GeoWorld: Geometric World Models

Supplementary Material

1. Preliminaries

1.1. Energy-Based World Models

Energy-Based World Models (EBWM) [2, 3, 5, 41] are derived from Energy-Based Models (EBM) [9, 26], which define a scalar energy function $F(x, y)$ that measures how compatible two variables are, such as a current world state s_x and a possible future state s_y. A low energy value corresponds to a plausible scenario, while a high energy value indicates an implausible one. Instead of predicting a single future, the energy landscape implicitly represents all plausible futures as valleys of low energy. Because energy replaces probability, the model can naturally handle multi-modal or uncertain worlds without the need for explicit sampling or normalization. Reasoning and planning are therefore formulated as energy minimization, where the goal is to find the configuration (actions, latents, or next states) that minimizes the expected energy:

$$\text{Plan} = \arg\min_{\text{actions}} F(s_t, s_{t+1:T}). \tag{23}$$

1.2. Hierarchical Planning

Hierarchical planning often consists of two levels: high-level planning computes trajectories in abstract latent space that minimize energy (i.e., the most plausible and least costly transitions), while lower levels fill in the physical details [41].

The world model learns hierarchical latent abstractions, where low-level modules predict short-term fine details and higher-level modules capture long-term abstract dynamics.

Specifically, planning becomes energy minimization in latent space. High-level modules operate on abstract latent states $s^{(2)}$ that evolve slowly, where planning corresponds to finding a geodesic of minimum energy between the abstract states $s_A^{(2)}$ and $s_B^{(2)}$, yielding a coarse trajectory that serves as the overall plan. Lower levels then refine this trajectory into fine-grained predictions $s_t^{(1)}$, minimizing sub-energies conditioned on the higher-level plan. As a result, the overall behavior emerges through hierarchical energy descent, where each layer enforces consistency between its predictions and the layer above. In other words, high-level planning computes trajectories in abstract latent space that minimize energy (i.e., the most plausible and least costly transitions), while lower levels fill in the physical details.

Formally, for a hierarchical world model $F^{(L)}$, the optimal plan is defined as:

$$\text{Optimal Plan:} \quad \min_{\{a_t, z_t\}} \sum_{l=1}^{L} \sum_{t} F^{(l)}\big(s_t^{(l)}, s_{t+1}^{(l)}, z_t^{(l)}\big). \tag{24}$$

Each $F^{(l)}$ expresses the energy cost of moving between abstract states at level l, and gradients through this hierarchy yield a coherent plan across scales.

1.3. Joint-Embedding Predictive Architecture

Joint-Embedding Predictive Architectures (JEPA) [41] learn a predictive world model directly in latent space rather than generating pixels. A JEPA encodes observations into a compact representation space and predicts future latent states by minimizing an energy or similarity objective between encoded targets and predicted embeddings. This joint-embedding formulation bypasses the need for autoregressive pixel generation, which is computationally expensive and prone to error accumulation over long horizons. Learning in latent space instead focuses the model on high-level structure, semantics, and temporal dependencies rather than low-level appearance details, enabling more stable and efficient multi-step prediction. By operating on representations rather than images, JEPA captures the underlying dynamics of the environment while avoiding the challenges of modeling raw pixel distributions.

1.3.1. JEPA

We define the current observation x and target y. $E_\theta(\cdot)$ denotes the observation encoder that maps raw visual inputs into the latent representation space, $\bar{E}_\theta(\cdot)$ denotes the target encoder with exponential moving average (EMA) weights, s_x and s_y are the latent representation of the current and target states obtained from the encoders, and z is a latent variable capturing uncertainty. $P_\phi(\cdot)$ denotes the predictor, and θ and ϕ denote the parameters (weights) of the encoder and predictor networks, respectively.

For a unified one-step observation–target formulation, we first encode the current observation:

$$s_x = E_\theta(x) \tag{25}$$

Then we perform latent prediction, which takes the current state s_x and the uncertainty z as input and predicts the latent representation of the target state $\hat{s}_y$:

$$\hat{s}_y = P_\phi(s_x, z) \tag{26}$$

Similarly, we encode the target representation:

$$s_y = \bar{E}_\theta(y) \tag{27}$$

For planning, the training objective becomes an energy minimization in latent space, which involves finding a geodesic of minimum energy between s_x and s_y to obtain the optimal plan.

$$\min_z \; C(s_x, s_y, z), \quad \text{where } s_x = E_\theta(x),\; s_y = \bar{E}_\theta(y). \tag{28}$$

Here, C represents the energy cost of moving between the abstract states s_x and s_y. It learns two encoders (for past and future) and a predictor that maps s_x to s_y, where the energy is defined as the representation mismatch between the predicted and true embeddings. The JEPA is non-generative and is trained non-contrastively to ensure that the embeddings remain both informative and predictable [4]. As a result, it forms a predictive world model that learns latent abstractions.

Hierarchical-JEPA If we extend to hierarchical planning, we can develop a model that learns hierarchical latent abstractions (Hierarchical-JEPA), in which low-level modules predict short-term fine details, while higher-level modules capture long-term abstract dynamics. As an example, consider a two-level model where high-level JEPA layers operate on abstract latent states $s^{(2)}$ that evolve slowly. Planning in this context corresponds to finding a geodesic of minimum energy between the abstract states $s_x^{(2)}$ and $s_y^{(2)}$, yielding a coarse trajectory, i.e., the plan. Meanwhile, lower levels refine this trajectory into fine-grained predictions $s_t^{(1)}$, minimizing sub-energies conditioned on the higher-level plan. Overall behavior emerges through hierarchical energy descent, where each layer enforces consistency between its predictions and the layer above.

In other words, high-level planning computes trajectories in abstract latent space that minimize energy (i.e., most plausible and least costly transitions), while lower levels fill in the physical details.

Formally, the training objective of a Hierarchical-JEPA is given by:

$$\min_{\{z_t\}} \sum_{l=1}^{L} \sum_t C^{(l)}\big(s_t^{(l)}, s_{t+1}^{(l)}, z_t^{(l)}\big), \tag{29}$$

where each $C^{(l)}$ represents the energy cost of moving between abstract states at level l, and gradients through this hierarchy yield a coherent plan across scales.

1.3.2. I-JEPA

Image-JEPA (I-JEPA) [2] extends JEPA to learn semantic image representations by predicting latent features of masked image regions from visible context patches.

In I-JEPA, an image y is divided into non-overlapping patches, from which a single large block is sampled as the context and several smaller blocks are sampled as targets. The context block x is fed to the context encoder $E_\theta(\cdot)$ to obtain patch-level latent representations s_x, while the target blocks are processed by a target encoder with EMA weights, $\bar{E}_\theta(\cdot)$, to produce target embeddings s_y. The predictor network $P_\phi(\cdot)$ takes the context representation s_x along with positional mask tokens $\{m_j\}_{j\in B_i}$ indicating the spatial locations of each target block, and predicts the corresponding feature vectors $\hat{s}_y = P_\phi(s_x, \{m_j\}_{j\in B_i})$ for those regions.

The training objective minimizes the average squared distance between the predicted and target representations of the target blocks, averaged over all sampled blocks and training samples in the dataset $\mathcal{D}$:

$$\min_{\theta,\phi} \; \mathbb{E}_{(x,y)\sim\mathcal{D}} \left[\frac{1}{M} \sum_{i=1}^{M} \sum_{j\in B_i} \Big\| P_\phi\big(E_\theta(x), \{m_j\}_{j\in B_i}\big) - \bar{E}_\theta(y)_j \Big\|_2^2 \right]. \tag{30}$$

Here, M denotes the number of target blocks, $\mathbb{E}[\cdot]$ denotes the expectation (average) over all training samples (x, y), and the loss measures the representation-level prediction error rather than pixel-level reconstruction, allowing I-JEPA to learn highly semantic, non-generative representations.

1.3.3. V-JEPA

Video-JEPA (V-JEPA) [5] and V-JEPA 2 [3] extend JEPA to learn spatio-temporal video representations by predicting masked tubelet features from visible context regions.

