

Steering Multi-Task Diffusion Policies from Symbolic Plans

Lenkung von Multitask-Diffusionsmodellen durch symbolische Pläne

Bachelor thesis by Jan Gröpper

Date of submission: October 8, 2026

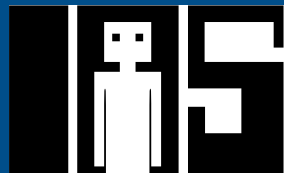
1. Review: Claudius Kienle

2. Review: Jan Peters

Darmstadt



TECHNISCHE
UNIVERSITÄT
DARMSTADT



Erklärung zur Abschlussarbeit gemäß §22 Abs. 7 und §23 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Jan Gröpper, die vorliegende Bachelorarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Fall eines Plagiats (§38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung gemäß §23 Abs. 7 APB überein.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Darmstadt, 8. Oktober 2026



J. Gröpper

Abstract

Task and Motion Planning (TAMP) combines symbolic task planning with continuous robot motion generation. While symbolic planners can reason over long sequences of actions, the corresponding robot motions are typically generated by motion generators, step by step. Diffusion models can represent multiple manipulation behaviors and have been combined with symbolic planning and state constraints. However, it remains insufficiently explored whether a single unconditional multi-task trajectory model can be steered by modular predicates that describe only a desired symbolic environment state, without requiring the predicates to classify complete robot trajectories or retraining the diffusion model for each new steering signal. In this thesis, we investigate such an interface between symbolic planning and diffusion-based motion generation. The diffusion model jointly generates trajectories of robot states and latent representations of depth observations. We first learn the depth representation and investigate how its information content affects downstream diffusion sampling. We then study a flexible interface to symbolic planning, where we guide an unconditional diffusion model with predicate classifiers that describe desired environment states. We propagate predicate classifier gradients through the denoiser, allowing a classifier that observes only the terminal environment state to influence the complete generated trajectory. In our controlled manipulation environment, guidance from only the terminal latent representation increases task success from 48.4% without guidance to 96.0%.

Zusammenfassung

Die Aufgaben- und Bewegungsplanung (Task and Motion Planning, TAMP) verbindet symbolische Aufgabenplanung mit der Erzeugung kontinuierlicher Roboterbewegungen. Während symbolische Planer auch über lange Aktionssequenzen schlussfolgern können, werden die zugehörigen Roboterbewegungen typischerweise schrittweise durch Bewegungsgeneratoren erzeugt. Diffusionsmodelle können unterschiedliche Manipulationsverhalten in einem gemeinsamen Modell abbilden und wurden bereits mit symbolischer Planung und Zustandsbedingungen kombiniert. Bislang ist jedoch nur unzureichend untersucht, ob sich ein einziges unkonditioniertes Multi-Task-Trajektorienmodell mithilfe modularer Prädikate steuern lässt, die lediglich einen gewünschten symbolischen Umgebungszustand beschreiben, ohne dass diese Prädikate vollständige Robotertrajektorien klassifizieren müssen oder das Diffusionsmodell für jedes neue Steuerungssignal erneut trainiert werden muss. In dieser Arbeit untersuchen wir eine solche Schnittstelle zwischen symbolischer Planung und diffusionsbasierter Bewegungsgenerierung. Das Diffusionsmodell erzeugt gemeinsam Trajektorien von Roboterzuständen und latente Repräsentationen von Tiefenbildern. Zunächst lernen wir die Repräsentation der Tiefenbilder und untersuchen, wie sich deren Informationsgehalt auf das nachgelagerte Sampling des Diffusionsmodells auswirkt. Anschließend untersuchen wir eine flexible Schnittstelle zur symbolischen Planung, bei der ein unkonditioniertes Diffusionsmodell durch Prädikatklassifikatoren gesteuert wird, die gewünschte Umgebungszustände beschreiben. Dazu propagieren wir die Gradienten der Prädikatklassifikatoren durch den Denoiser, sodass ein Klassifikator, der ausschließlich den abschließenden Umgebungszustand betrachtet, die gesamte generierte Trajektorie beeinflussen kann. In unserer kontrollierten Manipulationsumgebung steigert die Steuerung allein anhand der abschließenden latenten Repräsentation die Erfolgsrate von 48,4 % ohne Steuerung auf 96,0 %.

Contents

1. Introduction	2
2. Foundations	4
2.1. Representation Learning	4
2.2. Generative Modeling with Diffusion	5
2.3. Neuro-Symbolic Task and Motion Planning	20
3. Related Work	21
3.1. Representation Learning	21
3.2. Diffusion Trajectory Generation	21
3.3. Diffusion Guidance	23
3.4. Task and Motion Planning	23
4. Methodology	25
4.1. Latent Environment Representations	27
4.2. Diffusion Trajectory Sampler	30
4.3. Guidance from Symbolic Plans	33
5. Experiments	36
5.1. Environments and Data	36
5.2. Evaluation Metrics	40
5.3. Latent Representations	40
5.4. Trajectory Sampling	52
5.5. Predicate Guidance	54
6. Discussion	59
7. Summary	62
8. Outlook	63

A. Network Architectures	70
A.1. Notation and Building Blocks	70
A.2. Stage 1: Latent Representation Learning	72
A.3. Stage 2: Diffusion Trajectory Sampler	77
A.4. Predicate Classifiers	79
A.5. CALVIN State Estimator	81
B. Hyperparameters	83
B.1. CALVIN Latent Representation Experiments	83
B.2. Predicate Guidance	88

1. Introduction

Real-world robotics tasks can often be decomposed into a sequence of sub-tasks. For example, a robot tasked with assembling a piece of furniture must perform many different sub-tasks such as picking up objects, moving them to a new location, and inserting them into the correct position. Importantly, those sub-tasks are often highly dependent on each other and cannot be performed in arbitrary order. Thus, accomplishing a task like this requires planning over sequences of actions that lead to a desired end state. While modern control policies are highly capable of performing short-horizon robot movements, they lack the ability to reason over long sequences of different behaviors. Neuro-symbolic TAMP combines high-level symbolic task planning with continuous motion planning.

A symbolic planner represents the state of the environment through predicates and generates a sequence of symbolic actions that transition between symbolic states. A predicate is a Boolean function over objects in the environment that describes a property or relation of the current state. For example, the symbolic action `close(microwave)` may lead to a state in which the predicate `closed(microwave)` is true. The motion planner must generate the continuous robot motion that realizes this transition. Predicates therefore provide a natural interface between symbolic planning and continuous motion generation: the symbolic planner can specify which predicates should hold in the next state, while the motion planner has to generate a trajectory that reaches such a state.

Existing TAMP approaches often associate different symbolic actions with different skills, samplers, or policies [1, 2]. Recent work on diffusion policies instead shows that a single model can represent robot motions for different behaviors [3, 4]. These models are commonly controlled through direct conditioning with text embeddings and lack the long-horizon reasoning capabilities of symbolic planners.

Fang et al. [5] combine diffusion models with symbolic planning and use classifier guidance to satisfy symbolic constraints. However, to steer the diffusion model between different behaviors, the classifier is trained on trajectories to detect entire robot motions

corresponding to a symbolic action. This approach does not consider how a diffusion trajectory generator can be steered directly from symbolic states.

A symbolic planner specifies a desired next state through predicates. Predicates often describe properties of only part of a generated trajectory. For example, whether a microwave is closed can be determined from the final environment state, although reaching this state depends on the complete robot trajectory. Using such predicates to steer the complete trajectory therefore requires the guidance signal to affect parts of the trajectory on which the predicate is not directly evaluated.

To use guidance signals from the environment state, existing methods like DiMSam [5] generate environment observations as part of the trajectory. Fang et al. [5] address the high dimensionality of observations using a learned latent representation. Recent work by Heek et al. [6] shows how the latent representation can be optimized for downstream generative modeling by controlling the information content of the learned latent space. How this trade-off affects diffusion-based robot trajectory generation has not been thoroughly investigated.

In this thesis, we focus on the motion planning component of TAMP and assume that a symbolic planner provides a sequence of symbolic actions and states. We investigate a single diffusion model that generates trajectories of environment and robot states and can be steered by the symbolic action or by predicates describing the desired next symbolic state. We further investigate guidance from predicates that depend only on part of the generated trajectory. Finally, we study how controlling the information content of a learned environment representation affects trajectory generation.

Chapter 2 introduces the foundational concepts used throughout this thesis. Chapter 3 discusses related work in the areas of neuro-symbolic TAMP, diffusion policies, as well as representation learning and guidance of diffusion models. Chapter 4 describes the proposed representation learning approach, motion planning model and guidance methods. Chapter 5 presents the experimental setup and results. Finally, Chapters 6, 7, and 8 discuss the results and conclude the thesis, with possible future directions.

2. Foundations

2.1. Representation Learning

2.1.1. Latent Representations

Let $x \in \mathbb{R}^d$ and $f_{\text{enc}} : \mathbb{R}^d \rightarrow \mathbb{R}^p$ be a function, which we call *encoder*. Then we call $z = f_{\text{enc}}(x)$ a *latent representation* of x . We also call z a *latent variable*. Typically, the encoder is a parametric function $f_{\text{enc}, \theta}$ with parameters $\theta \in \mathbb{R}^k$. The encoder does not need to be a function. We often consider latent representations as random variables z that are distributed according to a distribution $q_{\theta}(z|x)$, which is parametrized by θ . In this case, we call the conditional distribution q_{θ} the encoder.

In a machine learning setting, we are usually not interested in the latent representation itself but how the information it contains about a sample x can be used to solve downstream tasks. A general intuition for why we would use latent representations instead of original data points is that a latent variable could be a more compact or less noisy representation of the data. And that it could therefore improve the performance or efficiency of downstream machine learning approaches.

The concept of a latent variable is therefore similar to the concept of features in classical machine learning methods. In classical machine learning algorithms, like Support Vector Machines (SVMs) [7], the data is mapped into a feature space in which it becomes (linearly) separable. However, features in traditional machine learning methods are usually handcrafted. Whereas with latent variables, we emphasize that they are learned. Latent representations are usually learned with a neural network. Often, the latent representation is learned separately from the downstream tasks. Learning a latent representation means optimizing the parameters of the encoder.

Most of the time, we are interested in learning latent representations whose dimensionality is smaller than the dimensionality of the original data x , i.e., $p < d$. Thus we can view

$f_{\text{enc}, \theta}$ as a function that compresses x into z . For compression, we want to reduce p as much as possible while keeping as much information of x as possible. A natural way to optimize the encoder for this is by defining a second function $f_{\text{dec}} : \mathbb{R}^p \rightarrow \mathbb{R}^d$, which we call *decoder*, which should invert the latent variable. In the probabilistic setting, the decoder is a distribution $p_{\theta}(x|z)$, where θ are the parameters of the decoder. We then optimize the encoder and decoder jointly, where the decoder has to predict the input of the encoder. We emphasize that this is an unsupervised learning objective, since it uses no labels. The basic idea, that we could learn useful representations with neural networks by optimizing for reconstruction while constraining the dimensionality of the intermediate representations, was established early on in the development of neural networks [8, 9].

2.2. Generative Modeling with Diffusion

2.2.1. Generative Modeling

Let $\mathcal{D} \subseteq \mathbb{R}^d$ be a set of data points $x \in \mathcal{D}$. We assume an unknown data distribution $p_{\text{data}}(x)$ exists, from which the data points are drawn. The goal of generative modeling is to find a sampling procedure that generates new samples according to p_{data} .

