Appendix

In this Appendix, we discuss how we factorize the hierarchical decision-making process in Section A. In Section B, we then detail the background and architecture for visually grounded task planning, visual planning with video diffusion, and action planning with inverse dynamics model. In Section C, we discuss the training and evaluation details for the different levels of planning in `HiP` and the corresponding training hyperparameters. Finally, in Section D, we showcase additional ablation studies comparing different approaches to enforce consistency across the levels of hierarchy.

A Factorizing Hierarchical Decision-Making Process

We model the hierarchical decision-making process described in Section 2 with p_Θ which can be factorized into the task distribution p_θ, visual distribution p_ϕ, and action distribution p_ψ.

$$p_\Theta(W, \{\tau_x^i\}, \{\tau_a^i\}|g, x_{1,1}) = \prod_{i=1}^{N} p_\theta(w_i|g, x_{i,1}, w_{<i}, \tau_x^{<i}, \tau_a^{<i}) \prod_{i=1}^{N} p_\phi(\tau_x^i|w_{\leq i}, x_{i,1}, g, \tau_x^{<i}, \tau_a^{<i}) \prod_{i=1}^{N} p_\psi(\tau_a^i|\tau_x^{\leq i}, w_{\leq i}, x_{i,1}, g, \tau_a^{<i})$$

Here, given random variables Y^i, $Y^{<i}$ and $Y^{\leq i}$ represents $\{Y^1, \ldots, Y^{i-1}\}$ and $\{Y^1, \ldots, Y^i\}$ respectively. Now, we apply Markov assumption: Given current observation $x_{i,1}$, future variables $(w_i, \tau_x^i, \tau_a^i)$ and past variables $(w_j, \tau_x^j, \tau_a^j \;\; \forall j < i)$ are conditionally independent.

$$p_\Theta(W, \{\tau_x^i\}, \{\tau_a^i\}|g, x_{1,1}) = \prod_{i=1}^{N} p_\theta(w_i|g, x_{i,1}) \prod_{i=1}^{N} p_\phi(\tau_x^i|w_i, x_{i,1}, g) \prod_{i=1}^{N} p_\psi(\tau_a^i|\tau_x^i, w_i, x_{i,1}, g)$$

We model task distribution p_θ with a large language model (LLM) which is independent of observation $x_{i,1}$. Since the image trajectory $\tau_x^i = \{x_{i,1:T}\}$ describes a physically plausible plan for achieving subgoal w_i from observation $x_{i,1}$, it is conditionally independent of goal g given subgoal w_i and observation $x_{i,1}$. Furthermore, we assume that an action $a_{i,t}$ can be recovered from observation at the same timestep $x_{i,t}$ and the next timestep $x_{i,t+1}$. Thus, we can write the factorization as

$$p_\Theta(W, \{\tau_x^i\}, \{\tau_a^i\}|g, x_{1,1}) = \left(\prod_{i=1}^{N} p_\theta(w_i|g)\right) \left(\prod_{i=1}^{N} p_\phi(\tau_x^i|w_i, x_{i,1})\right) \left(\prod_{i=1}^{N} \prod_{t=1}^{T-1} p_\psi(a_{i,t}|x_{i,t}, x_{i,t+1})\right)$$

B Background and Architecture

B.1 Task Planning

Background on Density Ratio Estimation. Let p and q be two densities, such that q is absolutely continuous with respect to p, denoted as $q << p$ i.e. $q(\boldsymbol{x}) > 0$ wherever $p(\boldsymbol{x}) > 0$. Then, their ratio is defined as $r(\boldsymbol{x}) = p(\boldsymbol{x})/q(\boldsymbol{x})$ over the support of p. We can estimate this density ratio $r(\boldsymbol{x})$ by training a binary classifier to distinguish between samples from p and q [44, 14, 17]. More recent work [43] has shown one can introduce auxiliary densities $\{m_i\}_{i=1}^M$ and train a multi-class classifier to distinguish samples between M classes to learn a better-calibrated and more accurate density ratio estimator. Once trained, the log density ratio can be estimated by $\log r(\boldsymbol{x}) = \hat{h_p}(\boldsymbol{x}) - \hat{h_q}(\boldsymbol{x})$, where $\hat{h_i}(\boldsymbol{x})$ is the unnormalized log probability of the input sample under the i^{th} density, parameterized by the model.