In V-JEPA and V-JEPA 2, a video clip y is tokenized into spatial–temporal patches (tubelets), and a subset of these patches is masked (replaced with mask tokens). The remaining unmasked patches form the context view, while the masked patches form the target view. The observation (context) encoder $E_\theta(\cdot)$ processes the masked version x (containing both visible and mask tokens) and produces latent embeddings for all positions, including the masked ones. However, the predictor $P_\phi(\cdot)$ focuses only on the masked positions. A target encoder with EMA weights, $\bar{E}_\theta(\cdot)$, computes the target embeddings s_y of the full (unmasked) input. The predictor takes the context representation $s_x = E_\theta(x)$ along with the mask token indicators Δ_y and predicts the feature vectors $\hat{s}_y$ for each masked patch position.

The training objective is to minimize the L_1 distance between the predicted representations of the masked regions and the target representations from the target encoder, averaged over the dataset $\mathcal{D}$:

$$\min_{\theta,\phi} \mathbb{E}_{(x,\Delta_y,y)\sim\mathcal{D}} \left\| P_\phi\big(E_\theta(x), \Delta_y\big) - \bar{E}_\theta(y) \right\|_1 \tag{31}$$

where $\mathbb{E}[\cdot]$ denotes the expectation (average) over all training samples (x, Δ_y, y).

That is,

$$\min_{\theta,\phi} \mathbb{E}_{(s_x,\Delta_y,s_y)\sim\mathcal{D}} \left[\|\hat{s}_y - s_y\|_1 \right], \text{ where } \hat{s}_y = P_\phi(s_x, \Delta_y). \tag{32}$$

Hence the loss function $\mathcal{L}$ measuring the L_1 distance between the $\hat{s}_y$ and s_y:

$$\mathcal{L}(\theta,\phi) = \left\| P_\phi\big(E_\theta(x), \Delta_y\big) - \bar{E}_\theta(y) \right\|_1 \tag{33}$$

V-JEPA 2-AC The action-conditioned variation (V-JEPA 2-AC) serves as a downstream extension of V-JEPA 2 that predicts future latent representations conditioned on agent actions.

As a step-by-step (1-step prediction) formulation, V-JEPA 2-AC adapts the frozen encoder $E(\cdot)$ from V-JEPA 2, pretrained on unlabeled videos, to encode the current observation x_t into the latent representation s_t^x and the future target x_{t+1} into s_{t+1}^x. The action-conditioned predictor $P_\phi(\cdot)$ takes the current latent s_t^x and the action a_t as input to predict the next-state latent representation $\hat{s}_{t+1}^x$. The L_1 loss in latent space then trains the predictor to align its predicted next-state representations with the encoded future representations:

$$\mathcal{L}_{\text{AC}}(\phi) = \left\| P_\phi\big(E(x_t), a_t\big) - E(x_{t+1}) \right\|_1 \tag{34}$$

$$= \left\| \hat{s}_{t+1}^x - s_{t+1}^x \right\|_1. \tag{35}$$

In the multi-step rollout setting, V-JEPA 2-AC extends the one-step formulation to predict a sequence of future latent representations over a time horizon T conditioned on a sequence of actions.

We randomly sample a mini-batch of 4-second video clips from the Droid dataset [39] and, for simplicity, discard any videos shorter than 4 seconds, leaving us with a smaller subset of the dataset comprising under 62 hours of video. The video clips are sampled with a resolution of 256×256 and a frame rate of 4 fps, yielding 16-frame clips $(x_t)_{t=1}^{16}$, where each x_t represents a single video frame. The robot's end-effector state in each observation is denoted by the sequence $(s_t^e)_{t=1}^{16}$, where s_t^e is a real-valued 7D vector defined relative to the base of the robot. We construct a sequence of actions $(a_t)_{t=1}^{15}$ by computing the change in end-effector state between adjacent frames. We use the V-JEPA 2 encoder $E(\cdot)$ as an image encoder and encode each frame independently in a given clip to obtain a sequence of feature maps $(s_t^x)_{t=1}^{16}$. The sequence of observed feature maps, end-effector states, and actions is temporally interleaved as $(s_t^x, s_t^e, a_t)_{t=1}^{15}$ and processed with the transformer predictor network $P_\phi(\cdot)$ to obtain a sequence of next-state representation predictions $(s_{t+1}^x)_{t=1}^{15}$:

$$(\hat{s}_{t+1}^x)_{t=1}^{15} = P_\phi\big((s_t^x, s_t^e, a_t)_{t=1}^{15}\big). \tag{36}$$

The teacher-forcing loss trains the predictor $P_\phi(\cdot)$ to accurately perform one-step future prediction by minimizing the L_1 distance between the predicted latent representation $\hat{s}_{t+1}^x$ and the encoded ground-truth latent s_{t+1}^x at each time step t:

$$\mathcal{L}_{\text{TF}}(\phi) = \frac{1}{T}\sum_{t=1}^{T} \left\| P_\phi(E(x_t), s_t^e, a_t) - E(x_{t+1}) \right\|_1 \tag{37}$$

$$= \frac{1}{T}\sum_{t=1}^{T} \left\| \hat{s}_{t+1}^x - s_{t+1}^x \right\|_1, \text{ where } T = 15. \tag{38}$$

The rollout loss involves feeding the predictor's output back as input, allowing the model to be trained to predict several timesteps ahead. In this case, we design a two-step rollout loss to improve the model's ability to perform autoregressive rollouts during inference. We can now denote the rollout loss as

$$\mathcal{L}_{\text{rollout}}(\phi) = \left\| P_\phi(E(x_t), s_t^e, (a_t)_t^{t+n}) - E(x_{t+n+1}) \right\|_1, \tag{39}$$

$$= \left\| P_\phi(s_t^x, s_t^e, a_t, a_{t+1}) - s_{t+2}^x) \right\|_1, \tag{40}$$

$$= \left\| \hat{s}_{t+2}^x - s_{t+2}^x \right\|_1, \text{ where } n = 1. \tag{41}$$

Hence, the total loss is

$$\mathcal{L}_{\text{AC}}(\phi) = \mathcal{L}_{\text{TF}}(\phi) + \mathcal{L}_{\text{rollout}}(\phi). \tag{42}$$

Inference We can then perform energy-based planning after training, using the frozen encoder E and predictor P. This predictor acts as the world model, capable of imagining how latent representations evolve when an action sequence is applied. At test time, no weights are trained, instead, we search for an action sequence that minimizes a goal-conditioned energy cost between the imagined future and the goal latent representation.

Given the current observation x_1, current end-effector state s_1^e, target goal image x_{1+T}, and planning horizon T, we encode the current and goal observations as:

$$s_1^x = E(x_1), \qquad s_{1+T}^x = E(x_{1+T}). \tag{43}$$

We then define the L_1 energy cost function:

$$C((\hat{a}_t)_{t=1}^T; s_1^e, s_1^x, s_{1+T}^x) = \left\| P((\hat{a}_t)_{t=1}^T; s_1^e, s_1^x) - s_{1+T}^x \right\|_1. \tag{44}$$

The optimal action sequence is obtained by minimizing this cost:

$$(a_t^*)_{t=1}^T = \arg\min_{(\hat{a}_t)_{t=1}^T} C((\hat{a}_t)_{t=1}^T; s_1^e, s_1^x, s_{1+T}^x). \quad (45)$$

Thus, the predictor is used to imagine the future latent trajectory, and planning reduces to finding the action sequence that minimizes the latent L1 distance to the goal embedding.

V-JEPA 2-AC uses the Cross-Entropy Method (CEM) [23] with 800 samples and 10 iterations to efficiently minimize C at each planning step, as shown in Algorithm 1. It executes only the first action on the robot before re-planning, as in receding horizon control, and is tested only on a horizon of $T = 1$.

In other words, during inference, the predictor P serves as a world model for *energy-based planning*, where at each step the CEM searches for an action sequence that minimizes the latent-space cost C between the imagined future and the goal representation, executing only the first action a_1^* before re-planning.

1.4. World Modeling Paradigms

There are two typical approaches for goal-conditional world modeling: generative world models [46, 59, 63] and predictive world models [2, 3, 5, 29, 41].

Generative world models. Generative world models typically build upon autoregressive (AR) transformers or semi-AR (autoregressive diffusion) models ρ that observe the visual context x_t and autoregressively predict the latent z_t and the next frame $\hat{x}_{t+1}$:

$$(z_t, \hat{x}_{t+1}) = \rho(x_t). \quad (46)$$

They often rely on an inverse dynamics model (IDM) π [67], trained separately, which maps the pair $(x_t, z_t, \hat{x}_{t+1})$ to an explicit action [10, 73, 80]:

$$a_t = \pi(x_t, z_t, \hat{x}_{t+1}). \quad (47)$$

In other words, it's a one-step inverse mapping of the environment's dynamics. Hence, it can only predict one step at a time, because it does not know the full trajectory structure or the energy landscape over multiple steps.