2.2.2. Discrete Diffusion Processes

A discrete diffusion process defines a Markov chain $z^{(1)}, \dots, z^{(T)} \in \mathbb{R}^d$ with joint distribution

$$q(z^{(1:T)} | x) = q(z^{(1)} | x) \prod_{t=1}^{T-1} q(z^{(t+1)} | z^{(t)}). \quad (2.1)$$

The marginal distributions conditioned on the clean data x are

$$q(z^{(t)} | x) = \mathcal{N}\left(z^{(t)}; \alpha_t x, \sigma_t^2 I\right), \quad t = 1, \dots, T. \quad (2.2)$$

The corresponding Gaussian Markov transitions are chosen as

$$q\left(z^{(t+1)} | z^{(t)}\right) = \mathcal{N}\left(z^{(t+1)}; \frac{\alpha_{t+1}}{\alpha_t} z^{(t)}, \left[\sigma_{t+1}^2 - \frac{\alpha_{t+1}^2}{\alpha_t^2} \sigma_t^2\right] I\right), \quad t = 1, \dots, T-1. \quad (2.3)$$

The sequences α and σ are defined over finite time steps $t = 1, \dots, T$. α is monotonically decreasing and σ is monotonically increasing. The combination of α and σ is referred to as the *noise schedule*. We define $\lambda_t = \log(\alpha_t^2/\sigma_t^2)$ as the log signal-to-noise ratio (SNR) of step t .

We typically assume that $\alpha_1 \approx 1$ and $\sigma_1 \approx 0$, such that $z^{(1)}$ is close to x . And we choose $\alpha_T \approx 0$ and $\sigma_T \approx 1$, such that the distribution of $z^{(T)}$ is approximately standard Gaussian.

Applying the reparameterization trick to Equation 2.2, we obtain

$$z^{(t)} = \alpha_t x + \sigma_t \epsilon^{(t)} \quad \epsilon^{(t)} \sim \mathcal{N}(0, I). \quad (2.4)$$

These equations reveal why we call α and σ the noise schedule and also what the general idea behind diffusion models is, as introduced by Sohl-Dickstein et al. [10]. Starting from a raw observation $x \in \mathcal{D}$, $z^{(t)}$ are noisy versions of x , that become increasingly noisy over time, until they are indistinguishable from random noise. We call this procedure of computing specific values $z^{(t)}$ for a given x the *noising* or *diffusion process* [11].

Learning a model that reverses the noising process from $z^{(T)}$ to x would mean we could generate samples from random noise. We call this reverse process the *denoising process* and the model that performs a *diffusion model* or *denoiser*. Mathematically, we define the reverse process as the joint distribution $p_\theta(x, z^{(1)}, \dots, z^{(T)})$, where we assume [11]

$$p_\theta(x, z^{(1)}, \dots, z^{(T)}) = p(z^{(T)}) p_\theta(x | z^{(1)}) \prod_{t=2}^T p_\theta(z^{(t-1)} | z^{(t)}) \quad (2.5)$$

$$\begin{aligned} \text{with} \quad & p_\theta(z^{(T)}) = \mathcal{N}(z^{(T)}; 0, I), \\ & p_\theta(x | z^{(1)}) = \mathcal{N}(x; \mu_\theta(z^{(1)}, 1), \sigma_\theta(z^{(1)}, 1)^2 I), \\ & p_\theta(z^{(t-1)} | z^{(t)}) = \mathcal{N}(z^{(t-1)}; \mu_\theta(z^{(t)}, t), \sigma_\theta(z^{(t)}, t)^2 I). \end{aligned}$$

Spelled out, this means we assume the reverse process to be Gaussian and markovian, and $z^{(T)}$ to be standard normal. These are reasonable assumptions because they follow from the definition of the forward noising process when we consider arbitrary small time steps [10, 12].

Maximum Likelihood Training

To learn the parameters θ of the reverse process, we can use a maximum likelihood approach, by maximizing a lower bound on the log-likelihood of the data distribution

$p_{\text{data}}(x)$ [10]:

$$\begin{aligned} \log p_{\theta}(x) &\geq \mathbb{E}_{q(z^{(1)}|x)} \left[\log p_{\theta}(x | z^{(1)}) \right] - D_{\text{KL}} \left(q(z^{(T)} | x) \parallel p(z^{(T)}) \right) \\ &\quad - \sum_{t=2}^T \mathbb{E}_{q(z^{(t)}|x)} \left[D_{\text{KL}} \left(q(z^{(t-1)} | z^{(t)}, x) \parallel p_{\theta}(z^{(t-1)} | z^{(t)}) \right) \right] \end{aligned} \quad (2.6)$$

All distributions denoted by q can be derived by the fixed forward process and contain no learned parameters (beside an optionally learned noise schedule). The marginal $q(z^{(t)} | x)$ describes the distribution of the noisy data at step t , while $q(z^{(t-1)} | z^{(t)}, x)$ is the posterior distribution of the previous state given the current state and the original data. Because the forward process is linear and Gaussian, both distributions are available in closed form. In particular, $q(z^{(t-1)} | z^{(t)}, x) = \mathcal{N}(z^{(t-1)}; \tilde{\mu}(z^{(t)}, x, t), \tilde{\sigma}^2(t)I)$ is Gaussian with the mean dependent on the noisy data $z^{(t)}$, clean data x and time step t . The variance $\tilde{\sigma}^2(t)$ is only dependent on the time step t . The closed form expressions for $\tilde{\mu}$ and $\tilde{\sigma}^2$ are given by

$$\tilde{\mu}(z^{(t)}, x, t) = \frac{\alpha_t \sigma_{t-1}^2}{\alpha_{t-1} \sigma_t^2} z^{(t)} + \frac{\alpha_{t-1}}{\sigma_t^2} \left(\sigma_t^2 - \frac{\alpha_t^2}{\alpha_{t-1}^2} \sigma_{t-1}^2 \right) x \quad (2.7)$$

$$\tilde{\sigma}^2(t) = \frac{\sigma_{t-1}^2}{\sigma_t^2} \left(\sigma_t^2 - \frac{\alpha_t^2}{\alpha_{t-1}^2} \sigma_{t-1}^2 \right). \quad (2.8)$$

For an in-depth derivation of this lower bound, we refer to Luo [13].

The terms in Equation 2.6 have intuitive interpretations: the first term is a reconstruction loss from the least noisy step $z^{(1)}$ to the original data x . The second term measures how close our model's distribution of the most noisy step $z^{(T)}$ is to the standard normal prior. The third term measures how much the distribution of any $z^{(t)}$ in the forward process diverges from the denoising process. We call the third term the *denoising loss*. Since we want to generate samples with high likelihood, we want to maximize the lower bound in Equation 2.6. Thus, we have to

- maximize the reconstruction log-likelihood $\mathbb{E}_{q(z^{(1)}|x)} [\log p_{\theta}(x | z^{(1)})]$
- minimize the prior KL divergence $D_{\text{KL}}(q(z^{(T)} | x) \parallel p(z^{(T)}))$
- minimize the denoising loss $\sum_{t=2}^T \mathbb{E}_{q(z^{(t)}|x)} [D_{\text{KL}}(q(z^{(t-1)} | z^{(t)}, x) \parallel p_{\theta}(z^{(t-1)} | z^{(t)}))]$

$q(z^{(t-1)} | z^{(t)}, x)$ is Gaussian with mean $\tilde{\mu}(z^{(t)}, x, t)$ and variance $\tilde{\sigma}^2(t)I$. If we set the variance of the denoising distribution $\sigma_\theta(z^{(t)}, t)^2 = \tilde{\sigma}^2(t)$, the diffusion model only parameterizes the mean of the denoising distribution. And since $z^{(t)}$ and t are known, the diffusion model has to predict the clean data x , where the mean of the denoising distribution then becomes $\tilde{\mu}(z^{(t)}, \hat{x}_\theta(z^{(t)}, t), t)$. The denoising loss term in Equation 2.6 can then be substantially simplified, where minimizing the KL divergence at each denoising step reduces to minimizing

$$\frac{1}{2} \left(e^{\lambda_{t-1}} - e^{\lambda_t} \right) \left\| \hat{x}_\theta(z^{(t)}, t) - x \right\|_2^2 \quad (2.9)$$

where $\hat{x}_\theta(z^{(t)}, t)$ is the estimate of the clean data point x , based on the current denoising step t , the noise variable $z^{(t)}$, and model parameters θ . This simplification is possible because the distributions q and p_θ are both Gaussian and only the mean of q being dependent on the clean data and current noise variable. For a comprehensive derivation of this result, we again refer to Luo [13].

Using Equation 2.9 as a loss function means the diffusion model is trained to predict clean data points x , given noisy samples $z^{(t)} \sim q(\cdot | x)$ and time step t as inputs, where t is uniformly sampled during training.

It is also possible to avoid the simplification of setting $\sigma_\theta(z^{(t)}, t)^2 = \tilde{\sigma}^2(t)$. Nichol et al. [14] learn the variance alongside the mean and show an improved tradeoff between sample quality and number of denoising steps.

Equivalent Diffusion Parameterizations

We can derive two equivalent parameterizations of the denoising objective given in Equation 2.9, which were first introduced by Ho et al. [11]. We refer to the parameterization in Equation 2.9 as the x_0 parameterization.

By rearranging Equation 2.4 we obtain

$$z^{(t)} = \alpha_t x + \sigma_t \epsilon^{(t)} \iff \epsilon^{(t)} = \frac{z^{(t)} - \alpha_t x}{\sigma_t}. \quad (2.10)$$

Instead of predicting x directly, the model can therefore predict the noise $\epsilon^{(t)}$. The corresponding clean-data prediction is

$$\hat{x}_\theta(z^{(t)}, t) = \frac{z^{(t)} - \sigma_t \hat{\epsilon}_\theta(z^{(t)}, t)}{\alpha_t}. \quad (2.11)$$

Consequently,

$$\hat{x}_\theta(z^{(t)}, t) - x = \frac{\sigma_t}{\alpha_t} \left(\epsilon^{(t)} - \hat{\epsilon}_\theta(z^{(t)}, t) \right). \quad (2.12)$$

Substitution into 2.9 yields the equivalent noise-prediction least-squares error with its noise-schedule-dependent weight:

$$\frac{1}{2} \left(\frac{\alpha_{t-1}^2 \sigma_t^2}{\sigma_{t-1}^2 \alpha_t^2} - 1 \right) \left\| \hat{\epsilon}_\theta(z^{(t)}, t) - \epsilon^{(t)} \right\|_2^2. \quad (2.13)$$

Equivalently, using the log-SNR, the coefficient is $\frac{1}{2} (e^{\lambda_{t-1} - \lambda_t} - 1)$.

We also refer to the noise-prediction parameterization as the ϵ parameterization. Even though theoretically equivalent, ϵ parameterization often performs better in practice than x_0 parameterization [11, 15].

A third equivalent parameterization predicts the score of the noisy marginal

$$q_t(z^{(t)}) = \int q(z^{(t)} | x) p_{\text{data}}(x) dx. \quad (2.14)$$

Tweedie’s formula [16] relates the score of this marginal to the conditional mean of the added noise, with

$$\nabla_{z^{(t)}} \log q_t(z^{(t)}) = -\frac{1}{\sigma_t} \mathbb{E} \left[\epsilon^{(t)} | z^{(t)} \right]. \quad (2.15)$$

A noise-prediction model therefore also defines a score model $s_\theta(z^{(t)}, t) = -\hat{\epsilon}_\theta(z^{(t)}, t)/\sigma_t$, where $s_\theta(z^{(t)}, t)$ denotes the estimate of the score $\nabla_{z^{(t)}} \log q_t(z^{(t)})$.

Conversely, the clean-data estimate can be recovered from the predicted score using Tweedie’s formula:

$$\hat{x}_\theta(z^{(t)}, t) = \frac{z^{(t)} + \sigma_t^2 s_\theta(z^{(t)}, t)}{\alpha_t}. \quad (2.16)$$

We refer to this third parameterization as the *score* parameterization. In practice, the equivalence to noise prediction allows the score to be learned from synthetically corrupted training samples without evaluating q_t explicitly. The resulting estimate can be used directly by score-based samplers and modified to implement guidance [17].