Learning a Classifier to Visually Ground Task Planning. We estimate the density ratio $\frac{p(x_{i,1}|w_i,g)}{p(x_{i,1}|g)}$ with a multi-class classifier $f_\phi(x_{i,1}, \{w_j\}, g)$ trained to distinguish samples amongst the conditional distributions $p(x_{i,1}|w_i, g), ..., p(x_{i,1}|w_M, g)$ and the marginal distribution $p(x_{i,1}|g)$. Upon convergence, the classifier learns to assign high scores to $(x_{i,1}, w_i, g)$ if w_i is the subgoal corresponding to the observation $x_{i,1}$ and task g and low scores otherwise.

Architecture. We parameterize f_ϕ as a 4-layer multi-layer perceptron (MLP) on top of an ImageNet-pretrained vision encoder (ResNet-18 [15]) and a frozen pretrained language encoder (Flan-T5-Base [8]). The vision encoder encodes the observation $x_{i,1}$, and the text encoder encodes the subgoals

w_j and the goal g. The encoded observation, the encoded subgoals, and the encoded goal are concatenated, and passed through a MLP with 3 hidden layers of sizes 512, 256, and 128. The output dimension for MLP (i.e., number of classes for multi-classification) M is 6 for `paint-block` environment and 5 for `object-arrange` environment.

Choice of Large Language Model. We use GPT3.5-turbo [36] as our large language model.

B.2 Visual Planning

Background. Diffusion Probabilistic Models [41, 20] learn the data distribution $h(\boldsymbol{x})$ from a dataset $\mathcal{D} \coloneqq \{\boldsymbol{x}^i\}$. The data-generating procedure involves a predefined forward noising process $q(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k)$ and a trainable reverse process $p_\phi(\boldsymbol{x}_{k-1}|\boldsymbol{x}_k)$, both parameterized as conditional Gaussian distributions. Here, $\boldsymbol{x}_0 \coloneqq \boldsymbol{x}$ is a sample, $\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_{K-1}$ are the latents, and $\boldsymbol{x}_K \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ for a sufficiently large K. Starting with Gaussian noise, samples are then iteratively generated through a series of "denoising" steps. Although a tractable variational lower-bound on $\log p_\phi$ can be optimized to train diffusion models, Ho et al. [20] propose a simplified surrogate loss:

$$\mathcal{L}_{\text{denoise}}(\theta) \coloneqq \mathbb{E}_{k\sim[1,K],\boldsymbol{x}_0\sim h,\epsilon\sim\mathcal{N}(\mathbf{0},\boldsymbol{I})}[||\epsilon - \epsilon_\phi(\boldsymbol{x}_k, k)||^2]$$

The predicted noise $\epsilon_\theta(\boldsymbol{x}_k, k)$, parameterized with a deep neural network, estimates the noise $\epsilon \sim \mathcal{N}(0, I)$ added to the dataset sample $\boldsymbol{x}_0$ to produce noisy $\boldsymbol{x}_k$.

Guiding Diffusion Models with Text. Diffusion models are most notable for synthesizing high-quality images [39, 34] and videos [21, 49] from text descriptions. Modeling the conditional data distribution $q(\boldsymbol{x}|\boldsymbol{y})$ makes it possible to generate samples satisfying the text description $\boldsymbol{y}$. To enable conditional data generation with diffusion, Ho and Salimans [19] modified the original training setup to learn both a conditional $\epsilon_\phi(\boldsymbol{x}_k, \boldsymbol{y}, k)$ and an unconditional $\epsilon_\phi(\boldsymbol{x}_k, k)$ model for the noise. The unconditional noise is represented, in practice, as the conditional noise $\epsilon_\phi(\boldsymbol{x}_k, \emptyset, k)$, where a dummy value $\emptyset$ takes the place of $\boldsymbol{y}$. The perturbed noise $\epsilon_\phi(\boldsymbol{x}_k, \emptyset, k) + \omega\big(\epsilon_\phi(\boldsymbol{x}_k, \boldsymbol{y}, k) - \epsilon_\phi(\boldsymbol{x}_k, \emptyset, k)\big)$ (i.e. *classifier-free guidance*) is used to later generate samples.