Moreover, it must explicitly generate or reconstruct the next frame (or latent visual tokens that decode into pixels), and perform planning by predicting how the world will look after an implicit action, which connects to the challenge mentioned in 1.3.

Predictive world models Predictive world models are not generative, they do not model pixel distributions. Instead, they learn an energy landscape in latent space that measures compatibility between current and target states.

However, predictive world models require an explicit goal observation x_{t+1} to compute their goal-conditioned

Algorithm 1 Energy-Based Planning with Cross-Entropy Method (CEM)

Require: Predictor P (world model), encoder $E(\cdot)$, planning horizon T, number of samples N, number of elites K, number of iterations I, initial mean μ_0, and covariance Σ_0.

Ensure: Optimal action sequence $(a_t^*)_{t=1}^T$.

Input: current observation x_1, end-effector state s_1^e, and goal image x_{1+T}.

Encode current and goal observations: $s_1^x = E(x_1)$, $s_{1+T}^x = E(x_{1+T})$.

for $j = 1$ **to** I **do**

(1) Sample: Draw N candidate action sequences $\{\hat{a}_{1:T}^{(n)}\}_{n=1}^N$ from $\mathcal{N}(\mu_{j-1}, \Sigma_{j-1})$.

(2) Evaluate: For each candidate sequence, compute its energy cost:

$$C^{(n)} = \left\| P_\phi(\hat{a}_{1:T}^{(n)}; s_1^e, s_1^x) - s_{1+T}^x \right\|_1.$$

(3) Select elites: Sort $\{C^{(n)}\}$ in ascending order and select the top K sequences with the lowest cost to form the elite set $\mathcal{E}_j = \{\hat{a}_{1:T}^{(n)} \mid n \in \text{top-}K(C)\}$. These elites represent trajectories that drive the world model's imagined latent state closest to the goal latent s_{1+T}^x.

(4) Update distribution: Compute the new mean and covariance of the elite set:

$$\mu_j = \frac{1}{K} \sum_{(\hat{a}_t)_{t=1}^T \in \mathcal{E}_j} (\hat{a}_t)_{t=1}^T,$$

$$\Sigma_j = \frac{1}{K} \sum_{(\hat{a}_t)_{t=1}^T \in \mathcal{E}_j} \left((\hat{a}_t)_{t=1}^T - \mu_j\right)\left((\hat{a}_t)_{t=1}^T - \mu_j\right)^\top.$$

The new distribution $\mathcal{N}(\mu_j, \Sigma_j)$ is now centered around promising low-energy action sequences.

(5) Repeat: Continue iterating Steps 3–4 for I iterations. As the process proceeds, the sampling distribution progressively concentrates around action sequences that minimize the latent-space cost C_{L1}.

end for

(6) Execute: Select the action sequence corresponding to the lowest final cost:

$$(a_t^*)_{t=1}^T = \arg\min_{(\hat{a}_t)_{t=1}^T} C((\hat{a}_t)_{t=1}^T; s_1^e, s_1^x, s_{1+T}^x).$$

and execute only the first action a_1^* on the robot.

(7) Re-plan: Observe the next frame x_2, re-encode $s_2^x = E(x_2)$, and repeat the process from Step 1 (receding horizon control).

energy, as shown in Equation 44, and minimize the energy cost with the CEM for planning.

This design trade-off is intentional, not accidental. As mentioned in 1.3, we need to avoid pixel prediction during planning since pixels are noisy, unimportant, and computationally expensive [41]. Therefore, predictive world models are not intended to be self-contained simulators. Moreover, unlike the IDM, which predicts only a single action given consecutive states, CEM performs multi-step trajectory optimization by searching over candidate action sequences to minimize the latent-space energy cost, enabling long-horizon planning rather than one-step reactive control.

1.5. Hyperbolic Learning

Hyperbolic space, denoted as $\mathbb{H}^n$, is a negatively curved Riemannian manifold characterized by exponential volume growth and a saddle-shaped geometry [11]. Unlike Euclidean space, where parallel lines remain equidistant, lines in hyperbolic space diverge, and the volume expands exponentially with radius. This property makes hyperbolic geometry naturally suited for representing hierarchical or tree-like data structures [51, 55], such as hierarchical planning for world models, where the number of nodes grows exponentially with depth. In deep learning, hyperbolic space enables exponentially efficient representations of hierarchies by compressing large-scale differences while maintaining fine-grained local relationships. As a result, it has been widely applied in representation learning [25, 51, 55, 83], computer vision and graphics [30] that require modeling multi-level, non-Euclidean structures.

Hyperbolic space $\mathbb{H}^n$ is an abstract Riemannian manifold of constant negative curvature that does not depend on any coordinate system. In order to represent points in this curved space for computation, coordinate models are introduced to map $\mathbb{H}^n$ into Euclidean space while preserving its geometric structure. Two of the most common models are the Poincaré ball model $\mathbb{B}^n$ [13, 30, 51, 83] and the Lorentz (or hyperboloid) model $\mathbb{L}^n$ [25, 55], both providing isometric representations of the same manifold but differing in their coordinate systems and numerical properties.

1.5.1. Poincaré Ball Model.

The Poincaré ball model $\mathbb{B}^n$ represents hyperbolic space as an open unit ball embedded in Euclidean space, defined as

$$\mathbb{B}^n = \{z \in \mathbb{R}^n : \|z\| < 1\}. \tag{48}$$

It is endowed with a Riemannian metric that encodes constant negative curvature, ensuring that Euclidean distances are reweighted to reflect hyperbolic geometry. Each point z lies strictly inside the ball, and distances are measured using the hyperbolic metric rather than Euclidean norms. This bounded representation makes the geometry intuitive and well-suited for visualization, as tree-like hierarchies naturally fit inside a finite domain where the boundary corresponds to infinite distance. It constrains embeddings to normalized radii, but note that in the Poincaré model "normalization" means keeping points *inside* the unit ball, not unit-norm on a sphere (that corresponds to the hyperspherical case).

Geodesics The geodesic, or the Poincaré-ball hyperbolic distance, between two points $u, v \in \mathbb{B}^n$ is a circular arc perpendicular to the boundary of the ball. Its length is given by the hyperbolic distance function [28, 51]

$$d_{\mathbb{H}}(u, v) = \operatorname{arcosh}\left(1 + 2\frac{\|u - v\|^2}{(1 - \|u\|^2)(1 - \|v\|^2)}\right). \tag{49}$$

This metric measures the shortest path along the curved manifold rather than in Euclidean space, capturing the exponential growth of distances as points approach the boundary of the Poincaré ball.

Exponential Map In Riemannian geometry, the exponential map

$$\exp_x : \mathbf{T}_x\mathbb{H}^n \to \mathbb{H}^n \tag{50}$$

takes a tangent vector $v \in \mathbf{T}_x\mathbb{H}^n$ (the tangent space at point x) and moves it along the geodesic starting from x in direction v, traveling a distance equal to $\|v\|$ under the hyperbolic metric. In other words, $\exp_x(v)$ can be interpreted as starting at x and walking along the manifold in the direction of v for a distance $\|v\|$. This operation maps local Euclidean updates v into global manifold coordinates, ensuring that updates remain consistent with the geometry of hyperbolic space. In hyperbolic space, the mapping from the tangent space to the manifold through the exponential map is one-to-one. Although there exist manifolds equipped with hyperbolic metrics where this mapping is not one-to-one [85], the Poincaré model of hyperbolic space preserves this one-to-one correspondence, ensuring a well-defined relationship between the tangent space and the manifold.

The exponential map from the origin of the Poincaré ball, denoted as $\exp_0(v)$, maps Euclidean vectors directly into hyperbolic space, which is particularly useful for initialization. It is defined as

$$\exp_0(v) = \tanh(\|v\|)\frac{v}{\|v\|}. \tag{51}$$

This formulation is simple because the tangent space at the origin aligns perfectly with the Euclidean space, making the mapping between Euclidean and hyperbolic representations straightforward.