2.2.3. Continuous-Time Diffusion

So far, we defined the diffusion process over a finite number of discrete time steps. In the following, we instead use a continuous diffusion time $t \in [0, 1]$. The noise schedule is then defined by continuous functions $\alpha(t)$ and $\sigma(t)$. For a clean data point x , the marginal distribution at diffusion time t is

$$q\left(z^{(t)} \mid x\right) = \mathcal{N}\left(z^{(t)}; \alpha(t)x, \sigma(t)^2 I\right). \quad (2.17)$$

Equivalently, we can sample from this distribution using

$$z^{(t)} = \alpha(t)x + \sigma(t)\epsilon, \quad \epsilon \sim \mathcal{N}(0, I). \quad (2.18)$$

As in the discrete case, $\alpha(t)$ decreases and $\sigma(t)$ increases with diffusion time. We choose the endpoints such that $\alpha(0) \approx 1$ and $\sigma(0) \approx 0$, while $\alpha(1) \approx 0$ and $\sigma(1) \approx 1$. Thus $z^{(0)}$ is approximately the clean data, and $z^{(1)}$ is approximately standard Gaussian noise.

For two arbitrary diffusion times $s, t \in [0, 1]$ with $s < t$, the corresponding forward transition is

$$q\left(z^{(t)} \mid z^{(s)}\right) = \mathcal{N}\left(z^{(t)}; \alpha_{t|s}z^{(s)}, \sigma_{t|s}^2 I\right), \quad (2.19)$$

where

$$\alpha_{t|s} = \frac{\alpha(t)}{\alpha(s)} \quad (2.20)$$

and

$$\sigma_{t|s}^2 = \sigma(t)^2 - \frac{\alpha(t)^2}{\alpha(s)^2} \sigma(s)^2. \quad (2.21)$$

This is the continuous-time equivalent of the Gaussian transition between two discrete diffusion steps. An important difference is that s and t can now be arbitrary points in $[0, 1]$ instead of indices of a fixed set of diffusion steps. During training, we can therefore sample arbitrary diffusion times, while at inference time we can choose the sequence of diffusion times used for sampling.

Similar to before, we define the log-SNR as

$$\lambda(t) = \log \frac{\alpha(t)^2}{\sigma(t)^2}. \quad (2.22)$$

The log-SNR gives a convenient way to describe the noise level with a single function. For high values of $\lambda(t)$, the noisy sample contains mostly signal, while for low values it contains mostly noise. Since $\alpha(t)$ decreases and $\sigma(t)$ increases, $\lambda(t)$ is monotonically decreasing.

Variance Preservation

In the variance-preserving parameterization of the diffusion process [18], we impose an additional constraint on α and σ , so

$$\alpha^2(t) + \sigma^2(t) = 1, \quad \forall t \in [0, 1]. \quad (2.23)$$

Under this constraint, we only have to specify one of the two functions α or σ , since $\sigma(t)^2 = 1 - \alpha^2(t)$ and $\alpha(t)^2 = 1 - \sigma^2(t)$.

With the variance-preservation constraint, we can derive the functions α and σ from a given log-SNR function $\lambda(t)$. By using the definition of λ in Equation 2.22 and the constraint in Equation 2.23, we obtain

$$\sigma(t)^2 = \frac{1}{1 + e^{\lambda(t)}} = \text{sigmoid}(-\lambda(t)) \quad (2.24)$$

$$\alpha(t)^2 = 1 - \frac{1}{1 + e^{\lambda(t)}} = \frac{1}{1 + e^{-\lambda(t)}} = \text{sigmoid}(\lambda(t)) \quad (2.25)$$

This alone does, however, not guarantee that $\alpha(t)$ and $\sigma(t)$ are monotonically decreasing and increasing, respectively. But since the sigmoid function is monotonically increasing, we can ensure the monotonicity of α and σ by requiring that $\lambda(t)$ is monotonically decreasing.

2.2.4. Sampling from Diffusion Models

So far, we have discussed what a diffusion model is and how to train it. In this section, we look at how to use a trained diffusion model to generate new samples. We have seen that the forward diffusion process iteratively corrupts a clean data sample x into noisy samples $z^{(t)}$ until it is indistinguishable from random noise. The diffusion model is then trained to reverse this process. So to generate new samples, we can start from a random noise sample $z^{(T)} \sim \mathcal{N}(0, I)$ and iteratively denoise it until we obtain a clean data sample. This is the basic idea behind sampling from diffusion models [10].

Ancestral Sampling

As described above, we can sequentially sample from the reverse process defined in Equation 2.5, to obtain a clean sample. Since the reverse process is Markovian, we first sample the most noisy variable from the prior

$$z^{(T)} \sim p(z^{(T)}) = \mathcal{N}(0, I), \quad (2.26)$$

and subsequently sample

$$z^{(t-1)} \sim p_\theta(z^{(t-1)} | z^{(t)}), \quad t = T, \dots, 2. \quad (2.27)$$

Finally, a clean sample is obtained from $p_\theta(x | z^{(1)})$. Sampling sequentially from the conditional distributions of the reverse process is commonly referred to as *ancestral sampling* [10, 11].

Since the individual reverse transitions are Gaussian,

$$p_\theta(z^{(t-1)} | z^{(t)}) = \mathcal{N}\left(z^{(t-1)}; \mu_\theta(z^{(t)}, t), \sigma_\theta(z^{(t)}, t)^2 I\right), \quad (2.28)$$

each sampling step can be implemented using the reparameterization

$$z^{(t-1)} = \mu_\theta(z^{(t)}, t) + \sigma_\theta(z^{(t)}, t)\epsilon, \quad \epsilon \sim \mathcal{N}(0, I). \quad (2.29)$$

Therefore, ancestral sampling is stochastic, since a new Gaussian noise sample is introduced at every reverse transition.

So far, there is actually a gap in our discussion. We defined the reverse process in terms of its mean μ_θ . But we saw before in Section 2.2.2 that the optimization of the reverse process is done by predicting the clean data sample x from the noisy sample $z^{(t)}$ (or equivalently, the noise sample ϵ or the score $s_\theta(z^{(t)}, t)$), but not the reverse mean directly. For the clean-data parameterization, the reverse mean can instead be obtained from the predicted clean sample $\hat{x}_\theta(z^{(t)}, t)$.

To see how, we define the forward transition coefficients for $t = 2, \dots, T$ as

$$a_t = \frac{\alpha_t}{\alpha_{t-1}}, \quad (2.30)$$

$$v_t = \sigma_t^2 - a_t^2 \sigma_{t-1}^2. \quad (2.31)$$

Then

$$q(z^{(t)} | z^{(t-1)}) = \mathcal{N}\left(z^{(t)}; a_t z^{(t-1)}, v_t I\right). \quad (2.32)$$

Together with the marginal $q(z^{(t-1)} | x)$, this gives the Gaussian posterior

$$q(z^{(t-1)} | z^{(t)}, x) = \mathcal{N}\left(z^{(t-1)}; \tilde{\mu}_t(z^{(t)}, x), \tilde{\sigma}_t^2 I\right), \quad (2.33)$$

where

$$\tilde{\mu}_t(z^{(t)}, x) = \frac{a_t \sigma_{t-1}^2}{\sigma_t^2} z^{(t)} + \frac{\alpha_{t-1} v_t}{\sigma_t^2} x, \quad (2.34)$$

$$\tilde{\sigma}_t^2 = \frac{\sigma_{t-1}^2 v_t}{\sigma_t^2}. \quad (2.35)$$

When the reverse variance is fixed to the posterior variance $\tilde{\sigma}_t^2$, the reverse mean can be parameterized by replacing the unknown clean data x in Equation 2.34 with its model prediction

$$\mu_\theta(z^{(t)}, t) = \tilde{\mu}_t\left(z^{(t)}, \hat{x}_\theta(z^{(t)}, t)\right). \quad (2.36)$$

Consequently, one ancestral denoising step becomes

$$z^{(t-1)} = \tilde{\mu}_t\left(z^{(t)}, \hat{x}_\theta(z^{(t)}, t)\right) + \tilde{\sigma}_t \epsilon, \quad \epsilon \sim \mathcal{N}(0, I). \quad (2.37)$$

For a noise- or score-parameterized diffusion model, the corresponding clean-data estimate can be recovered using the equivalent parameterizations derived in Section 2.2.2.

Ancestral sampling therefore requires evaluating the diffusion model sequentially at every reverse noise level. It samples exactly from the learned generative distribution p_θ . Equation 2.37 shows that every ancestral sampling step is stochastic, which will be important for upcoming discussions.

Denoising Diffusion Implicit Models

Ancestral sampling follows the learned reverse Markov chain and therefore typically requires evaluating the diffusion model at every noise level. Song et al. [19] show that

the denoising model can instead be used with a different class of generative processes, called *Denoising Diffusion Implicit Models* (DDIMs), without retraining the model.

The central observation is that the denoising objective depends on the marginal noisy-data distributions

$$q(z^{(t)} | x) = \mathcal{N}\left(z^{(t)}; \alpha(t)x, \sigma(t)^2 I\right), \quad (2.38)$$

but these marginals do not uniquely determine the joint distribution over different noise levels. DDIM therefore constructs alternative, generally non-Markovian processes that preserve these marginals.

For two noise levels $s < t$, consider the conditional distribution

$$q_\rho(z^{(s)} | z^{(t)}, x) = \mathcal{N}\left(z^{(s)}; \alpha(s)x + \sqrt{\sigma(s)^2 - \rho_{t \rightarrow s}^2} \frac{z^{(t)} - \alpha(t)x}{\sigma(t)}, \rho_{t \rightarrow s}^2 I\right), \quad (2.39)$$

where

$$0 \leq \rho_{t \rightarrow s}^2 \leq \sigma(s)^2 \quad (2.40)$$

controls the stochasticity of the process.

To see that Equation 2.39 preserves the marginal distribution at time s , we use

$$z^{(t)} = \alpha(t)x + \sigma(t)\epsilon, \quad \epsilon \sim \mathcal{N}(0, I), \quad (2.41)$$

and reparameterize Equation 2.39 as

$$z^{(s)} = \alpha(s)x + \sqrt{\sigma(s)^2 - \rho_{t \rightarrow s}^2} \epsilon + \rho_{t \rightarrow s} \xi, \quad \xi \sim \mathcal{N}(0, I). \quad (2.42)$$

Since ϵ and ξ are independent, the total variance of the two noise terms is

$$\sigma(s)^2 - \rho_{t \rightarrow s}^2 + \rho_{t \rightarrow s}^2 = \sigma(s)^2. \quad (2.43)$$

Consequently,

$$q_\rho(z^{(s)} | x) = \mathcal{N}\left(z^{(s)}; \alpha(s)x, \sigma(s)^2 I\right), \quad (2.44)$$

which is identical to the marginal distribution of the original diffusion process. Therefore, the denoising model trained using these marginals can also be used for the alternative process [19].

At sampling time the clean data x in Equation 2.39 is unknown and is replaced by the model prediction

$$\hat{x}_\theta(z^{(t)}, t). \quad (2.45)$$

Equivalently, the noise component of $z^{(t)}$ is estimated as

$$\hat{\epsilon}_\theta(z^{(t)}, t) = \frac{z^{(t)} - \alpha(t)\hat{x}_\theta(z^{(t)}, t)}{\sigma(t)}. \quad (2.46)$$

One generalized DDIM sampling step from t to $s < t$ can therefore be written as

$$z^{(s)} = \alpha(s)\hat{x}_\theta(z^{(t)}, t) + \sqrt{\sigma(s)^2 - \rho_{t \rightarrow s}^2} \hat{\epsilon}_\theta(z^{(t)}, t) + \rho_{t \rightarrow s} \xi, \quad \xi \sim \mathcal{N}(0, I). \quad (2.47)$$

A particularly important choice is

$$\rho_{t \rightarrow s} = 0. \quad (2.48)$$

Equation 2.47 then becomes

$$z^{(s)} = \alpha(s)\hat{x}_\theta(z^{(t)}, t) + \sigma(s)\hat{\epsilon}_\theta(z^{(t)}, t). \quad (2.49)$$

No new random noise is introduced during the denoising step. Thus, after the initial state $z^{(1)}$ has been sampled from the prior, the entire denoising process is deterministic. This deterministic process is the DDIM sampler.