Video Diffusion in Latent Space. As diffusion models generally perform denoising in the input space [20], the optimization and inference become computationally demanding when dealing with high-dimensional data, such as videos. Inspired by recent works [38, 49], we first use an autoencoder v_{enc} to learn a latent space for our video data. It projects an observation trajectory τ_x (i.e., video) into a 2D tri-plane representation [49] $\tau_z = [\tau_z^T, \tau_z^H, \tau_z^W]$ where τ_z^T, τ_z^H, τ_z^W capture variations in the video across time, height, and width respectively. We then diffuse over this learned latent space [49].

Latent Space Video Diffusion for Visual Planning. Our video diffusion model $p_\phi(\tau_x^i|w_i, x_{i,1})$ generates video τ_x^i given a language subgoal w_i and the current observation $x_{i,1}$. It is parameterized through its noise model $\epsilon_\phi((\tau_z^i)_k, w_i, x_{i,1}, k) \coloneqq \epsilon_\phi((\tau_z^i)_k, l_{\text{enc}}(w_i), v_{\text{enc}}(x_{i,1}), k)$ where $\tau_z^i \coloneqq v_{\text{enc}}(\tau_x^i)$ is the latent representation of video τ_x^i over which we diffuse. We condition the noise model ϵ_ϕ on subgoal w_i using a pretrained language encoder l_{enc} and on current observation $x_{i,1}$ using video encoder v_{enc}. To use v_{enc} with a single observation $x_{i,1}$, we first tile the observation along the temporal dimension to create a video.

Architecture. We now detail the architectures of different components:

- **Video Autoencoder** We borrow our architecture for v_{enc} from PVDM [49] which uses transformers to project video $\tau_x \in \mathbb{R}^{T\times H\times W}$ to latent codes $\tau_z = [\tau_z^T, \tau_z^H, \tau_z^W]$ where $\tau_z^T \in \mathbb{R}^{C\times H'\times W'}$, $\tau_z^H \in \mathbb{R}^{C\times T\times W'}$, $\tau_z^W \in \mathbb{R}^{C\times H'\times T}$. Here, $T = 50$ represents the time horizon of a video, $H = 48$ represents video height, $W = 64$ represents video width, $C = 4$ represents latent codebook dimension, $H' = 12$ represents latent height, and $W' = 8$ represents latent width.
- **Language Encoder** We use Flan-T5-Base [8] as the pretrained frozen language encoder l_{enc}.
- **Noise Model** We borrow PVDM-L architecture [49] which uses 2D UNet architecture, similar to the one in Latent Diffusion Model (LDM) [38], to represent $p(\tau_z|\tau_z')$. In our case, $\tau_z = v_{\text{enc}}(\tau_x^i)$ and $\tau_z' = v_{\text{enc}}(x_{i,1})$. To further condition noise model ϵ_ϕ on $l_{\text{enc}}(w_i)$, we augment the 2D UNet Model with cross-attention mechanism borrowed by LDM [38].

For implementing these architectures, we used the codebase https://github.com/sihyun-yu/PVDM which contains the code for PVDM and LDM.