The exponential map from a general point $x \in \mathbb{B}^n$ must account for the curvature around x. It moves the point x

along the geodesic in the direction of the tangent vector v. The key difference from the origin case is that x is not the origin, so we need a way to "add" v to x under hyperbolic geometry. In Euclidean space, moving from a point x by a vector v is simply computed as $x + v$. However, in hyperbolic geometry, vector addition is replaced by *Möbius addition*, denoted as $x \oplus v$. Therefore, the general exponential map is expressed as

$$\exp_x(v) = x \oplus \left(\tanh\left(\frac{\lambda_x \|v\|}{2} \right) \frac{v}{\|v\|} \right), \tag{52}$$

where

$$\lambda_x = \frac{2}{1 - \|x\|^2} \tag{53}$$

is the *conformal factor* that rescales distances locally. This formulation ensures that the operation respects hyperbolic curvature instead of Euclidean linearity.

The Möbius addition of two vectors x and y is defined as

$$x \oplus y = \frac{(1 + 2\langle x, y \rangle + \|y\|^2)x + (1 - \|x\|^2)y}{1 + 2\langle x, y \rangle + \|x\|^2\|y\|^2}. \tag{54}$$

Substituting $y = \tanh\left(\frac{\lambda_x \|v\|}{2} \right) \frac{v}{\|v\|}$, we obtain an explicit expression for $\exp_x(v)$ in hyperbolic space

$$\exp_x(v) = \frac{\left(1 + 2\alpha\langle x, \frac{v}{\|v\|} \rangle + \alpha^2\right)x + (1 - \|x\|^2)\alpha \frac{v}{\|v\|}}{1 + 2\alpha\langle x, \frac{v}{\|v\|} \rangle + \alpha^2\|x\|^2}, \tag{55}$$

where

$$\alpha = \tanh\left(\frac{\lambda_x \|v\|}{2} \right), \quad \lambda_x = \frac{2}{1 - \|x\|^2}. \tag{56}$$

This formulation explicitly shows how the exponential map combines the curvature-adjusted scaling (via λ_x and $\tanh$) with the non-linear composition of x and v under Möbius addition, ensuring consistency with hyperbolic geometry.

Logarithmic Map The logarithmic map serves as the inverse of the exponential map, mapping points from the manifold back to the tangent space at a given point $x \in \mathbb{B}^n$. Formally, it is defined as

$$\log_x : \mathbb{B}^n \to \mathbf{T}_x\mathbb{H}^n, \tag{57}$$

which takes a point $y \in \mathbb{B}^n$ and returns a tangent vector $v \in \mathbf{T}_x\mathbb{H}^n$ that, when re-projected through the exponential map, satisfies $\exp_x(v) = y$. This operation locally linearizes the manifold around x, allowing differential computations such as gradient-based optimization to be performed in the tangent space.

The logarithmic map from the origin of the Poincaré ball, denoted as $\log_0(y)$, converts a point $y \in \mathbb{B}^n$ back to its Euclidean tangent vector and is defined as

$$\log_0(y) = \operatorname{arctanh}(\|y\|) \frac{y}{\|y\|}. \tag{58}$$

This formulation is the inverse of the exponential map at the origin, satisfying $\exp_0(\log_0(y)) = y$, and is computationally simple since the tangent space at the origin coincides with $\mathbb{R}^n$.

For a general point $x \in \mathbb{B}^n$, the logarithmic map must account for the local curvature around x. It can be expressed using the inverse of the Möbius addition:

$$\log_x(y) = \frac{2}{\lambda_x} \operatorname{arctanh}\big(\|-x \oplus y\|\big) \frac{-x \oplus y}{\|-x \oplus y\|}, \tag{59}$$

where

$$\lambda_x = \frac{2}{1 - \|x\|^2} \tag{60}$$

is the same conformal factor as in the exponential map, and $-x \oplus y$ denotes Möbius addition with the inverse of x.

Expanding the definition of the Möbius addition into the logarithmic map expression, we obtain an explicit formulation of $\log_x(y)$ in the Poincaré ball model. Substituting Eq. 54 into Eq. 59, we first compute the intermediate term $-x \oplus y$ as

$$-x \oplus y = \frac{(1 - 2\langle x, y \rangle + \|y\|^2)(-x) + (1 - \|x\|^2)y}{1 - 2\langle x, y \rangle + \|x\|^2\|y\|^2}. \tag{61}$$

Then, the logarithmic map can be written explicitly as

$$\log_x(y) = \frac{2}{\lambda_x} \operatorname{arctanh}\left(\frac{\|N(x,y)\|}{D(x,y)} \right) \frac{N(x,y)}{\|N(x,y)\|}, \tag{62}$$

where

$$N(x,y) = \left(1 - 2\langle x, y \rangle + \|y\|^2\right)(-x) + \left(1 - \|x\|^2\right) y, \tag{63}$$

$$D(x,y) = 1 - 2\langle x, y \rangle + \|x\|^2\|y\|^2, \quad \lambda_x = \frac{2}{1 - \|x\|^2}. \tag{64}$$

This expanded form explicitly expresses the logarithmic map in terms of x and y, showing how the non-linear geometry of the Poincaré ball modifies vector displacement through the Möbius addition. It provides the tangent vector at x that points toward y with a magnitude corresponding to the hyperbolic distance between them.

This formulation ensures that $\log_x(y)$ returns a tangent vector at x whose exponential map precisely recovers y, i.e., $\exp_x(\log_x(y)) = y$. Together, the exponential and logarithmic maps establish a smooth and invertible correspondence between the Euclidean tangent space and the curved manifold, enabling consistent optimization and representation learning in hyperbolic space.

Curvature The above formulation corresponds to the *unit-curvature* case, where the curvature is fixed as $K = -1$. In general, hyperbolic space has a constant negative curvature usually written as $K = -c$, where $c > 0$. When $c = 1$, the space has curvature -1, which is the normalized convention adopted by most works. The general form instead keeps c as a free curvature parameter, so the ball's radius becomes $1/\sqrt{c}$. This allows different degrees of curvature — flatter when $c \to 0$, and more curved when $c \to \infty$.

$$\mathbb{B}^n = \{z \in \mathbb{R}^n : c\|z\|^2 < 1\}, \tag{65}$$

where the radius is $1/\sqrt{c}$ and curvature $K = -c$. The Riemannian metric is scaled by the *conformal factor*

$$\lambda_x = \frac{2}{1 - c\|x\|^2}, \tag{66}$$

which defines the metric tensor as

$$\mathbf{G}_x = \lambda_x^2 \mathbf{I}. \tag{67}$$

The general form of the Poincaré-ball hyperbolic distance under curvature $K = -c$ $(c > 0)$ between two points $u, v \in \mathbb{B}^n$ is given by [13]

$$d_{\mathbb{H}}(u, v) = \frac{1}{\sqrt{c}} \operatorname{arcosh}\left(1 + 2c\frac{\|u - v\|^2}{(1 - c\|u\|^2)(1 - c\|v\|^2)}\right). \tag{68}$$

The exponential map with the curvature c is expressed as

$$\exp_x(v) = x \oplus_c \left(\tanh\left(\frac{\sqrt{c}\,\lambda_x\|v\|}{2}\right)\frac{v}{\sqrt{c}\,\|v\|}\right), \tag{69}$$

where

$$\lambda_x = \frac{2}{1 - c\|x\|^2}, \tag{70}$$

and $\oplus_c$ denotes the Möbius addition under curvature c.

The Möbius addition of two vectors x and y under curvature c is defined as [13, 28]

$$x \oplus_c y = \frac{(1 + 2c\langle x, y\rangle + c\|y\|^2)x + (1 - c\|x\|^2)y}{1 + 2c\langle x, y\rangle + c^2\|x\|^2\|y\|^2}. \tag{71}$$

So the explicit expression for $\exp_x(v)$ in hyperbolic space with curvature c is given by

$$\exp_x(v) = \frac{\left(1 + 2c\alpha\langle x, \frac{v}{\|v\|}\rangle + c\alpha^2\|v\|^2\right)x + (1 - c\|x\|^2)\alpha v}{1 + 2c\alpha\langle x, \frac{v}{\|v\|}\rangle + c^2\alpha^2\|x\|^2\|v\|^2}, \tag{72}$$

where

$$\alpha = \tanh\left(\frac{\sqrt{c}\,\lambda_x\|v\|}{2}\right), \quad \lambda_x = \frac{2}{1 - c\|x\|^2}. \tag{73}$$

For a general curvature $K = -c$ $(c > 0)$, the logarithmic map must account for the local curvature around x. It can be expressed using the inverse of the Möbius addition as

$$\log_x(y) = \frac{2}{\lambda_x\sqrt{c}} \operatorname{arctanh}\left(\sqrt{c}\|-x \oplus_c y\|\right)\frac{-x \oplus_c y}{\|-x \oplus_c y\|}, \tag{74}$$

where

$$\lambda_x = \frac{2}{1 - c\|x\|^2} \tag{75}$$

is the conformal factor, and $-x \oplus_c y$ denotes the Möbius addition under curvature $-c$.