Using

$$\hat{x}_\theta(z_t, t) = \frac{z_t - \sigma(t)\hat{\epsilon}_\theta(z_t, t)}{\alpha(t)}, \quad (2.50)$$

the deterministic DDIM update can equivalently be written as

$$z_s = \frac{\alpha(s)}{\alpha(t)} z_t + \left(\sigma(s) - \frac{\alpha(s)\sigma(t)}{\alpha(t)} \right) \hat{\epsilon}_\theta(z_t, t). \quad (2.51)$$

More generally, the stochasticity can be controlled by setting

$$\rho_{t \rightarrow s} = \eta \tilde{\sigma}_{s|t}, \quad \eta \in [0, 1], \quad (2.52)$$

where $\tilde{\sigma}_{s|t}$ denotes the posterior standard deviation of the corresponding Markov diffusion transition. The choice $\eta = 0$ gives deterministic DDIM sampling, while $\eta = 1$ recovers the variance of the ancestral DDPM transition when the original diffusion steps are used [19].

An important advantage of Equation 2.47 is that s does not need to be the immediately preceding training noise level. Instead, sampling can be performed over a shorter sequence

$$1 = t_N > t_{N-1} > \dots > t_1 > t_0 = 0, \quad (2.53)$$

and Equation 2.47 can be applied directly between consecutive elements of this sequence. Consequently, a diffusion model trained with a large number of noise levels can be sampled using substantially fewer neural-network evaluations.

In the deterministic case, all randomness of the generated sample is therefore contained in the initial latent variable z_1 . The resulting probability distribution is defined implicitly by the deterministic mapping from this latent variable to the final sample.

2.2.5. Conditional Diffusion Models

So far, we have only considered sampling from a marginal distribution $q(x)$. But we also want to generate samples from conditional distributions $q(x | y)$, for example to generate an image x conditioned on a class label y , or an end effector trajectory conditioned on a task description.

Sohl-Dickstein et al. [10] introduced the idea of conditional generation with diffusion models by considering conditional distributions along the entire diffusion trajectory, i.e., over all $z^{(t)}$. This is a natural extension of the reverse process, where we now want to sample from $p_\theta(z^{(t-1)} | z^{(t)}, y)$ instead of $p_\theta(z^{(t-1)} | z^{(t)})$.

Practically speaking, we care about conditional generation because we want to generate samples with certain properties, not just any sample from the data distribution. If we can associate the properties of interest with every sample in the training data, and if we can feed a representation of these properties into a neural network, then the simplest idea for conditional generation is to feed these representations into the diffusion model at training time. Since the conditioning input contains information about the training sample, it should make the training objective easier and thus train the diffusion model to use the conditioning input effectively. In this case, the diffusion model can be formalized as computing $\mu_\theta(z^{(t)}, t, y)$ and $\sigma_\theta(z^{(t)}, t, y)$. We refer to this basic steering approach as *direct conditioning*. Direct conditioning can often be combined with other steering methods discussed below [20].

Inpainting

In some sampling applications, we want to generate samples where certain parts of the sample are already known. For example, in image generation, we may want to use an existing image and use a diffusion model to change a specific part of the image while keeping the rest of the image unchanged. Similarly, if we generate an end effector trajectory, we may want to fix the initial state to the current robot state or the final state to a desired goal position. In both examples, we want the generated sample to be consistent with the known parts. It would not be sufficient to simply generate a new sample and then overwrite the known parts, because the generated and known parts would likely be inconsistent. Thus, some form of conditioning is required where the diffusion model uses the known parts to generate the unknown parts. This is commonly referred to as *inpainting* in the image generation literature.

In Section 2.2.4 we discussed the sampling process for a diffusion model and saw that we compute the reverse process $p_\theta(z^{(t-1)} | z^{(t)})$ through predicting the clean data sample $\hat{x}_\theta(z^{(t)}, t)$ from the noisy sample $z^{(t)}$. To achieve inpainting, Sohl-Dickstein et al. [10] show that we can replace the predicted clean data sample $\hat{x}_\theta(z^{(t)}, t)$ with a modified version $\hat{x}_\theta(z^{(t)}, t) \odot m + x_{\text{known}} \odot (1 - m)$, where m is a binary mask that indicates which parts of the sample are known and which parts are unknown. This simple modification has been shown to work well in practice and to outperform non-diffusion-based inpainting methods for images [21].

We also want to emphasize that inpainting is a very ‘lightweight’ recipe for conditional generation with diffusion models, since it requires no architecture modifications, no additional training, and no separate models.

Classifier Guidance

Using Bayes’ rule, we can write the conditional reverse distribution as

$$p_\theta(z^{(t-1)} | z^{(t)}, y) \propto p_\theta(z^{(t-1)} | z^{(t)})p_\theta(y | z^{(t-1)}). \quad (2.54)$$

This means we can use the unconditional reverse process $p_\theta(z^{(t-1)} | z^{(t)})$ and reweight it with the conditional distribution. Sohl-Dickstein et al. [10] show that for a classifier $p_\theta(y | z^{(t-1)})$, the gradient of the log-likelihood $\nabla_{z^{(t-1)}} \log p_\theta(y | z^{(t-1)})$ can be used to reweight the unconditional reverse process. This is known as *classifier guidance* (CG) [20].

The proposed sampling procedure [10] works by modifying the means of the unconditional reverse process with the gradient of the log-likelihood of the classifier

$$\mu_\theta(z^{(t)}, t, y) = \mu_\theta(z^{(t)}, t) + \sigma_\theta(z^{(t)}, t)^2 I \nabla_{z^{(t)}} \log p_\theta(y | z^{(t)}), \quad (2.55)$$

where $\mu_\theta(z^{(t)}, t)$ and $\sigma_\theta(z^{(t)}, t)$ are the mean and standard deviation of the unconditional reverse process $p_\theta(z^{(t-1)} | z^{(t)})$. And $\mu_\theta(z^{(t)}, t, y)$ is the mean of the conditional reverse process $p_\theta(z^{(t-1)} | z^{(t)}, y)$. For the variance of the conditional reverse process, the variance of the unconditional reverse process is used [10].

Later work on text-to-image generation [20] found that the ‘strength’ of the steering from the classifier can be controlled by introducing a hyperparameter s that scales the classifier gradient

$$\mu_\theta(z^{(t)}, t, y) = \mu_\theta(z^{(t)}, t) + s \sigma_\theta(z^{(t)}, t)^2 I \nabla_{z^{(t)}} \log p_\theta(y | z^{(t)}). \quad (2.56)$$

Dhariwal et al. [20] also show empirically that this scaling factor introduces an explicit tradeoff between diversity and fidelity of the generated samples, where larger values of s lead to higher fidelity but lower diversity and vice versa. For later discussions we emphasize that the classifier in this setting is trained to predict the class label y from *noisy* samples $z^{(t)}$, not only from the clean data.

Classifier-Free Guidance

In contrast to direct conditioning, CG requires training a separate classifier to predict the conditioning input y from the noisy sample $z^{(t)}$. But CG allows us to tune the strength of the conditioning at inference time, which is not possible with direct conditioning.

To combine the advantages of a simple training procedure with the flexibility of CG, Ho et al. [22] propose a method called *classifier-free guidance* (CFG). The key idea is to train a single diffusion model to predict both conditional and unconditional samples and to use a weighted combination of the two predictions at inference time to achieve a similar effect as CG.

Specifically, Ho et al. [22] train a single diffusion model to generate $\mu_\theta(z^{(t)}, t, y)$ for conditional samples and $\mu_\theta(z^{(t)}, t)$ for unconditional samples. This is achieved by randomly dropping the conditioning input y during training. At inference time, the conditional and unconditional predictions are combined as

$$\mu_\theta = \mu_\theta(z^{(t)}, t) + s \left(\mu_\theta(z^{(t)}, t, y) - \mu_\theta(z^{(t)}, t) \right), \quad (2.57)$$

where s is a hyperparameter that controls the strength of the conditioning.

Ho et al. [22] show empirically that CFG achieves a similar effect as CG, where larger values of s lead to higher fidelity but lower diversity and vice versa. While CFG does not require training a separate classifier, it practically requires longer training times of the diffusion model. Additionally, CFG requires two forward passes of the diffusion model at test time.

2.2.6. Latent Diffusion Models

For data $x \in \mathbb{R}^d$, the diffusion model has to predict d values at every denoising step. For large values of d , this can become computationally expensive. For images, d can often be in the tens of thousands or even in the millions. We note that this is the same for all diffusion parameterizations we introduced earlier, since both the score and the noise have the same dimensionality as the clean data samples.

In Section 2.1.1 we discussed latent variables as a method to learn a compressed representation of our data. We can use this idea to train diffusion models on the latent representations, instead of the data itself [23]. This way, the number of values that the diffusion model learns to predict at each denoising step is reduced to the dimensionality of the latent representations. To generate new samples, we sample a latent representation $z \sim p_\theta$ from the diffusion model and use the decoder of the latent variables to generate a sample $x = f_{\text{dec}}(z)$. This strategy extends our overall sampling architecture from just a diffusion model to also learning an encoder and a decoder of the clean data.

Practically, there is a tradeoff between the reconstruction quality of the decoder and the quality of the samples generated by the diffusion model. Intuitively, the reconstruction objective of the decoder encourages the latent representation to contain as much information on the clean data as possible. On the other hand, the distribution of a high-information latent representation is more difficult for the diffusion model to learn. It can be shown empirically that this tradeoff exists and should be considered when developing a latent diffusion model [6].

2.3. Neuro-Symbolic Task and Motion Planning

2.3.1. Terminology

In this section we will introduce key terminology for the neuro-symbolic TAMP problem, following the definitions by Silver et al. [1]. We do not formalize these definitions here, because it is not necessary for our work. For formal definitions, we refer to Silver et al. [1].

We call the setting in which the robot operates the environment. The environment consists of different types of objects, where multiple objects of the same type can exist. Each object can be associated with a list of properties, specific to its type, such as size or position. We call the set of all objects with their current properties the continuous state of the environment. We define a set of logical predicates over those object types. Those predicates are boolean functions that can be evaluated on the objects in the environment. For example, a predicate could be `is_on_table(apple)`, and it would evaluate to true if the apple is on the table. Predicates can be defined over multiple objects, for example `is_on(red block, blue block)` would evaluate to true if the red block is on top of the blue block. The set of all true predicates at a given time is called the symbolic state of the environment. Symbolic actions can be used to transition between symbolic states. We assume knowledge of a transition function that defines all valid transitions between symbolic states. Practically, we are interested in achieving a certain symbolic goal state, starting from a start state. The symbolic planner is responsible for generating a sequence of symbolic actions that lead from the initial state to the goal state. We call the sequence of symbolic actions and states a plan. The motion planner is responsible for generating robot movements that fulfill each symbolic transition in the plan, leading from one symbolic state to the next.

3. Related Work

In this chapter, we discuss related work on representation learning, diffusion trajectory generation, diffusion guidance, and task and motion planning.

3.1. Representation Learning

Unified Latents [6] propose a method for learning latent representations that are regularized by a diffusion prior and reconstructed with a diffusion decoder. Heek et al. [6] use explicit loss weightings to control the amount of information contained in the latent representation, trading reconstruction quality against how difficult the latent distribution is to model with diffusion. They evaluate the approach on image and video generation. We extend the Unified Latents [6] approach to depth images and investigate how the information content of the latent representation affects downstream diffusion trajectory generation for motion planning.

3.2. Diffusion Trajectory Generation

Diffusion models were first introduced for trajectory planning by Janner et al. [24], who train an unconditional diffusion model from demonstration data and use it as a sampler of both states and actions. Besides the unconditional diffusion model, Janner et al. [24] train a reward model to predict the expected return of a trajectory at different diffusion noise levels. The reward model is used to guide the diffusion model at inference time, using classifier guidance. Additionally, Janner et al. [24] show that an unconditional diffusion model can be used to generate trajectories for different tasks, by using different task specific reward models for guidance.

To use diffusion models as policies trained with behavioral cloning, Chi et al. [25] propose generating shorter sequences of actions instead of full task horizon trajectories. Those short action sequences are commonly referred to as *action chunks*. The diffusion model uses direct conditioning on the current observation to generate the next action chunk. This way, they train task-specific policies from expert demonstrations. In contrast to Janner et al. [24], no guidance method is used, since the policy is single-task. Similar to our work, they use the diffusion model to generate actions in task space and show that their approach works significantly better with position control than with velocity control.

