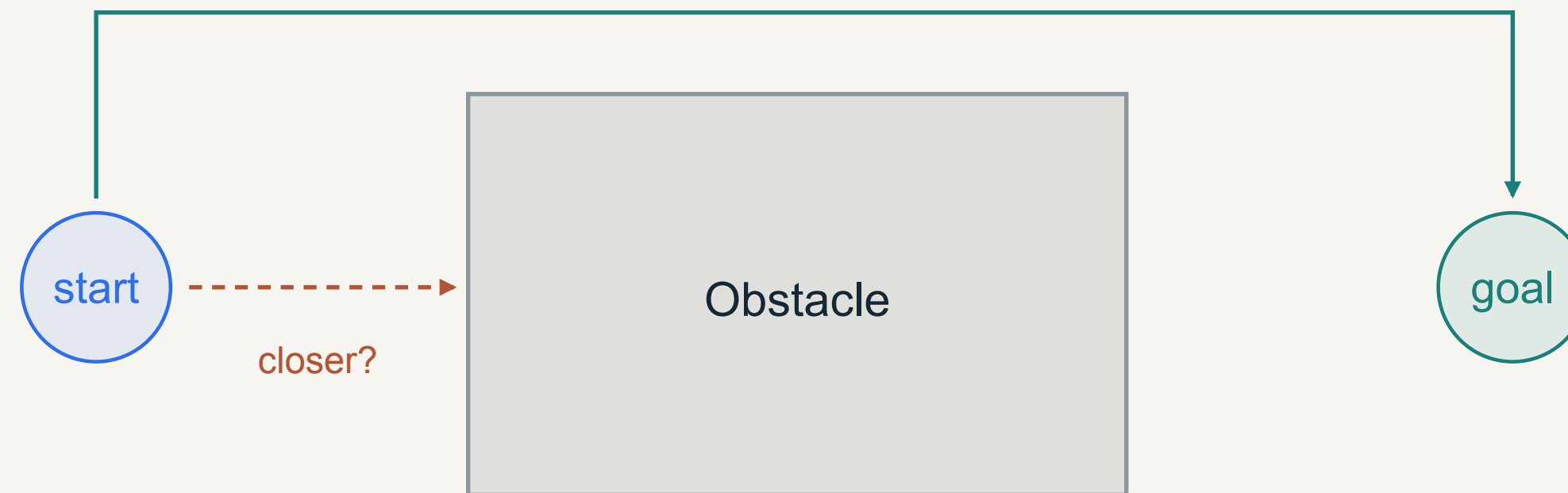
Actual distance to goal: large

The planner chooses this push because the world model predicts a low cost.

Same cost rule in both cases. The world model gets the motion wrong.

Possible responses: collect obstacle interactions and restrict poorly supported actions.

A correct model can still be hard to plan with



A useful first action may increase distance to the goal.

Predicting a plan

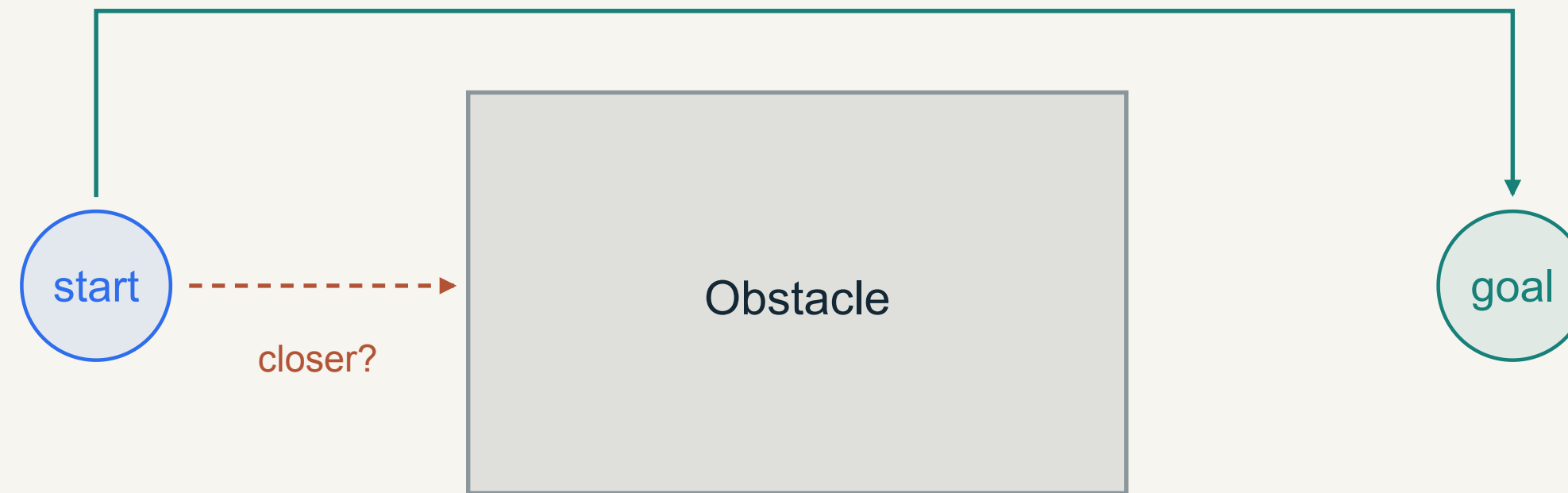
Evaluate one proposed sequence.