Classifier for Consistency between Visual Planning and Action Planning. To ensure consistency between visual planning and action planning, we want to sample observation trajectories that maximizes both conditional observation trajectory likelihood from diffusion and the likelihood of sampled actions given the observation trajectory (see equation 4). To approximate likelihood calculation of action trajectory, we learn a binary classifier g_ψ that models if the observation trajectory leads to a high likelihood action trajectories. Since diffusion happens in latent space and we use gradients from g_ψ to bias the denoising of the video diffusion, $g_\psi(\tau_z^i)$ takes the observation trajectory in latent space. The binary classifier g_ψ is trained to distinguish between observation trajectories in latent space sampled from video dataset $\tau_z^i = v_{\text{enc}}(\tau_x^i), \tau_x^i \sim \mathcal{D}_{\text{video}}$ (i.e. label of 1) and observation trajectories in latent space sampled from video dataset whose frames where randomly shuffled $(\tau_z^i)' = v_{\text{enc}}(\sigma(\tau_x^i)), \tau_x^i \sim \mathcal{D}_{\text{video}}$ (i.e. label of 0). Here, σ denotes the random shuffling of frames. To randomly shuffle frames in an observation trajectory (of length 50), we first randomly select 5 frames in the observation trajectory. For each of the selected frame, we randomly permute it with its neighboring frame (i.e. either with the frame before it or with the frame after it). Once g_ψ is trained, we use it to bias the denoising of the video diffusion

$$\hat{\epsilon} := \epsilon_\phi((\tau_z)_k, v_{\text{enc}}(x_t), k) + \omega(\epsilon_\phi((\tau_z)_k, v_{\text{enc}}(x_t), l_{\text{enc}}(w), k) - \epsilon_\phi((\tau_z)_k, v_{\text{enc}}(x_t), k)) - \omega' \nabla_{(\tau_z)_k} \log g_\psi(1|(\tau_z)_k)$$

Here, $\hat{\epsilon}$ is the noise used in denoising of the video diffusion and ω, ω' are guidance hyperparameters.

Classifier Architecture. The classifier $g_\psi(\tau_z = [\tau_z^T, \tau_z^H, \tau_z^W])$ has a ResNet-9 encoder that converts τ_z^T, τ_z^H, and τ_z^W to latent vectors, then concatenate those latent vectors and passes the concatenated vector through an MLP with 2 hidden layers of sizes 256 and 128 and an output layer of size 1.

B.3 Action Planning

To do action planning, we learn an inverse dynamics model to $p_\psi(a_{i,t}|x_{i,t}, x_{i,t+1})$ predicts 7-dimensional robot states $s_{i,t} = p_\psi(x_{i,t})$ and $s_{i,t+1} = p_\psi(x_{i,t+1})$. The first 6 dimensions of the robot state represent joint angles and the last dimension of the robot state represents the gripper state (i.e., whether it's open or closed). The first 6 action dimension is represented as joint angle difference $a_{i,t}[:6] = s_{i,t+1}[:6] - s_{i,t}[:6]$ while the last action dimension is gripper state of next timestep $a_{i,t}[-1] = s_{i,t+1}[-1]$.

Architecture. We use ViT-B [9] (VC-1 [31] initialization) along with a linear layer to parameterize p_ψ. ViT-B projects the observation $x_{i,t} \in \mathbb{R}^{48\times64\times3}$ to 768 dimensional latent vector from which the linear layer predicts the 7 dimensional state $s_{i,t}$.

C Training and Evaluation

C.1 Task Planning

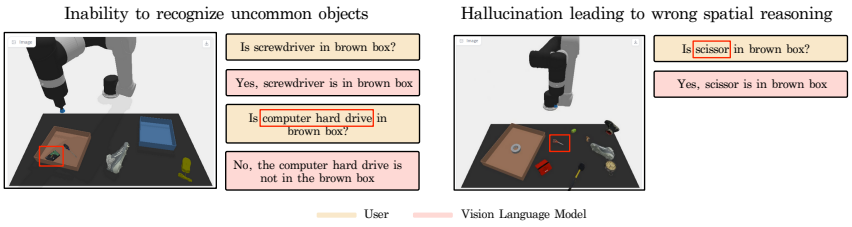

Figure 8: **Failure in VLM.** to recognize uncommon objects like computer hard drives and occasional hallucination of object presence, leading to incorrect visual reasoning.