Ke et al. [3] train a multi-task diffusion policy and combine many different modalities to directly condition the diffusion model. Besides the current robot state, the diffusion model receives embeddings of RGB-D images and language instructions as inputs. The language instruction and scene images are embedded into the model by using powerful CLIP encoders [26]. They use no explicit guidance method, but instead feed all modalities into the diffusion model with direct conditioning. The diffusion model outputs a trajectory of end effector positions, orientations and gripper commands. In their ablation study, Ke et al. [3] show that using 3D information significantly improves the performance on certain tasks, such as inserting a light bulb into a socket, compared to using only 2D images.

The work by Fang et al. [5] is the most similar to our approach. Fang et al. [5] use a diffusion-based trajectory generator of latent point cloud representations and end effector positions. The diffusion model is steered with standard classifier guidance to perform different tasks and to fulfill constraints such as collision avoidance. Furthermore, Fang et al. [5] explicitly use their diffusion trajectory generator inside a TAMP system with a PDDL-Stream planner [27]. However, important differences to our approach remain. First, Fang et al. [5] use point clouds with a reconstruction objective, to learn a latent environment representation. In contrast, we learn a latent representation based on depth images and show that using only a reconstruction objective leads to reduced planning accuracy and task success. Second, the role of the symbolic information is different. Fang et al. [5] use standard classifier guidance [20] to steer the diffusion model between different behaviors, with classifiers trained to detect the corresponding symbolic action. In contrast, we investigate whether the desired next symbolic state, represented by predicates, can directly provide the guidance signal for a single multi-task diffusion model.

3.3. Diffusion Guidance

Bansal et al. [28] propose Universal Guidance, a method for steering diffusion models with guidance functions that were not trained on noisy diffusion inputs. In their forward guidance approach, the guidance function is evaluated on the predicted clean sample, and the gradient is propagated through the denoiser to the current noisy sample. This allows pre-trained classifiers and other models to be used for guidance without retraining them for the diffusion noise schedule. We build on this approach for predicate guidance, where the guidance function is a predicate classifier and the generated sample is a (partial) trajectory instead of an image. Compared to previous work [5, 29], this allows us to use classifiers that are not trained on noisy samples and that do not take the entire trajectory as inputs.

DynaGuide [29] uses a multi-task diffusion policy that is trained without direct task-conditioning. To perform a requested task, the diffusion policy is steered with a learned ‘outcome prediction model’. The outcome prediction model is a neural network that predicts a DINOv2 [30] embedding of the expected final state, given an action chunk from the diffusion policy and the current observation embedding. Importantly, the outcome prediction model predicts the final state embedding of the task, not the state after executing only one action chunk. The guidance signal is then based on the difference between the predicted final state embedding and the desired final state embedding. The desired final state embedding is obtained from the demonstration data itself. DynaGuide shows an effective way to steer an unconditional multi-task diffusion policy with classifier guidance. However, the guidance signal requires knowledge of the desired final state embedding, which limits generalization and the flexibility of a TAMP system. Additionally, requiring powerful pre-trained models such as DINOv2 increases the computational requirements.

3.4. Task and Motion Planning

Silver et al. [1] use bilevel planning with neuro-symbolic skills for long-horizon robotic tasks. Predicate classifiers map continuous object-centric states to symbolic states, while symbolic operators define preconditions and effects over these predicates. A symbolic planner generates sequences of symbolic states and skills, which are executed using neural subgoal samplers and policies in the continuous state space. This shows that predicates can connect symbolic planning with continuous robot behavior, supporting our use of



predicate classifiers as guidance from symbolic plans. In contrast to Silver et al. [1], where individual skills have separate policies, we use predicates to guide a single multi-task diffusion model during inference.

4. Methodology

This thesis focuses on the motion planning part of the TAMP problem. We do not implement the symbolic task planner itself. Instead, we assume that a symbolic planner generates a sequence of high-level actions and corresponding symbolic states. A symbolic state is described by a set of predicates that can be inferred from environment observations. For example, the symbolic action `close(microwave)` should lead to a symbolic state that includes the predicate `closed(microwave)`. We assume a partially observable environment where observations come from an observer camera.

The motion planner must generate continuous robot motions that realize the transitions between these symbolic states. Instead of using separate motion planning models for every symbolic action, we investigate a more general approach where a single model generates trajectories for different symbolic transitions. In our case, we use a diffusion model as a trajectory generator, which we discuss in detail in Section 4.2.

Since we use a single model to realize different symbolic actions, we need a test-time steering mechanism. As we outline in Section 4.3, we investigate steering from the symbolic action as well as from predicates directly.

To be able to steer the diffusion model from visual predicates, the diffusion model generates trajectories of both robot states and latent environment representations. As shown in Figure 4.1, our architecture consists of two stages. In Stage 1, we learn a latent representation of depth images from the observer camera. In Stage 2, we train the diffusion model with latent representations from the frozen encoder and demonstration data. The diffusion model is conditioned on the initial robot state and the initial environment observation, which is encoded into the latent representation.

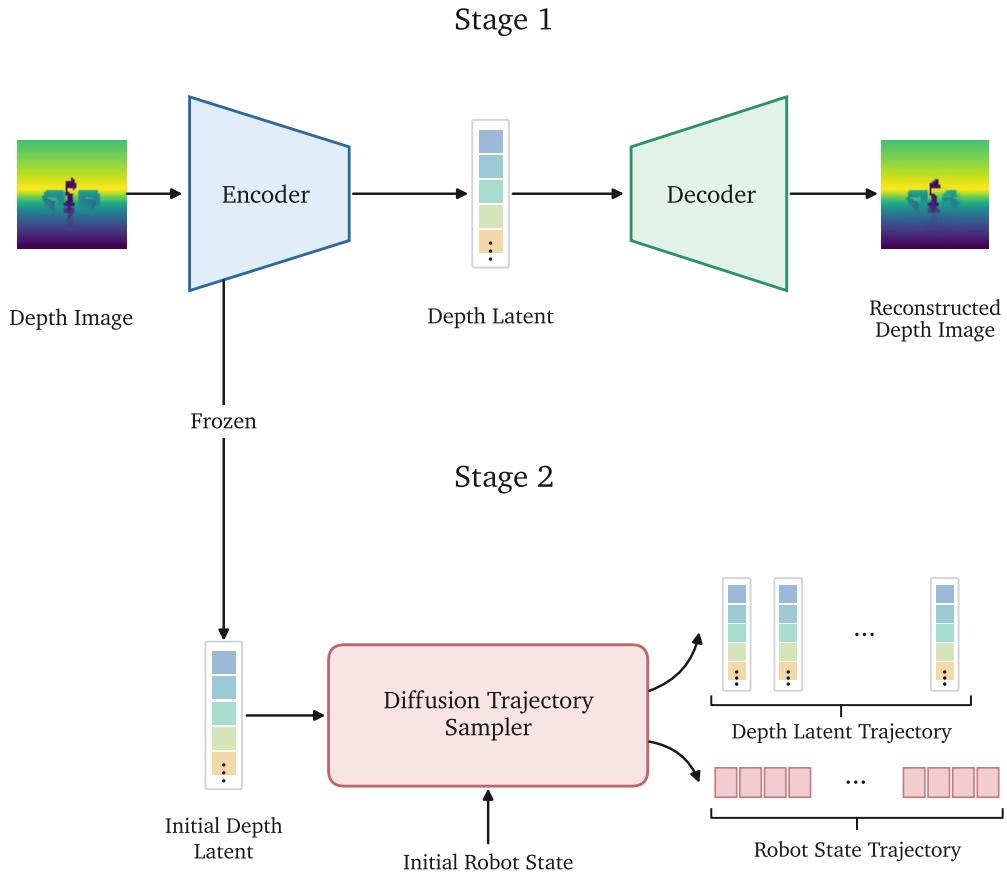


Figure 4.1.: Overview of our two-stage method. Stage 1 learns a depth latent with an encoder and a diffusion decoder. Stage 2 conditions on the initial latent and robot state to sample trajectories of robot states and depth latents.

4.1. Latent Environment Representations

Following the work by Fang et al. [5], we use a partially observable environment representation based on depth sensors and learn a latent representation for downstream use in our Stage 2 diffusion model. We use a latent representation instead of the raw depth data, because typical RGB-D cameras produce hundreds of thousands of depth measurements per frame. Thus, it is currently infeasible to use raw depth measurements as a state representation for the diffusion model, given the computational and real-time constraints in robot applications.

As discussed in Section 2.2.6, there is a trade-off between the amount of information in the latent representation and the learnability of the latent distribution for the Stage 2 diffusion model. We use a principled way to control this trade-off, by following the work by Heek et al. [6]. Therefore, as shown in Figure 4.2, our Stage 1 system consists of three models: a deterministic encoder, a diffusion prior, and a diffusion decoder.

The deterministic encoder $f_\theta : \mathbb{R}^{h_x \times w_x} \rightarrow \mathbb{R}^{h_z \times w_z \times c_z}$ maps depth images $x \in \mathbb{R}^{h_x \times w_x}$ to latent variables $z \in \mathbb{R}^{h_z \times w_z \times c_z}$. Here h_x, w_x are the height and width of the depth images, and h_z, w_z, c_z are the height, width and number of channels of the latent representation. In all our experiments, $h_x \times w_x = 200 \times 200$. The parameters of the encoder are denoted by θ . The diffusion decoder is a diffusion model $p_\psi(x | z)$ trained to generate the original depth image, conditioned on the latent representation. The parameters of the decoder are denoted by ψ . The diffusion prior $p_\phi(z)$ is an unconditional diffusion model trained jointly with the encoder and decoder to generate latent representations. The parameters of the prior are denoted by ϕ .

In contrast to Heek et al. [6], we use the x_0 parameterization for the diffusion decoder and prior, instead of the ϵ parameterization. We empirically found that the x_0 parameterization had superior performance on our datasets. Therefore, we adapt the loss functions by Heek et al. [6] to the x_0 diffusion parameterization. Therefore, we follow the work by Hooeboom et al. [31] and use the following loss function for the prior

$$\mathcal{L}_{\text{prior}}(\phi, \theta) = -\frac{1}{2} \lambda'_{\text{prior}}(t) e^{\lambda_{\text{prior}}(t)} \left\| z - \hat{z}(z^{(t)}, \phi) \right\|_2^2 + D_{KL}(q(z^{(1)} | x) || p(z^{(1)})), \quad (4.1)$$

and the decoder

$$\mathcal{L}_{\text{decoder}}(\psi, \theta) = -\frac{1}{2} \lambda'_{\text{decoder}}(t) e^b \text{sig}(\lambda_{\text{decoder}}(t) - b) \left\| x - \hat{x}(x^{(t)}, z^{(0)}, \psi) \right\|_2^2 \quad (4.2)$$

where sig is the sigmoid function and b is a hyperparameter.