Expanding the definition of Möbius addition into the logarithmic map expression, we obtain an explicit formulation of $\log_x(y)$ in the general Poincaré ball model with curvature $-c$. Substituting the c-dependent Möbius addition (Eq. 71) into Eq. 74, we first compute the intermediate term $-x \oplus_c y$ as

$$-x \oplus_c y = \frac{(1 - 2c\langle x, y\rangle + c\|y\|^2)(-x) + (1 - c\|x\|^2)y}{1 - 2c\langle x, y\rangle + c^2\|x\|^2\|y\|^2}. \tag{76}$$

Then, the logarithmic map can be written explicitly as

$$\log_x(y) = \frac{2}{\lambda_x\sqrt{c}} \operatorname{arctanh}\left(\sqrt{c}\frac{\|N_c(x, y)\|}{D_c(x, y)}\right)\frac{N_c(x, y)}{\|N_c(x, y)\|}, \tag{77}$$

where

$$N_c(x, y) = \left(1 - 2c\langle x, y\rangle + c\|y\|^2\right)(-x) + \left(1 - c\|x\|^2\right)y, \tag{78}$$

$$D_c(x, y) = 1 - 2c\langle x, y\rangle + c^2\|x\|^2\|y\|^2, \quad \lambda_x = \frac{2}{1 - c\|x\|^2}. \tag{79}$$

When $c = 1$, this expression reduces to the unit-curvature form of the logarithmic map given in Eq. 59.

Lorentz Model. The Lorentz model $\mathbb{L}^n$ represents hyperbolic space as the upper sheet of a two-sheeted hyperboloid embedded in Minkowski space $\mathbb{R}^{n+1}$, defined as

$$\mathbb{L}^n = \{p \in \mathbb{R}^{n+1} : \langle p, p\rangle_L = -1/\kappa,\ p_0 > 0\}, \tag{80}$$

where $\langle p, q\rangle_L = -p_0q_0 + \sum_{i=1}^n p_iq_i$ is the Lorentzian inner product. This formulation is unbounded and algebraically convenient, allowing closed-form computation of geodesic distances and stable gradient optimization. Because of its numerical robustness and simple analytical expressions for exponential and logarithmic maps, the Lorentz model is widely adopted in hyperbolic representation learning, particularly in entailment-based and hierarchical vision-language models.

1.6. Value Function

Reinforcement learning (RL) is about learning to act in an environment so as to maximize future reward. The value function in RL estimates the expected cumulative future reward that an agent can obtain from a particular state or state-action pair [70]. It helps the agent decide which actions are more desirable in the long run, guiding it towards making decisions that maximize its total reward over time. There are two main types: state-value function $V(s)$, which predicts the value of a given state, and state–action value function $V(s, a)$, which predicts the value of taking a specific action in a given state.

State Value Function. The value function $V(s)$ represents the expected return (cumulative reward) starting from a specific state s and following a particular policy, which defines the agent's strategy for choosing actions. It indicates how good it is for the agent to be in a certain state. For example, if an agent is located at a particular position in a maze, the value function $V(s)$ represents the expected total reward it will obtain from that point until it reaches the target, assuming it continues to follow its current policy.

State-Action Value Function. The value function $V(s, a)$ or $Q(s, a)$, also known as the action value function, represents the expected return (cumulative reward) starting from a specific state s, taking an action a, and subsequently following a particular policy. It indicates how good it is for the agent to take a specific action in a given state. For example, if an agent is at a particular position in a maze, the function $V(s, a)$ represents the expected total reward it will obtain by choosing a particular action at that point and then following its current policy until it reaches the target.

Formal Definition. Given a state s_t, an action a_t, a policy $\pi(a \mid s)$ that defines how the agent acts, a reward function $r(s, a)$, and a discount factor $\gamma \in [0, 1)$, the *state-value function* under policy π is formally defined as

$$V^{\pi}(s_t) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k \, r(s_{t+k}, a_{t+k}) \,\middle|\, s_t\right]. \tag{81}$$

It measures the expected cumulative reward that an agent will receive when starting from state s_t and following policy π thereafter.

Optimal Value Function. If the agent follows the best possible policy that maximizes the expected reward, we obtain the *optimal value function*:

$$V^{*}(s_t) = \max_{\pi} \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k \, r(s_{t+k}, a_{t+k}) \,\middle|\, s_t\right]. \tag{82}$$

Intuitively, $V^*(s_t)$ quantifies the maximum expected cumulative reward that an agent can achieve when starting from state s_t and following an optimal policy thereafter.

Bellman Optimality Equation. The value function encodes the long-term consequences of actions and forms the foundation of reasoning in reinforcement learning [6]. Once $V^*(s)$ is known, the optimal policy can be derived if the one-step transition dynamics $P(s' \mid s, a)$ are available. Specifically, the optimal policy π^* satisfies the Bellman optimality equation:

$$\pi^{*}(s) = \arg\max_{a} \left[r(s, a) + \gamma \sum_{s'} P(s' \mid s, a)\, V^{*}(s') \right]. \tag{83}$$

This recursive relationship expresses how the value of a state depends on the values of its successor states, thereby capturing the essence of sequential decision-making.

Interpretation. The optimal value function $V^*(s)$ can be viewed as a potential field or energy map over the state space. States with high value correspond to desirable or low-energy configurations that are closer to reward, whereas states with low value represent undesirable or high-energy configurations that are further away. From this perspective, acting optimally can be interpreted as following the gradient of the value landscape toward regions of higher value (or lower energy). This analogy bridges reinforcement learning with energy-based modeling, suggesting that value functions implicitly define an energy surface that guides the agent toward optimal behavior.

Path Value Function. In generic RL settings above, the value depends on future rewards under the optimal policy. But if we redefine reward as the negative energy cost $-c$ of moving between states:

$$r(s, a, s') = -c(s, s'), \tag{84}$$

and the cumulative reward becomes the total negative cost, then the optimal path value function $V^*(s, s')$ corresponds to the negative of the minimum accumulated cost from s to s' [19].

In this case, the optimal path value function $V^*(s, s')$ obeys triangle inequality [21, 49, 57]:

$$\begin{aligned} & c(s_1, s_3) \le c(s_1, s_2) + c(s_2, s_3) \\ \Rightarrow \quad & V^{*}(s_1, s_3) \ge V^{*}(s_1, s_2) + V^{*}(s_2, s_3). \end{aligned} \tag{85}$$

2. Motivation

World state transitions (from video observations) naturally form a hierarchical structure that is suitable represented in

hyperbolic space. Let s_t denote the state at time t and $\mathcal{A}$ be a discrete action set with cardinality $|\mathcal{A}| = B$. The world evolves according to the transition $s_{t+1} = f(s_t, a_t)$, where $a_t \in \mathcal{A}$. When predicting d steps into the future, each action choice produces a distinct future trajectory, resulting in $N_d = B^d$ possible future states. These futures form a exponentially branching tree, where the depth corresponds to the prediction horizon and the branching factor is determined by the action space. As a result, future world states are naturally organized hierarchically: states at smaller depths represent coarse, high-level abstractions, while states at larger depths correspond to finer, more detailed futures. Similar motivations are also supported by [64].

3. Baseline Details

As mentioned in Section 4.2 of main content, in both Procedural Planning (PP) and Visual Planning with Videos setup, we evaluate against three categories of baselines. *LLM-based* approaches rely on large language or vision-language models for reasoning, instruction following, and multi-step planning. *Generative (world) models* perform planning by generating pixels or latent video tokens and using visual rollouts to guide decision-making. *Predictive (world) models* focus purely on action prediction, estimating future action sequences directly without generating visual frames.

Random Selection. Following prior work [15], actions are sampled uniformly at random from the available action set to form a plan, without considering the task context.