Training Objective and Dataset for Learned Classifier. We use a softmax cross-entropy loss to train the multi-class classifier $f_\phi(x_{i,1}, \{w_j\}_{j=1}^M, g)$ to classify an observation $x_{i,1}$ into one of the M given subgoal. We train it using the classification dataset $\mathcal{D}_{\text{classify}} := \{x_{i,1}, g, \{w_j\}_{j=1}^M, i\}$ consisting of observation $x_{i,1}$, goal g, candidate subgoals $\{w_j\}_{j=1}^M$ and the correct subgoal label i. While the dataset for `paint-block` contains approximately 58k data points, the dataset for `object-arrange` consists of 82k data points.

Vision-Language Model (VLM) as a Classifier. We use a frozen pretrained Vision-Language Model (VLM) (MiniGPT4 [53]) as a classifier. We first sample a list of all possible subgoals $W = \{w_i\}_{i=1}^M$ from the LLM given the language goal g. We then use the VLM to eliminate subgoals from W that have been completed. For each subgoal, we question the VLM whether that subgoal has been completed. For example, consider the subgoal "Place white block in yellow bowl". To see if the subgoal has been completed, we ask the VLM "Is there a block in yellow bowl?". Consider the subgoal "Place green block in brown box" as another example. To see if the subgoal has been completed, we ask the VLM "Is there a green block in brown box?". Furthermore, if the VLM says "yes" and the subgoal has been completed, we also remove other subgoals from W that should have been completed, such as "Place white block in green bowl". Once we have eliminated the completed subgoals, we use the domain knowledge to determine which subgoal to execute out of all the remaining subgoals. As an example, if the goal is to "Stack green block on top of blue block in brown box" and we have a green block in green bowl and a blue block in blue bowl, we should execute the subgoal "Place blue block in brown box" before the subgoal "Place green block on blue block". While this process of asking questions from VLM to determine the remaining subgoals and then sequencing the remaining subgoals doesn't require any training data, it heavily relies on the task's domain knowledge.

Failure Modes of VLM. We observe two common failure modes of the VLM approach in `object-arrange` environment and visualize them in Figure 8. First, because the model is not trained on any in-domain data, it often fails to recognize uncommon objects, such as computer hard drives, in the observations. Second, it occasionally hallucinates the presence of objects at certain locations and thus leads to incorrect visual reasoning.

VLM as a Subgoal Predictor. We also tried to prompt the VLM with 5 examples of goal g and subgoal candidates $\{w_i\}_{i=1}^M$ and then directly use it to generate the next subgoal w_i given the observation $x_{i,1}$ and the goal g. However, it completely failed. We hypothesize that the VLM fails to directly generate the next subgoal due to its inability to perform in-context learning.

Evaluation. We evaluate the trained classifier f_ϕ and the frozen VLM for subgoal prediction accuracy on 5k unseen datapoints, consisting of observation, goal, candidate subgoals and correct subgoal, generated from test tasks $\mathcal{T}_{\text{test}}$. We average over 4 seeds and show the results in Figure 7.

C.2 Visual Planning

Ego4D dataset processing. We pre-train on *canonical clips* of the Ego4D dataset which are text-annotated short clips made from longer videos. We further divide each canonical clip into 10sec segments from which we derive 50 frames. We resize each frame to 48×64. We create a pretraining Ego4D dataset of (approximately) 344k short clips, each consisting of 50 frames and a text annotation. We use the loader from R3M [33] codebase (https://github.com/facebookresearch/r3m) to load our pretraining Ego4D dataset.

Training Objective and Dataset. We use pixel-level L1 reconstruction and negative perceptual similarity for training the autoencoder v_{enc}. We borrow this objective from PVDM [49] paper except we don't use adversarial loss. We keep the language encoder frozen. We use denoising loss in video latent space $\mathbb{E}_{k\sim[1,K],\tau_z,w,x\sim\mathcal{D},\epsilon\sim\mathcal{N}(O,I)}[\|\epsilon - \epsilon_\phi((\tau_z)_k, l_{\text{enc}}(w), v_{\text{enc}}(x), k)\|^2]$ to train the noise model ϵ_ϕ. We replace w with a null token so that ϵ_ϕ learns both a text-conditional model and an unconditional model. We pretrain the autoencoder v_{enc} and the noise model ϵ_ϕ on the processed Ego4D dataset. We then finetune it on our dataset $\mathcal{D}_{\text{video}} := \{\tau_x^i, w_i\}$ consisting of approximately 100k observation trajectories of length $T = 50$ and associated text subgoals.