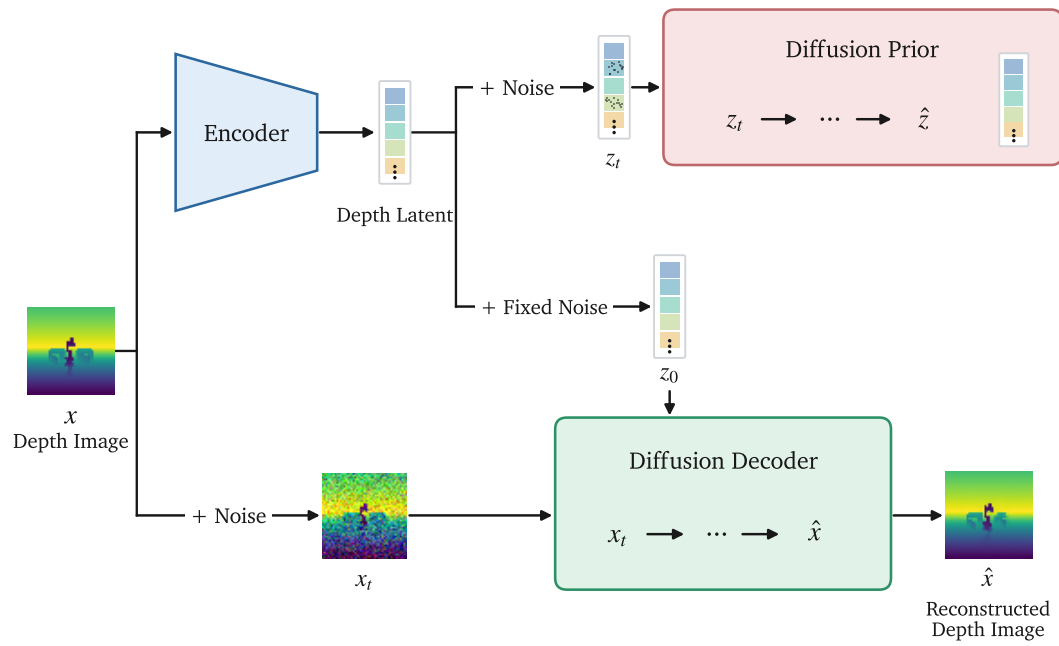


Figure 4.2.: Stage 1 training of the depth representation. The diffusion prior models latents perturbed at variable noise levels. The diffusion decoder reconstructs depth from a noisy depth image and a latent perturbed at a fixed noise level.

The complete loss function for training the Stage 1 system is given by

$$\mathcal{L}(\theta, \phi, \psi) = \mathcal{L}_{\text{prior}}(\phi, \theta) + c_{\text{lf}} \mathcal{L}_{\text{decoder}}(\psi, \theta) \quad (4.3)$$

The prior loss acts as a regularizer for the latent representation, while the decoder loss encourages the latent representation to contain sufficient information for reconstruction.

As in Unified Latents [6], we introduce a loss factor c_{lf} as an additional hyperparameter to control the amount of information in the latent representation and the latents given to the decoder during training are noised with the diffusion noise schedule of the prior at $t = 0$.

We use the same variance-preserving noise schedule for the prior and the decoder

$$\lambda_{\text{prior}}(t) = \lambda_{\text{decoder}}(t) = -20t + 10. \quad (4.4)$$

and we use deterministic DDIM [19] sampling with 32 steps. We chose the noise schedule and the number of sampling steps based on empirical performance on our datasets.

We experiment with different values of c_{lf} and b and estimate the information content of the latent representation by computing the ELBO of the prior. While c_{lf} scales the decoder loss relative to the prior loss, b allows us to distribute the decoder loss weight differently over the diffusion time steps. Figure 4.3 shows how the hyperparameters c_{lf} and b affect the decoder loss weight distribution. At higher noise levels, the decoder should rely more on the latent representation, since the noised input contains less information. Thus shifting more weight to higher noise levels should encourage more information in the latent representations.

We also experiment with a deterministic decoder, instead of a diffusion decoder. In this case, we use the mean squared error (MSE) as the loss function for the decoder together with the same prior loss function and decoder loss factor c_{lf} as above. Table A.5 specifies its architecture.

In this thesis, we train all encoders and decoders from scratch on the same dataset that is used to train the diffusion model. We use depth data from an RGB-D camera as input to the encoder. We don't use the RGB channels, because they are not necessary for the tasks we consider in this thesis. The network architectures for the encoder and decoder are based on the ResNet [32] architecture. See Appendix A for details.

Similar to Fang et al. [5], we also tried using point clouds as a data representation but found that using depth images greatly simplified our model architectures while reducing training times and increasing performance.

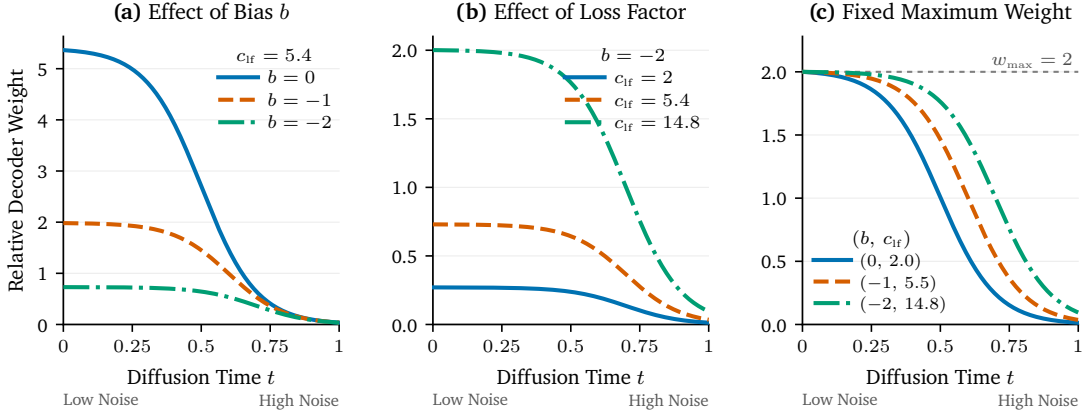


Figure 4.3.: Effect of the sigmoid bias b and decoder loss factor c_{lf} on the x_0 diffusion decoder weighting. Diffusion time increases from low-noise inputs at $t = 0$ to high-noise inputs at $t = 1$. In (c), the loss factor is adjusted such that all weighting functions have the same maximum.

4.2. Diffusion Trajectory Sampler

The trajectory sampler follows the design by Fang et al. [5]. We use a diffusion model that generates trajectories $\tau \in \mathbb{R}^{H \times (d_z + d_r)}$ of length H , where d_z is the dimensionality of the latent environment representation, and d_r is the dimensionality of the robot state. We also call H the horizon of the sampler. For the depth image latents, we use a three-dimensional latent representation, so that $d_z = h_z \cdot w_z \cdot c_z$.

The robot state is represented in task space with end effector position $p_i \in \mathbb{R}^3$, end effector orientation $R_i \in \mathbb{R}^{3 \times 3}$ and gripper state g_i , so that $d_r = 10$. The orientation is represented by the 6-dimensional vector $(R_i e_1, R_i e_2)^T$, where $e_1 = (1, 0, 0)^T$ and $e_2 = (0, 1, 0)^T$. The robot state and depth image at each time step are taken from the same frame of the demonstration.

We compare absolute and relative representations of the robot state. In the absolute representation, all positions and orientations are given in world coordinates. In the relative representation, the first robot state is absolute, and all subsequent positions and

orientations are relative to the end effector frame at the first time step:

$$p_i^{\text{rel}} = R_1^T(p_i - p_1), \quad R_i^{\text{rel}} = R_1^T R_i, \quad i = 2, \dots, H. \quad (4.5)$$

The gripper state remains absolute in both representations.

The network architecture of the Stage 2 diffusion model can be described as a two-stream residual network, where one stream processes the spatial latent representation and the other stream processes the robot state. When we use CFG, the diffusion model has an additional input for the task or target condition. See Appendix A for details on the architecture of the diffusion model.

In all our experiments, we use inpainting to condition the diffusion model on the initial environment observation and the initial robot state. Since we always use inpainting at test time, we include it already during training of the diffusion model. This means inpainting here is a form of direct conditioning while keeping the network architecture unchanged.

To train the diffusion model, we first encode the depth images of the demonstrations using the trained encoder from Stage 1.

We standardize the first and second moments of the latent variables and robot states before applying the diffusion process. Latent variables and robot states are standardized independently. For the latent variables, we standardize over channels. For the robot states, we standardize over each coordinate. In the absolute representation, we use the same values at every time step. In the relative representation, we compute separate values for the absolute initial states and all subsequent states. This also applies to the orientation representation and gripper state.

Let $\tau = ((z_1, r_1)^T, \dots, (z_H, r_H)^T)$ be a standardized trajectory sampled uniformly from the dataset. For each trajectory, we sample a diffusion time $t \sim \mathcal{U}(0, 1)$ and independent Gaussian noise $\epsilon \sim \mathcal{N}(0, I)$. We use the same variance-preserving noise schedule and linear log-SNR function as in Stage 1 to obtain the noisy trajectory $\tau^{(t)}$. The initial latent and robot state are inpainted with their clean values.

With x_0 parameterization, the diffusion model predicts the clean trajectory $\hat{\tau} = ((\hat{z}_1, \hat{r}_1)^T, \dots, (\hat{z}_H, \hat{r}_H)^T) = D_\theta(\tau^{(t)}, t)$. We minimize

$$\mathcal{L}(\theta) = \mathbb{E}_{\tau, t, \epsilon} \left[\frac{1}{H-1} \sum_{i=2}^H \left(\frac{w}{d_z} \|z_i - \hat{z}_i\|_2^2 + \frac{1-w}{d_r} \|r_i - \hat{r}_i\|_2^2 \right) \right]. \quad (4.6)$$

The factor w controls the relative importance of the mean squared errors of the latent variables and robot states. We exclude the first time step from the loss and use no additional weighting over diffusion time.

We use deterministic DDIM sampling with 32 steps to generate trajectories from the diffusion model.

For planning, we ideally want to be able to generate trajectories over the entire horizon of the task. However, different tasks have different horizons, and demonstrations usually have different lengths, even within the same task. At the same time, our diffusion model requires a fixed output size. And on top of that, we cannot practically use very large output sizes due to computational constraints. To clearly separate between the two horizons at play, we call the length of the demonstrations, which include all actions to complete a task, the *task horizon* and the length of the generated trajectories, the *sampling horizon*.

We investigate different approaches to deal with the mismatch between the task horizon and the sampling horizon. The first and well-established method is to use a receding window approach, where we generate multiple trajectories of the sampling horizon length and stitch them together to form a trajectory of the desired task horizon length. However, this approach can be prone to compounding errors and temporal inconsistencies between the generated trajectories. Another disadvantage of this approach is that it makes guidance from the symbolic planner more difficult. For example, if a predicate applies to the final state of a task, it cannot be directly used to guide the generation of the first trajectory chunks, because they don't contain the final state at the end of the task horizon. This is problematic for robotics tasks, where, for example, the approach to an object early in the trajectory can be crucial for the success of the task.

Therefore, we also experiment with a fixed task horizon. In this case, the sampling horizon can be set to the task horizon and cover the entire task. Since the demonstrations have different lengths, we need to enforce a fixed task horizon by standardizing the lengths of the dataset trajectories. Since our sampling horizon is shorter than any task horizon in our dataset, we only need to reduce the length of the demonstrations. We choose the simplest approach to this, which is to uniformly downsample the demonstrations to the desired task horizon length. Of course, this reduces the amount of information in the demonstrations and generated trajectories, which can lead to important parts of the movement being lost. However, we find this approach to be sufficient for our environments and tasks. Future work should investigate more sophisticated approaches, for example using hierarchical trajectory generation with different sparsity levels, or using more informed downsampling methods that preserve important parts of the trajectories (see, for example Chen et al. [33] and Ke et al. [3]).

4.3. Guidance from Symbolic Plans

We use diffusion guidance as the core mechanism to bridge the gap between symbolic task planning and motion generation. In this work, we use both CFG and classifier-based guidance. CFG is a very effective and simple way to steer the diffusion model. Classifier-based guidance is more flexible in this regard, since it allows us to use new guidance signals without retraining the diffusion model.

For classifier guidance, we have to decide what the classifiers should be trained on. A symbolic plan is a sequence of high-level actions and intermediate symbolic states. In principle, we can use the symbolic actions as well as the intermediate symbolic states to guide the diffusion model. We think that using the symbolic states is the more powerful approach. A symbolic state can be defined by a set of predicates that are true (or false) in that state. Predicates can be simple boolean classifiers, trained to predict the truth value of a predicate from an environment observation. The only training data they require is a set of labeled environment observations. Training such a classifier is a very well understood problem with existing datasets [34]. On the other hand, when using a classifier based on symbolic actions, the classifier has to predict whether a generated trajectory matches the symbolic action. This means the classifier needs to be trained on robot executed trajectories, which are more expensive to obtain than labeled environment observations and less generalizable than symbolic states.