Retrieval-Based. Following prior work [87], given the start and goal observations, this method retrieves the most similar trajectory from the training set by minimizing visual feature distance. The corresponding action sequence from the retrieved example is then used as the predicted plan.

LLM-based *LFP (Language-First Planning)* [42]. This method first converts both the start and goal observations into text and then prompts a large language model to infer the missing steps. The LLM predicts a sequence of intermediate actions based solely on language reasoning rather than visual planning.

VidAssist [36]. This method uses a vision-language model to extract temporal and spatial cues from the video, then queries a large language model to interpret these cues and generate an action sequence. The LLM refines and structures the predicted steps into a coherent plan, combining visual grounding with language-based reasoning.

SCHEMA [52]. This method performs procedure planning by modeling how states evolve over time. It aligns visual observations with textual state descriptions through cross-modal contrastive learning and uses a transformer backbone to represent state transitions. A large language model is then used to reason over these inferred intermediate states and generate the next actions, enabling structured step-by-step planning in instructional video settings.

Other VLMs. We also evaluate several large vision-language models, including InternVL3.5-241B [76], Qwen3-VL-Max [79], Gemini 2.5 Pro [20], and GPT-5 [53], using them in a zero-shot setting to perform visual reasoning and planning directly from video observations without task-specific training.

Generative (World) Models *DDN* [15]. This approach uses an autoregressive structure with two coordinated branches: one learns a compact representation of action steps, while the other predicts transitions in the latent feature space. By forecasting the next visual state rather than directly selecting actions, DDN models procedural progression through iterative frame prediction.

Int-MGAIL and Ext-MGAIL [7]. These generative models perform procedure planning by jointly learning a latent world model and an action policy through adversarial training, enabling multi-step action synthesis conditioned on visual goal states.

P^3IV [86]. This transformer-based model uses a learnable memory module together with an adversarial generation setup, and, similar to our method, outputs all action steps in a single forward pass rather than generating them sequentially.

PDPP [75]. This two-branch diffusion-based framework models temporal dependencies and action transitions, generating the full action sequence in parallel and progressively refining it over multiple denoising stages to improve coherence and logical structure.

KEPP [50]. This method incorporates structured procedural knowledge through a probabilistic knowledge graph learned from training plans, which serves as external guidance for step ordering. KEPP predicts the full action sequence in a single pass with limited supervision, producing strong performance in instructional video planning.

ActionDiffusion [61]. This diffusion-based approach generates the full action sequence by iteratively denoising a latent representation, allowing the model to refine predictions over multiple steps and capture long-term dependencies in instructional procedures.

MTID [87]. This model treats procedure planning as a multimodal trajectory generation problem, using a diffusion-based latent policy to synthesize complete action sequences conditioned on video observations while modeling long-term dependencies through iterative denoising.

VideoWorld [59]. This autoregressive framework generates future video frames step by step to model procedural progression, using predicted visual states to implicitly guide the unfolding action sequence.

Predictive (World) Models *WLTDO* [27]. This recurrent neural network model generates action sequences directly from paired observations, using temporal reasoning over the encoded features to predict ordered procedural steps.

UAAA [1]. This two-stage method predicts action steps autoregressively by combining an RNN with a hidden Markov model to model temporal uncertainty and step transitions in procedural tasks.

UPN [68]. This method learns a differentiable latent space suitable for planning by predicting trajectories in feature space, and a softmax output layer is used to convert the continuous plan representation into discrete action steps.

PlaTe [69]. This model builds on DDN by introducing transformer modules into its dual-branch architecture for action and state prediction, but follows a distinct evaluation protocol compared to other procedure planning methods.

E3P [74]. This method adopts an event-centric formulation, inferring latent events from visual observations and using them to guide intermediate action prediction. Through event-aware prompting and action relation modeling, E3P improves the logical structure of predicted steps and achieves strong performance on procedural planning benchmarks.

V-JEPA 2 [3]. A large-scale predictive world model pretrained on masked latent feature prediction over one million hours of unlabeled video. Action-conditioned post-training enables autoregressive rollouts for planning without pixel generation.

4. Energy Landscape

To better illustrate the difference between Euclidean predictive world models and our hyperbolic formulation, we visualize the *energy landscape* around a given latent state.

Δx and Δy. In the original V-JEPA 2-AC setup [3], Δx and Δy represent physical end-effector offsets in Cartesian coordinates. The visualization shows how the model's energy changes as the end-effector's target position varies along the Δx and Δy axes while keeping the vertical displacement fixed ($\Delta z = 0$).

Formally, the plotted quantity is:

$$s_{t+1}^{\text{hyp}} = s_t + (\Delta x, \Delta y, 0), \tag{86}$$

$$\text{Energy}(\Delta x, \Delta y) = c(s_t, s_{t+1}^{\text{hyp}}). \tag{87}$$

where c denotes the energy cost defined in Eq. 11.

In visual planning, Δx and Δy are no longer physical displacements. Instead, they represent latent displacements that probe the local geometry of the world model around a visual state.

In the Euclidean space, the encoder maps an observation x_t into a latent vector $s_t^x \in \mathbb{R}^n$. To visualize how the model evaluates hypothetical future states, V-JEPA 2 [3] perturbs the latent representation along two Euclidean axes. We choose two orthonormal directions in latent space, $u_1, u_2 \in \mathbb{R}^n$. A natural, semantically aligned choice is:

$$u_1 = \frac{E_\theta(x_{t+T}) - E_\theta(x_t)}{\|E_\theta(x_{t+T}) - E_\theta(x_t)\|}, \tag{88}$$

which represents the direction from the current state toward the goal (i.e., progress along the procedure).

The second direction, u_2, spans variations orthogonal to this progress direction (i.e., sampled from another trajectory at the same step and then orthonormalized against u_1).

Then, a hypothetical next latent state is defined as

$$s_{t+1}^{\text{hyp}} = s_t^x + \Delta x\, u_1 + \Delta y\, u_2, \tag{89}$$

and the corresponding energy landscape is

$$\text{Energy}(\Delta x, \Delta y) = \left\| s_{t+1}^x - s_{t+1}^{\text{hyp}} \right\|. \tag{90}$$

In GeoWorld, the encoder maps each observation x_t to a latent representation $s_{t,\mathbb{H}}^x$ on the hyperbolic manifold. To probe the local geometry around this latent state, we sweep two orthonormal directions in the tangent space $\mathbf{T}_0\mathbb{H}^n$, denoted as $(\Delta x, \Delta y)$. Each coordinate pair $(\Delta x, \Delta y)$ corresponds to a small displacement applied *at the tangent space* before projection onto the manifold via the exponential map:

$$s_{t+1,\mathbb{H}}^{\text{hyp}} = \exp_0\big(s_t^x + \Delta x\, u_1 + \Delta y\, u_2\big), \tag{91}$$

where u_1 and u_2 form an orthonormal basis in $\mathbf{T}_0\mathbb{H}^n$. Thus, $(\Delta x, \Delta y)$ describes *local perturbations of the latent state*, not pixel space offsets.

And the energy landscape is

$$\text{Energy}_{\mathbb{H}}(\Delta x, \Delta y) = d_{\mathbb{H}}\Big(s_{t+1,\mathbb{H}}^x,\ s_{t+1,\mathbb{H}}^{\text{hyp}}\Big). \tag{92}$$

Visualization. In Figure 2, we select a reference latent state s_t from the initial step of the *Replace Memory Chip* task in the COIN dataset [71], and visualize the local energy geometry by sweeping two orthonormal tangent-space directions $(\Delta x, \Delta y)$ around this state. Figure 2 compares the Euclidean (left) and hyperbolic (right) landscapes. The Euclidean surface shows a smooth, nearly symmetric paraboloid with weak directional structure, indicating that V-JEPA 2 treats perturbations homogeneously. In contrast, the hyperbolic surface in GeoWorld forms a sharper, curvature-aware basin with more pronounced directional variation. This reflects the ability of H-JEPA to encode hierarchical structure: states positioned higher in the task hierarchy lie at hyperbolically greater distances, creating more informative energy gradients during planning.