Restarts, structured proposals, or hierarchical search may help; none removes the difficulty in general.

Finding a good plan

Search a large, nonconvex action space.

Short horizons miss delayed benefits



A useful first action may increase distance to the goal.

A short horizon can prefer the dead end because the detour pays off later.

Longer H, subgoals, or temporal abstraction can help. Learned terminal values connect to L10.

Planning has a computation budget

$$\text{approximate rollout work} \propto I \times N \times M \times H$$

I

search rounds

N

candidate plans

M

futures per plan

Hmodel steps

Reduce work

Warm starts, useful proposals, batching, shorter rollouts.

Choose when to act

Larger action chunks save planning calls but delay feedback.

The best plan is of little use if it arrives after the decision deadline.

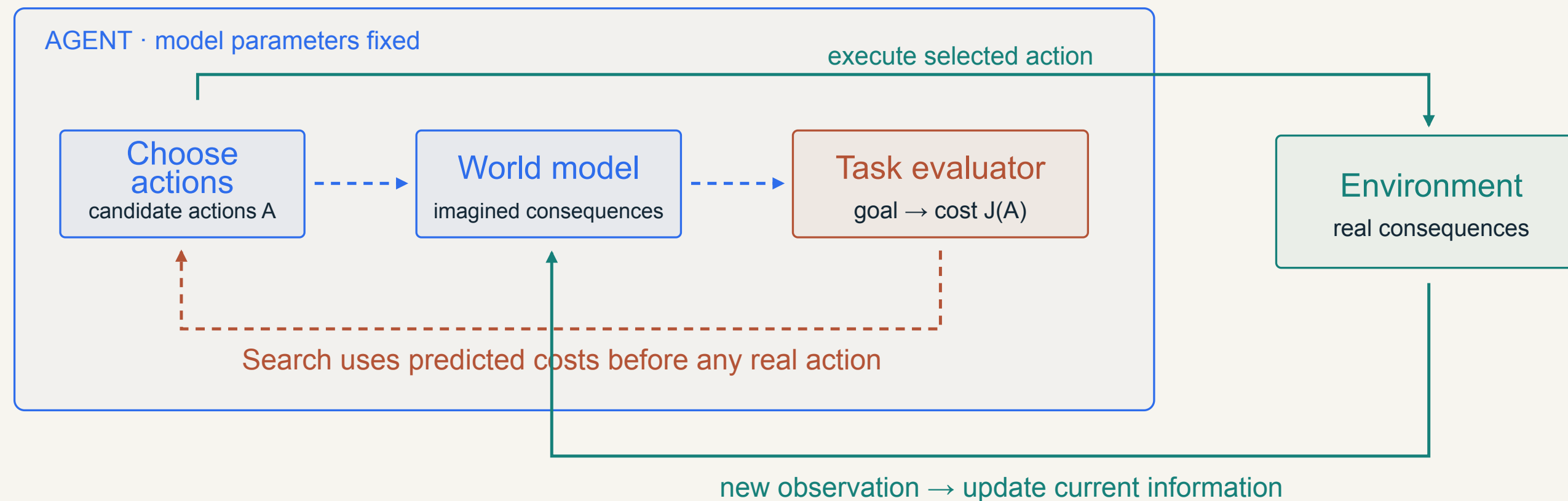
This counts model-transition work, not exact wall-clock latency. Autoregressive token generation adds work within a step.

Diagnose the failure in the closed loop

If the failure is...	A useful next question
Objective	Does the score actually represent success?
Information	Can the available observations distinguish relevant states?
Model	Do predicted consequences match actual outcomes?
Search	Can the optimizer find a good plan within this model?
Execution	Can we plan, act, and observe quickly enough?

Measure real task outcomes under stated compute and interaction budgets.

Using predictions to choose actions



Predict → evaluate → optimize actions → execute → observe.

Check: what changes during planning, and what stays fixed?

Learning behavior with world models

Today · online planning

Search over actions using a model and a task evaluator.

$$\mathcal{H}_t, g \xrightarrow{\text{search}} \mathbf{A}$$

L10 · learned behavior

Learn policies, values, and task-aware model components.

$$\pi(a \mid s), \quad V(s), \quad Q(s, a)$$

Can experience reduce the need to search from scratch?

Dreamer and TD-MPC: connect imagined futures to learning how to act.