One issue with steering the diffusion model from symbolic states is that we cannot reliably use standard classifier guidance. Let $p_\psi : \mathbb{R}^{d_\psi} \rightarrow [0, 1]$ be a predicate classifier that predicts the probability of a predicate being true. As discussed in Section 2.2.5, standard classifier guidance [20] uses the gradient $\nabla_{\tau^{(t)}} \log p_\psi(P\tau^{(t)})$ of the log-likelihood of the classifier, where $\tau^{(t)} \in \mathbb{R}^{H \times (d_z + d_r)}$ is the noisy trajectory at diffusion step t and P is a projection that extracts the relevant part of the trajectory used by the classifier. With standard ancestral sampling, the gradient is used to manipulate the mean of the Gaussian distribution from which the next noisy trajectory $\tau^{(t-1)}$ is sampled, with

$$\tau^{(t-1)} \sim \mathcal{N}(\mu_\theta(\tau^{(t)}, t) + s\sigma_\theta(t)^2 I \nabla_{\tau^{(t)}} \log p_\psi(P\tau^{(t)}), \sigma_\theta(t)^2 I). \quad (4.7)$$

As said above, we want to use general predicate classifiers trained on environment observations, not on entire trajectories. Such a classifier p_ψ will only use part of the trajectory as input, for example, the last latent variable of the trajectory, to determine the state of the environment at the end of the trajectory. As a result, the gradient $\nabla_{\tau^{(t)}} \log p_\psi(P\tau^{(t)})$ will be zero for all parts of the trajectory that are not used as inputs to the classifier. Looking

at Equation 4.7, we see that no information flows from the classifier to the parts of the trajectory that are not used as inputs to the classifier when the next noisy trajectory $\tau^{(t-1)}$ is sampled. The same is true for DDIM sampling. While the diffusion model itself could propagate information from the added gradient to other parts of the trajectory in the next denoising step, we find that this is not reliable in practice. A separate problem is that the classifier is trained on noisy data according to the diffusion noise schedule, which limits the use of pre-trained models and requires retraining models if the noise schedule is changed.

A solution to both problems is to use classifiers trained on clean data, and to propagate gradients through the denoiser $\nabla_{\tau^{(t)}} \log p_{\psi}(D_{\theta}(\tau^{(t)}, t))$ to the noisy trajectory $\tau^{(t)}$. Here D_{θ} is the diffusion model that predicts the clean trajectory from the noisy input. In this case, the gradients can be non-zero for all parts of the trajectory. This should be the case if the diffusion model has learned how earlier parts of the trajectory influence the later parts of the trajectory. We show that this approach works well in practice and that the gradients indeed propagate information to earlier parts of the trajectory, even if the classifier only uses the last latent variable of the trajectory as input. This approach is the same as in some training-free guidance approaches, such as forward guidance in Universal Guidance [28], which aim to solve the issue of training classifiers on noisy data.

Let the forward process be

$$\tau^{(t)} = \alpha(t)\tau + \sigma(t)\epsilon, \quad (4.8)$$

and let the denoiser directly predict the clean trajectory,

$$\hat{\tau} = D_{\theta}(\tau^{(t)}, t). \quad (4.9)$$

The guidance signal is obtained by evaluating the predicate classifier on the corresponding part of the predicted clean trajectory and differentiating its log-probability with respect to the current noisy sample:

$$g = \nabla_{\tau^{(t)}} \log p_{\psi}(P\hat{\tau}). \quad (4.10)$$

In the usual noise-prediction formulation, classifier guidance modifies the predicted noise as

$$\hat{\epsilon}_{\text{guided}} = \hat{\epsilon} - sg \quad (4.11)$$

with positive guidance scale s . Since noise and clean-sample predictions are related by

$$\hat{\epsilon} = \frac{\tau^{(t)} - \alpha(t)\hat{\tau}}{\sigma(t)}, \quad (4.12)$$

Algorithm 1: Predicate Guidance

Require: Unconditional denoiser D_θ , predicate classifier p_ψ

Require: Classifier input projection P , guidance scale s

Require: Initial noise $\eta \sim \mathcal{N}(0, I)$, noise schedule α and σ

Require: Diffusion coordinates $1 = t_0 > t_1 > \dots > t_n = 0$

```
1  $\tau^{(t_0)} \leftarrow \eta$ 
2 for  $i = 0, \dots, n - 1$  do
3    $\hat{\tau} \leftarrow D_\theta(\tau^{(t_i)}, t_i)$ 
4    $g \leftarrow \nabla_{\tau^{(t_i)}} \log p_\psi(P\hat{\tau})$ 
5    $\hat{\tau}_{guided} \leftarrow \hat{\tau} + s \frac{\sigma(t_i)}{\alpha(t_i)} g$ 
6    $\hat{\epsilon}_{guided} \leftarrow (\tau^{(t_i)} - \alpha(t_i) \hat{\tau}_{guided}) / \sigma(t_i)$ 
7    $\tau^{(t_{i+1})} \leftarrow \alpha(t_{i+1}) \hat{\tau}_{guided} + \sigma(t_{i+1}) \hat{\epsilon}_{guided}$ 
```

the equivalent update for an x_0 -parameterized denoiser follows from

$$\frac{\tau^{(t)} - \alpha(t) \hat{\tau}_{guided}}{\sigma(t)} = \frac{\tau^{(t)} - \alpha(t) \hat{\tau}}{\sigma(t)} - sg, \quad (4.13)$$

which gives

$$\hat{\tau}_{guided} = \hat{\tau} + s \frac{\sigma(t)}{\alpha(t)} g. \quad (4.14)$$

We then recover the corresponding guided noise prediction as

$$\hat{\epsilon}_{guided} = \frac{\tau^{(t)} - \alpha(t) \hat{\tau}_{guided}}{\sigma(t)}, \quad (4.15)$$

and use $\hat{\tau}_{guided}$ and $\hat{\epsilon}_{guided}$ in the deterministic DDIM update, as shown in Algorithm 1.

Note that, as stated above, the predicate classifier is trained on latent variables of environment observations, not on actual environment observations. To leverage pre-trained models for classification, one could, however, map the latent variables back to environment observations with the trained decoder. This works as long as the decoder is differentiable with respect to the latent variables.

5. Experiments

In this chapter we present our experiments and findings. We first present our environments and datasets in Section 5.1. Then, in Section 5.2 we define the evaluation metrics used in our experiments. Finally, we present our experiments and results. Our methodology naturally divides our analysis into three parts: first, we evaluate the reconstruction quality and generative performance of the Stage 1 system in Section 5.3. Second, we ablate the effect of the sampling horizon and robot state representation for the Stage 2 diffusion model in Section 5.4. Third, we evaluate predicate guidance in Section 5.5

5.1. Environments and Data

All our experiments use a 7-DoF Franka Emika Panda robot [35]. We use two different environments for our experiments.

To allow for principled and reproducible experiments that control for key data properties such as the number of tasks, the number of demonstrations per task, or the diversity of the demonstrations and the scene, we create our own environment. We use the Isaac Sim simulator [36] with Isaac Lab [37] and Isaac Lab-Arena [38] wrappers. Our custom Isaac Lab Environment consists of two tasks: closing a microwave and closing a stove.

To generate the demonstration data, we use a scripted policy. The scripted policy gives us control over the diversity and quality of the generated demonstrations while allowing us to generate an arbitrary number of demonstrations per task with little effort. However, scripted policies are only applicable to tasks that can be solved with a simple policy. This is why we restrict our experiments to tasks with rigid objects and simple manipulations.

We define two positions in which objects can be placed. We call these positions *Left Slot* and *Right Slot*. The objects present in the dataset are a stove and a microwave. Every demonstration successfully closes exactly one object. We call the object that is closed

Left Slot	Right Slot	Target Object
Microwave	-	Microwave
-	Microwave	Microwave
Stove	-	Stove
-	Stove	Stove
Microwave	Stove	Microwave
Microwave	Stove	Stove
Stove	Microwave	Stove
Stove	Microwave	Microwave

Table 5.1.: Scene configurations and target objects in the Isaac Lab Environment dataset. Each demonstration closes one target object.

the *target object*. The dataset contains both single-object scenes and scenes with one object of each type. Each object can be placed in either slot and be the target object. Figure 5.1 shows the scene configurations as RGB and depth observations. Table 5.1 shows the corresponding target objects.

The second environment we use is from the CALVIN benchmark [39]. CALVIN is a benchmark for long-horizon robot learning and provides a diverse set of human-generated demonstrations. We follow DynaGuide’s [29] preprocessing procedure to segment the continuous CALVIN-D play data into task-specific trajectories using simulator state information. Each segment is assigned a behavior label based on the corresponding object state changes. We randomly split the demonstrations into training, validation, and test sets. Figure 5.3 shows the manipulation tasks for which we have derived demonstration data.

We only use the depth data from the observer camera, not from the camera mounted on the robot. The observer camera is static in all demonstrations. Figure 5.2 shows RGB and depth observations from this camera. Both images show the same frame of a demonstration.

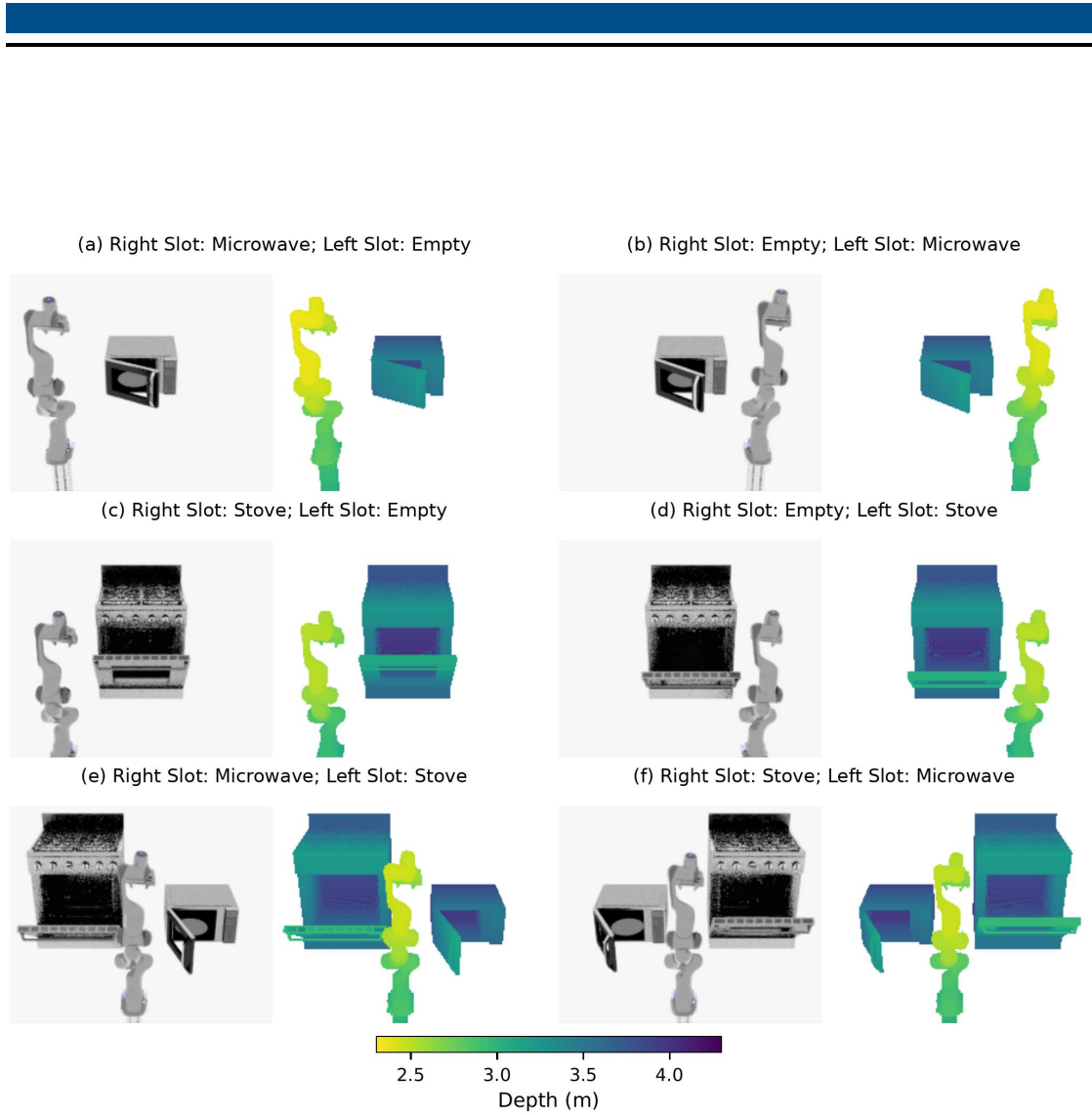


Figure 5.1.: Scene configurations in the Isaac Lab Environment. Each panel shows RGB observations on the left and depth observations on the right from the same frame. The depth scale is the same for all scene configurations. For scenes with both object types, either object can be the target. We removed floor and walls for visualization purposes.