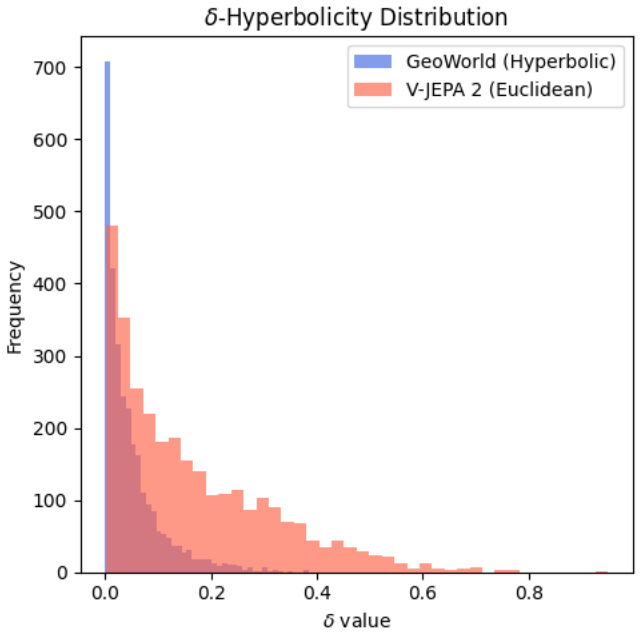


Figure 1. Gromov δ-hyperbolicity on CrossTask [88].

Such curvature-aware energy landscapes promote more stable long-horizon planning: CEM naturally follows the hyperbolic geodesics shaped by GeoWorld, resulting in more accurate multi-step trajectory optimization.

5. Ablation Study

Curvature As discussed in Section 4.3, the curvature $K = -c$ is learned in the logarithmic space by optimizing $\log(c)$, which is initialized at $c = 1$ and treated as a learnable scalar. This formulation ensures that c remains positive and stabilizes the gradients of both the hyperbolic distance and the exponential map [14, 25]. The learned curvature is further clamped to the range $[0.1, 10.0]$ to prevent training instability.

We analyze how the learnable curvature evolves during training and how it influences geometric planning quality. As shown in Fig. 2 (d), the curvature parameter c in GeoWorld typically starts near 1 and gradually decreases to a stable value around 0.3, indicating that the model learns a flatter yet still hyperbolic latent geometry. A smaller curvature reduces distortion in the exponential map and leads to more stable multi-step planning, especially for larger backbone encoders. The geometric effect of curvature is further visualized in Fig. 2 (a)–(c): as c decreases, geodesic paths bend less aggressively toward the origin (Fig. 2 (a)), boundary-anchored geodesic patterns become flatter (Fig. 2 (b)), and the hyperbolic distance between x and y contracts smoothly as curvature approaches zero (Fig. 2 (c)). This suggests that moderate negative curvature is sufficient to capture hierarchical structure while preserving stable value propagation across long planning horizons.

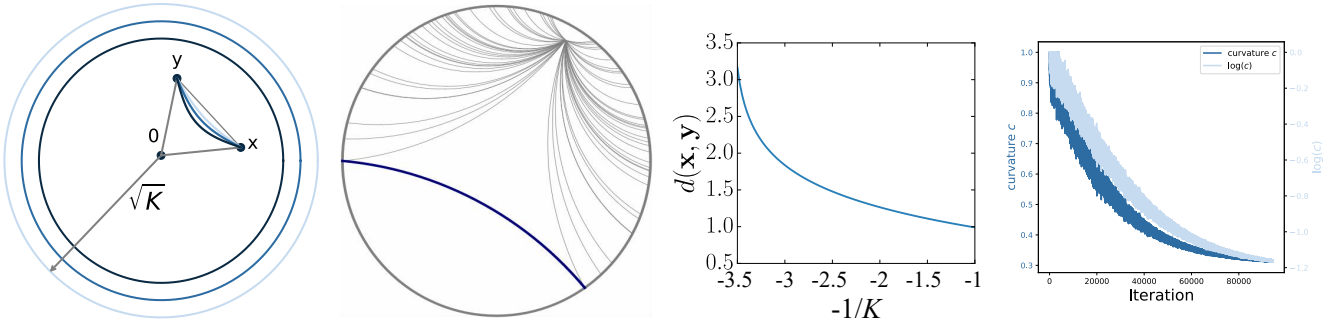
Table 1. Ablation of frozen encoder vs. fully fine-tuned model for visual planning with videos on CrossTask [88].

Method	T=3			T=4		
	SR	mAcc	mIoU	SR	mAcc	mIoU
GeoWorld ViT-L	44.80	70.54	86.30	30.63	65.46	79.73
w/ FFT	45.20	71.17	87.16	31.34	67.16	80.38
GeoWorld ViT-H	47.79	74.42	88.84	34.51	68.89	82.95
w/ FFT	48.46	74.94	89.10	34.95	69.42	83.47
GeoWorld ViT-g	49.23	76.64	90.61	35.49	71.00	84.50
w/ FFT	49.57	76.86	91.04	35.91	71.76	85.13
GeoWorld ViT-g_{384}	51.71	77.30	92.95	37.04	71.35	87.04
w/ FFT	52.04	77.98	93.61	37.85	72.24	87.80

Gromov δ-Hyperbolicity We visualize Gromov δ-hyperbolicity by sampling latent quadruples in CrossTask [88] and evaluating the four-point condition under each model's intrinsic metric (hyperbolic geodesic distance for GeoWorld and Euclidean distance for V-JEPA 2). As shown in Fig. 1, GeoWorld exhibits a substantially more concentrated distribution of near-zero δ values, indicating a stronger tree-like hierarchical geometry in its learned representation space.

Frozen Encoder vs. Fully Fine-Tuned As shown in Table 1, we evaluate the impact of Fully Fine-Tuning (FFT) the encoder during the supervised finetuning stage, compared to the original configuration where the encoder remains frozen and only a lightweight exponential projection layer is trainable. Fully fine-tuning yields consistent yet modest improvements across all metrics and model scales, with gains of approximately $0.3{-}0.8\%$ in SR and $0.5{-}1.2\%$ in mAcc and mIoU for both $T{=}3$ and $T{=}4$ planning horizons. While these improvements indicate that the encoder can still adapt beneficially to downstream visual planning objectives, the gains come at the cost of significantly increased trainable parameters and slower optimization. Moreover, the relative performance margin narrows as model size increases, suggesting diminishing returns for larger backbones. These results imply that the frozen-encoder design already captures task-relevant structure effectively, and full encoder finetuning provides only incremental benefit relative to the additional computation and memory overhead introduced.

Effectiveness of GRL As shown in Table 2, incorporating Geometric Reinforcement Learning (GRL) leads to clear and consistent improvements over the supervised finetuning (SFT) baseline. While SFT alone yields marginal gains over the pretrained V-JEPA 2 model, applying GRL independently further boosts SR, mAcc, and mIoU across both planning horizons, suggesting that GRL better aligns



(a) Curvature and geodesics. (b) Geodesic patterns. (c) Distance vs. curvature. (d) Curvature trend during training.

Figure 2. **Geometric effects and curvature dynamics**: (a) Poincaré disk geodesics connecting x and y under different curvatures K. As the curvature K becomes less negative (i.e., closer to 0), the hyperbolic distance between x and y increases, and the geodesic paths bend less and shift closer toward the origin. (b) Geodesic patterns induced by different boundary anchor points. Varying the anchor location produces a characteristic geodesic fan in the Poincaré disk. (c) As the curvature becomes less negative, the space flattens and the distance between x and y decreases. (d) Learnable curvature c during supervised training, showing a gradual decrease from its initialization and convergence to a stable value 0.3.

Table 2. Ablation of Supervised Fine-Tuning (SFT) vs. Geometric Reinforcement Learning (GRL) for visual planning with videos on CrossTask [88].

Method	T=3			T=4		
	SR	mAcc	mIoU	SR	mAcc	mIoU
V-JEPA 2 ViT-g_{384} [3]	50.16	74.86	91.73	35.01	70.24	85.05
GeoWorld ViT-g_{384}						
SFT Only	50.42	75.13	91.94	35.92	70.79	85.88
GRL Only	51.04	76.48	92.42	36.33	71.04	86.31
SFT + GRL	51.71	77.30	92.95	37.04	71.35	87.04

Table 3. Ablation of weighting hyperparameter λ in Supervised Fine-Tuning (SFT) Only for visual planning with videos on CrossTask [88].