Classifier Training Objective and Dataset. We use a binary cross-entropy loss to train the binary classifier $g_\psi(\tau_z)$ that predicts if the observation trajectory in latent space $\tau_z = v_{\text{enc}}(\tau_x)$ leads to high-likelihood action trajectory. It is trained using trajectories from video dataset $\tau_x \sim \mathcal{D}_{\text{video}}$.

C.3 Action Planning

Training Objective and Dataset. We train inverse dynamics p_ψ on a dataset $\mathcal{D}_{\text{inv}}$. Since actions are differences between robotic joint states, we train p_ψ to directly predict robotic state $s_{i,t} = p_\psi(x_{i,t})$ by minimizing the mean squared error between the predicted robotic state and ground truth robotic

state. Hence, $\mathcal{D}_{\text{inv}} \coloneqq \{\tau_x^i, \tau_s^i\}$ consists of 1k paired observation and robotic state trajectories, each having a length of $T = 50$.

Evaluation. We evaluate the trained p_ψ (i.e., VC-1 initialized model and other related models in Figure 6) on 100 unseen paired observation and robotic state trajectories generated from test tasks $\mathcal{T}_{\text{test}}$. We use mean squared error to evaluate our inverse dynamics models. We use 4 seeds to calculate the standard error, represented by the shaded area in Figure 6.

C.4 Hyperparameters

Task Planning. We train f_ϕ for 50 epochs using AdamW optimizer [30], a batch size of 256, a learning rate of $1e-3$ and a weight decay of $1e-6$. We used one V100 Nvidia GPU for training the multi-class classifier.

Visual Planning. We borrow our hyperparameters for training video diffusion from the PVDM paper [49]. We use AdamW optimizer [30], a batch size of 24 and a learning rate of $1e-4$ for training the autoencoder. We use AdamW optimizer, a batch size of 64, and a learning rate of $1e-4$ for training the noise model. During the pretraining phase with the Ego4D dataset, we train the autoencoder for 5 epochs and then the noise model for 5 epochs. During the finetuning phase with $\mathcal{D}_{\text{video}}$, we train the autoencoder for 10 epochs and then the noise model for 40 epochs. We used two A6000 Nvidia GPUs for training these diffusion models. We train g_ψ for 10 epochs using AdamW optimizer, a batch size of 256 and a learning rate of $1e-4$. We used one V100 Nvidia GPU for training the binary classifier. During classifier-free guidance, we use $\omega = 4$ and $\omega' = 1$.

Action Planning. We train VC-1 initialized inverse dynamics model for 20 epochs with AdamW optimizer [30], a batch size of 256 and a learning rate of $3e-5$. We trained other randomly initialized ViT-B inverse dynamics models and randomly initialized ResNet-18 inverse dynamics models for 20 epochs with AdamW optimizer, a batch size of 256, and a learning rate of $1e-4$. We used one V100 Nvidia GPU for training these inverse dynamics models.

D Additional Ablation Studies

D.1 Consistency between task planning and visual planning

To make task planning consistent with visual planning, we need to select subgoal w_i^* which maximizes the joint likelihood (see equation 3) of LLM $p_{\text{LLM}}(w_i|g)$ and video diffusion $p_\phi(\tau_x^i|w_i, x_{i,1})$. While generating videos for different subgoal candidates w_i and calculating the likelihood of the generated video is computationally expensive, we would still like to evaluate its performance in subgoal prediction given it is theoretically grounded. To this end, we first sample M subgoals $W = \{w_j\}_{j=1}^M$ from the LLM. Then, we calculate $w_i^* = \arg\max_{w \in W} \log p_\phi(\tau_x^i|w, x_{i,1})$ and use w_i^* as our predicted subgoal. Since $\log p_\phi(\tau_x^i|w, x_{i,1})$ is intractable, we estimate its variational lower-bound as an approximation. We use this approach for subgoal prediction in `paint-block` environment and compare its performance to that of the learned classifier. It achieves a subgoal prediction accuracy of $54.3 \pm 7.2\%$ whereas the learned classifier achieves a subgoal prediction accuracy of $98.2 \pm 1.5\%$ in `paint-block` environment. Both approaches outperform the approach of randomly selecting a subgoal from W (i.e., no task plan refinement), which yields a subgoal prediction accuracy of 16.67% given $M = 6$. The poor performance of the described approach could result from the fact that the diffusion model only coarsely approximates the true distribution $p(\tau_x^i|w_i, x_{i,1})$, which results in loose variational lower-bound and thus uncalibrated likelihoods from the diffusion model. A larger diffusion model could better approximate $p(\tau_x^i|w_i, x_{i,1})$, resulting in tighter variational lower-bound and better-calibrated likelihoods.

D.2 Consistency between visual planning and action planning

To make visual planning consistent with action planning, we need to select observation trajectory $(\tau_x^i)^*$ which maximizes joint likelihood (see equation 4) of conditional video diffusion $p_\phi(\tau_x^i|w_i, x_{i,1})$ and inverse model $\prod_{t=1}^{T-1} p_\psi(a_{i,t}|x_{i,t}, x_{i,t+1})$. While sampling action trajectories and calculating their likelihoods during every step of the denoising process is computationally inefficient, we would still like to evaluate its effectiveness in visual plan refinements. However, we perform video diffusion

in latent space while our inverse model is in observation space. Hence, for purpose of this experiment, we learn another inverse model $\overline{p}_{\overline{\psi}}(\tau_a^i|\tau_z^i)$ that uses a sequence model (i.e. a transformer) to produce an action trajectory τ_a^i given an observation trajectory in latent space τ_z^i. We train $\overline{p}_{\overline{\psi}}$ for 20 epochs on 10k paired observation and action trajectories, each having a length of $T = 50$. We use AdamW optimizer, a batch size of 256 and a learning rate of $1e-4$ during training. To generate an observation trajectory that maximizes the joint likelihood, we first sample 30 observation trajectories from video diffusion $p_\phi(\tau_x^i|w_i, x_{i,1})$ conditioned on subgoal w_i and observation $x_{i,1}$. For each generated observation trajectory τ_x^i, we sample a corresponding action trajectory τ_a^i and calculate its corresponding log-likelihood $\log \overline{p}_{\overline{\psi}}(\tau_a^i|v_{\text{enc}}(\tau_x^i))$. We select the observation trajectory τ_x^i with highest log-likelihood. Note that we only use $\overline{p}_{\overline{\psi}}$ for visual plan refinement and use p_ψ for action execution to ensure fair comparison. If we use this approach for visual plan refinement with `HiP`, we obtain a success rate of 72.5 ± 1.9 on unseen tasks in `paint-block` environment. This is comparable to the performance of `HiP` with visual plan refinements from learned classifier g_ψ which obtains a success rate of 72.8 ± 1.7 on unseen tasks in `paint-block` environment. In contrast, `HiP` without any visual plan refinement obtains a success rate of 71.1 ± 1.3 on unseen tasks in `paint-block` environment. These results show that g_ψ serves as a good approximation for estimating whether an observation trajectory leads to a high-likelihood action trajectory, while still being computationally efficient.

Architecture for $\overline{p}_{\overline{\psi}}$. We use a transformer model to represent $\overline{p}_{\overline{\psi}}(\tau_a|\tau_z = [\tau_z^T, \tau_z^H, \tau_z^W])$. We first use a ResNet-9 encoder to convert τ_z^T, τ_z^H, and τ_z^W to latent vectors. We then concatenate those latent vectors and project the resulting vector to a hidden space of 64 dimension using a linear layer. We then pass the 64 dimensional vector to a trajectory transformer model [24] which generates an action trajectory τ_a of length 50. The trajectory transformer uses a transformer architecture with 4 layers and 4 self-attention heads.