Method	T=3			T=4		
	SR	mAcc	mIoU	SR	mAcc	mIoU
V-JEPA 2 ViT-g_{384} [3]	50.16	74.86	91.73	35.01	70.24	85.05
GeoWorld ViT-g_{384}						
$\lambda = 1,\ 1-\lambda = 0$	50.16	74.88	91.79	34.65	69.48	84.10
$\lambda = 0.9,\ 1-\lambda = 0.1$	50.19	74.91	91.84	34.95	70.05	84.85
$\lambda = 0.8,\ 1-\lambda = 0.2$	50.25	74.96	91.88	35.31	70.40	85.27
$\lambda = 0.7,\ 1-\lambda = 0.3$	50.33	75.02	91.89	35.57	70.57	85.46
$\lambda = 0.6,\ 1-\lambda = 0.4$	50.37	75.06	92.92	35.82	70.66	85.72
$\lambda = 0.5,\ 1-\lambda = 0.5$	**50.42**	**75.13**	**91.94**	35.92	70.79	**85.88**
$\lambda = 0.3,\ 1-\lambda = 0.7$	50.39	75.07	91.82	**35.97**	**70.86**	85.74

the learned energy landscape with multi-step planning objectives. The combination of SFT and GRL achieves the strongest performance, indicating that SFT provides a strong initialization while GRL refines the latent dynamics toward energy-minimizing trajectories required for long-horizon reasoning. These findings highlight the complementary nature of supervised learning and reinforcement-based value shaping in predictive world models.

SFT Hyperparameters As shown in Table 3, incorporating the rollout loss into SFT consistently improves visual planning performance over the pure one-step objective ($\lambda = 1$). Once $1-\lambda > 0$, all metrics exhibit steady gains, indicating that multi-step rollout supervision provides additional temporal consistency beyond standard single-step training. As the rollout weight increases (i.e., smaller λ), improvements become more pronounced, particularly for the longer planning horizon ($T = 4$). For example, SR and mIoU steadily increase as λ decreases from 1 to 0.5, suggesting that stronger rollout supervision effectively mitigates error accumulation over longer sequences. This trend aligns with the intuition that longer-horizon prediction requires explicit multi-step consistency constraints rather than relying solely on local one-step accuracy. A balanced weighting around $\lambda = 0.5$ achieves the strongest overall performance across metrics, demonstrating that equal emphasis on one-step prediction and rollout consistency yields the best trade-off. Further increasing the rollout weight (e.g., $\lambda = 0.3$) leads to negligible changes for the shorter horizon ($T = 3$), while yielding slight yet consistent gains for the longer horizon ($T = 4$). This behavior indicates that stronger rollout supervision primarily benefits long-horizon planning, especially under the hyperbolic structure where multi-step geodesic consistency becomes more critical. In contrast, short-horizon planning does not induce strong hierarchical structure, limiting the advantage of hyperbolic geometry and GRL. The primary benefit of GeoWorld emerges as the planning horizon increases, where exponential branching and long-term abstraction become critical, as shown in Table 5.

GRL Hyperparameters As shown in Table 4, both the discount factor γ and the regularization weight β play im-

Table 4. Ablation of discount factor γ and the regularization weight $beta$ in Geometric Reinforcement Learning (GRL) for visual planning with videos on CrossTask [88].

Method	T=3			T=4		
	SR	mAcc	mIoU	SR	mAcc	mIoU
V-JEPA 2 ViT-g_{384} [3]	50.16	74.86	91.73	35.01	70.24	85.05
GeoWorld ViT-g_{384}						
SFT Only	50.42	75.13	91.94	35.92	70.79	85.88
$\beta = 0,\ \gamma = 0.99$	50.48	75.27	91.99	36.07	70.94	86.07
$\beta = 0.05,\ \gamma = 0.99$	51.04	76.21	92.39	36.58	71.13	86.45
$\beta = 0.2,\ \gamma = 0.99$	51.69	77.25	92.83	**37.15**	71.33	86.96
$\beta = 0.1,\ \gamma = 0.90$	51.02	76.39	92.04	36.42	70.88	86.33
$\beta = 0.1,\ \gamma = 0.95$	51.44	76.94	92.75	36.85	71.05	86.67
$\beta = \mathbf{0.1},\ \gamma = \mathbf{0.99}$	**51.71**	**77.30**	**92.95**	37.04	**71.35**	**87.04**

portant roles in shaping the learning dynamics in GRL. Increasing γ strengthens long-horizon supervision by assigning greater weight to later predicted steps, which benefits multi-step rollout consistency and improves SR, mAcc, and mIoU as the planning horizon increases from T=3 to T=4. Meanwhile, introducing the triangle inequality regularization term through $\beta > 0$ consistently boosts performance compared to the $\beta = 0$ setting, demonstrating that enforcing hyperbolic geodesic constraints helps stabilize the predictor and prevents degenerate shortcuts in latent space. Moderate regularization ($\beta = 0.1$) paired with a large discount factor ($\gamma = 0.99$) achieves the strongest results, indicating that encouraging long-horizon consistency while softly enforcing geodesic structure yields the most effective balance. These results validate the effectiveness of GRL as both a geometric constraint mechanism and a planning-aligned training signal.

Hyperbolic Geometry vs. GRL in Long-Horizon Planning *Section 4.5* in main paper reports results up to T=6, following the long-horizon setting in [87]. Table 5 further extends the evaluation to T=8 to stress-test planning stability under increasingly long rollouts. As the horizon grows, the vanilla V-JEPA 2 baseline exhibits rapid performance degradation, with SR dropping sharply from 50.16 at T=3 to 4.95 at T=8, highlighting severe error accumulation in long-horizon prediction. Introducing hyperbolic geometry substantially mitigates this collapse. SFT in hyperbolic space already improves stability at longer horizons, maintaining significantly higher SR at $T \geq 7$. Applying GRL in Euclidean space further strengthens multi-step consistency and consistently outperforms the baseline, demonstrating that rollout-based geometric regularization alone contributes meaningful gains even without hyperbolic modeling. When GRL is implemented in hyperbolic space, the advantage becomes more pronounced, particularly for $T \geq 6$, suggesting that enforcing geodesic consistency in a curvature-aware latent space better preserves long-range structural dependencies. The full model (SFT + GRL) achieves the strongest results across all horizons, with the performance gap widening as T increases. This trend indicates that SFT and GRL play complementary roles: SFT stabilizes short-term prediction, while GRL enhances long-horizon rollout consistency, together yielding a clear advantage in extended planning scenarios.

Table 5. SR of long horzion planning on CrossTask [88] videos.

Method	T=3	T=4	T=5	T=6	T=7	T=8
V-JEPA 2 ViT-g_{384}	50.16	35.01	23.17	16.88	8.26	4.95
SFT (Hyperbolic)	50.42	35.92	23.64	16.97	14.88	11.51
GRL (Euclidean)	50.26	35.47	23.85	17.03	15.12	12.74
GRL (Hyperbolic)	51.04	36.33	24.05	17.82	15.54	13.10
SFT + GRL	51.71	37.04	24.83	18.26	16.09	13.81

6. Error Accumulation in Long-Horizon Planning

Autoregressive (AR) methods inevitably lead to error accumulation in long-horizon planning, which is why many existing works focus on mitigating this issue through rollout loss. However, our claim is not that hierarchy replaces this effect, but that geometry shapes how errors accumulate. In Euclidean latent spaces, small prediction errors cause unconstrained drift that compounds uniformly over time, whereas hyperbolic geometry imposes a hierarchical structure on the latent space that constrains long-horizon trajectories along geodesically meaningful directions. In this sense, error accumulation and geometric drift are closely related: hierarchical geometry mitigates how errors propagate, while rollout loss and GRL help eliminate them.

7. Limitation and Future Work

Our intuition for hierarchical structure arises from state transitions in multi-step planning over futures. Therefore, even when the action sequences annotated in CrossTask [88] and COIN [71] appear linear, predicting d-step futures from a state induces an exponentially branching set of possible trajectories (B^d), forming an implicit tree underlying a hierarchical structure. We must clarify that, as mentioned in *Section 1.2*, sub-task hierarchies involving multi-level planning are the intuition of the original JEPA [41]. However, the hierarchical structure in GeoWorld arises from multi-step future expansion, rather than from explicit high-level planning and low-level execution.

Future work may involve sub-task hierarchies, such as high-level task labels, mid-level actions, and low-level end-effectors. Moreover, our framework is compatible with embodied planning. As this is a computer vision conference, we plan to extend our work to embodied settings in the future.