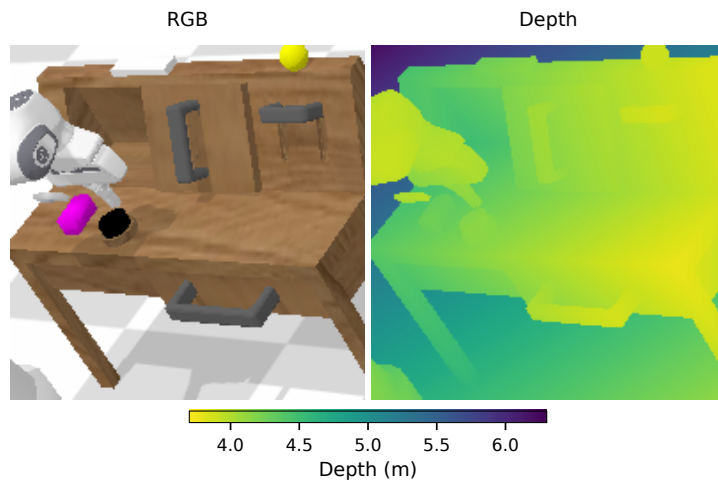


Figure 5.2.: RGB and depth observations from the static observer camera in the CALVIN environment. Both images show the same frame.

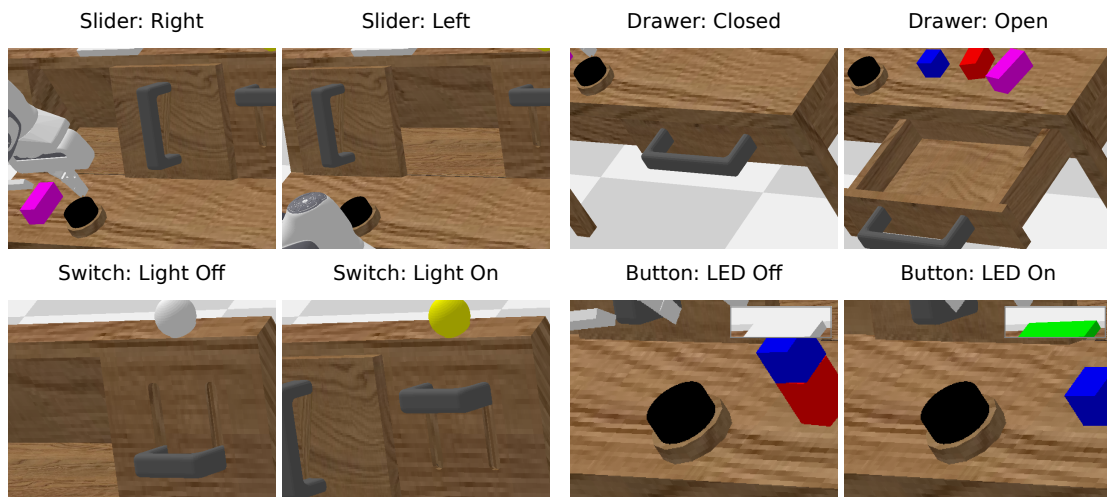


Figure 5.3.: Images of CALVIN manipulation tasks, generated in the simulator from demonstration states at higher image resolution. The slider and drawer are shown in different positions. The switch and button are shown with the corresponding light on or off.

5.2. Evaluation Metrics

We evaluate the reconstruction quality of the Stage 1 models using the root mean squared error (RMSE), mean absolute error (MAE), and Chamfer Distance (CD). To compute CD, we first convert the depth images to point clouds using the camera intrinsics. Since the observer camera in the CALVIN dataset is static, we use the same camera intrinsics for all images.

To compare distributions of depth images, we use the Fréchet Inception Distance (FID), introduced by Heusel et al. [40]. FID measures the distance between two Gaussian distributions fitted to feature embeddings of two sets of images. Let μ_r and Σ_r denote the mean and covariance of the reference features, and μ_g and Σ_g those of the evaluated features. The FID is then defined as

$$\text{FID} = \|\mu_r - \mu_g\|_2^2 + \text{Tr} \left(\Sigma_r + \Sigma_g - 2 \left(\Sigma_r^{1/2} \Sigma_g \Sigma_r^{1/2} \right)^{1/2} \right). \quad (5.1)$$

Lower values indicate that the two feature distributions are more similar.

Instead of using Inception features as in the original FID, we use features from the last hidden layer of a state estimation network trained on the CALVIN dataset. The network predicts the continuous states of the slider, switch, and drawer in the CALVIN environment. Table A.10 specifies the network architecture and the feature extraction used for FID. Since it is important for manipulation tasks that the latent representation contains sufficient information about those objects, a low FID on the state estimation features should indicate that the reconstructed or generated depth images capture the distribution of those object states well. This allows us to compare the reconstruction quality of the Stage 1 models and the generative performance of the Stage 2 model using the same metric.

5.3. Latent Representations

Here we discuss our results for learning latent representations from depth images. As discussed in Section 4.1, we want to control the amount of information contained in the latent representation. For accurate planning of manipulation tasks, the latent representation must contain sufficient information on the details of the scene, such as the shape and position of objects. However, more information in the latent representation can make diffusion sampling more difficult [6]. In this section we primarily investigate the approach by Heek et al. [6] to control the amount of information in the latent representation.

5.3.1. Latent Bitrates

As discussed in our methodology Section 4.1, we use Equations 4.1 and 4.2 to optimize the encoder and decoder. The Stage 1 and Stage 2 hyperparameters for both decoder types, together with the state-estimator settings for FID, are given in Appendix B.1.

Similar to Heek et al. [6], Table 5.2 shows that we can indeed control the latent bitrate by adjusting the loss factor c_{lf} and the sigmoid bias b . As expected, we see that increasing the loss weight of the decoder leads to higher latent bitrates and better reconstruction quality as measured by MAE and CD.

With $c_{lf} = 2.0$ and $b = 0$, the Stage 1 system suffers from mode collapse and has very low latent bitrate and poor reconstruction quality. When we increase the loss weight to $c_{lf} = 5.4$ and $b = -1.0$, we avoid mode collapse and obtain a latent bitrate of 61.64 bits per frame, with significantly improved reconstruction quality. Looking at Figure 4.3, we see that we primarily increase the loss weight of the decoder at higher noise levels, where the decoder should rely more on the latent variable, as the noisy input contains less information.

As discussed before, we are primarily interested in the effect of the latent bitrate on the Stage 2 diffusion model. Therefore, Table 5.2 also reports the FID of the Stage 1 encoder, by first encoding the depth image and then embedding the reconstructed depth image with the state estimation network. We call this the reconstruction FID (rFID), following the naming convention of Heek et al. [6]. The rFID can then be compared to the FID of the latent variables generated by the Stage 2 diffusion model, which we refer to as the generative FID (gFID). The state estimation network is the same for all models compared in Table 5.2. As expected, rFID improves with higher latent bitrates. On the other hand, gFID gets worse for higher latent bitrates, which is consistent with the findings by Heek et al. [6] for image and video generation.

c_{lf}	b	bits/frame	MAE (mm)	CD (mm)	rFID	gFID
2.0	0	0.26	146.21	50	0.1278	0.1171
5.4	-1.0	61.64	5.75	4.28	0.0007	0.0198
14.8	-2.0	81.54	4.79	4.64	0.0003	0.0861
40.0	-3.0	98.7	4.83	5.14	0.0004	0.2416

Table 5.2.: Latent bitrates, reconstruction quality, and FID metrics for Stage 1 models trained with different loss factors and sigmoid biases. We report the latent bitrate in bits per frame, the MAE, and CD between the reconstructed and ground truth depth images, as well as the rFID and gFID of the Stage 1 encoder and Stage 2 diffusion model, respectively. We compute these values over 1000 samples and 8 noise seeds per sample. The decoder uses the noised latent variable at $t = 0$ as input, as in training. This table is inspired by Table 2 in [6].

Deterministic Decoder

We also experiment with a deterministic decoder, instead of a diffusion decoder. This could be desirable for multiple reasons. First, it simplifies training and hyperparameter tuning. Second, it makes decoding latents faster. While the second point is not a concern for most of our experiments, a deterministic decoder could be used for guidance directly from depth images or by the task planner to sample generated trajectories in depth image space or any derived representation, such as state estimations.

For the deterministic decoder, we still want to control the amount of information in the latent representation. To see if the same approach can be used, we train models with different values of c_{lf} , which we apply, similar to the diffusion decoder, as a simple scaling factor for the decoder loss. As shown in Table 5.3, we again observe a clear correlation between the loss factor and the latent bitrate, as well as the same effects on reconstruction and FID metrics as seen in Table 5.2.

We find that the deterministic decoder is easier to tune and has better training efficiency under smaller compute budgets, as measured in FLOPs. Figure 5.4 compares the latent bitrate, CD, MAE, and RMSE of the deterministic decoder and the diffusion decoder as a function of Stage 1 training FLOPs. Decoder loss weights were selected based on the

c_{lf}	bits/frame	MAE (mm)	CD (mm)	rFID	gFID
1	68.77	6.24	7.21	0.0016	0.1464
2	86.77	5.83	6.9	0.0029	0.2065
3	100.71	5.47	6.58	0.0007	0.2626
4	112.48	5.43	6.56	0.0012	0.2849

Table 5.3.: Latent bitrates, reconstruction quality, and FID metrics for Stage 1 models trained with different loss factors and a deterministic decoder. We report the same metrics as in Table 5.2. The decoder uses the noised latent variable at $t = 0$ as input, as in training.

results of the latent bitrate estimation in Tables 5.2 and 5.3, to compare models under a similar latent information constraint. Specifically, the deterministic decoder uses $c_{lf} = 2.0$ and the diffusion decoder uses $c_{lf} = 5.4$ and $b = -1.0$. Panel (a) shows that the models converge to a similar latent bitrate early during training. While panel (d) shows that the deterministic decoder achieves a lower RMSE, panel (c) and (b) show that the diffusion decoder achieves equal or better CD and MAE. MAE and RMSE are computed in depth image space, where individual pixels are compared one to one. CD is computed in the *unordered* 3D point cloud space. CD does not penalize errors in the depth image, if the resulting 3D point cloud is still close to the ground truth. For example, if a single pixel at the edge of an object is decoded to be in the background rather than attached to the object, this pixel would be penalized by RMSE, and MAE, but not by CD. This suggests that the diffusion decoder is better at capturing the overall 3D structure of the scene, while the deterministic decoder is better at exactly reproducing depth images. But we cannot conclude whether this is due to the different objectives themselves or to the differences in the decoder architectures. Note that the diffusion decoder has to use a different network architecture than the deterministic decoder, because it has to take the noised sample and the diffusion time step as additional inputs to the latent variable. We believe that the advantage of a diffusion decoder could increase with more diverse datasets that have a wider range of objects and scenes.

Figure 5.5 compares original and reconstructed depth images for one *Drawer Open* demonstration from the held-out CALVIN test set. We use the deterministic encoder and decoder from the compute comparison in Figure 5.4. The reconstructed images preserve the shape

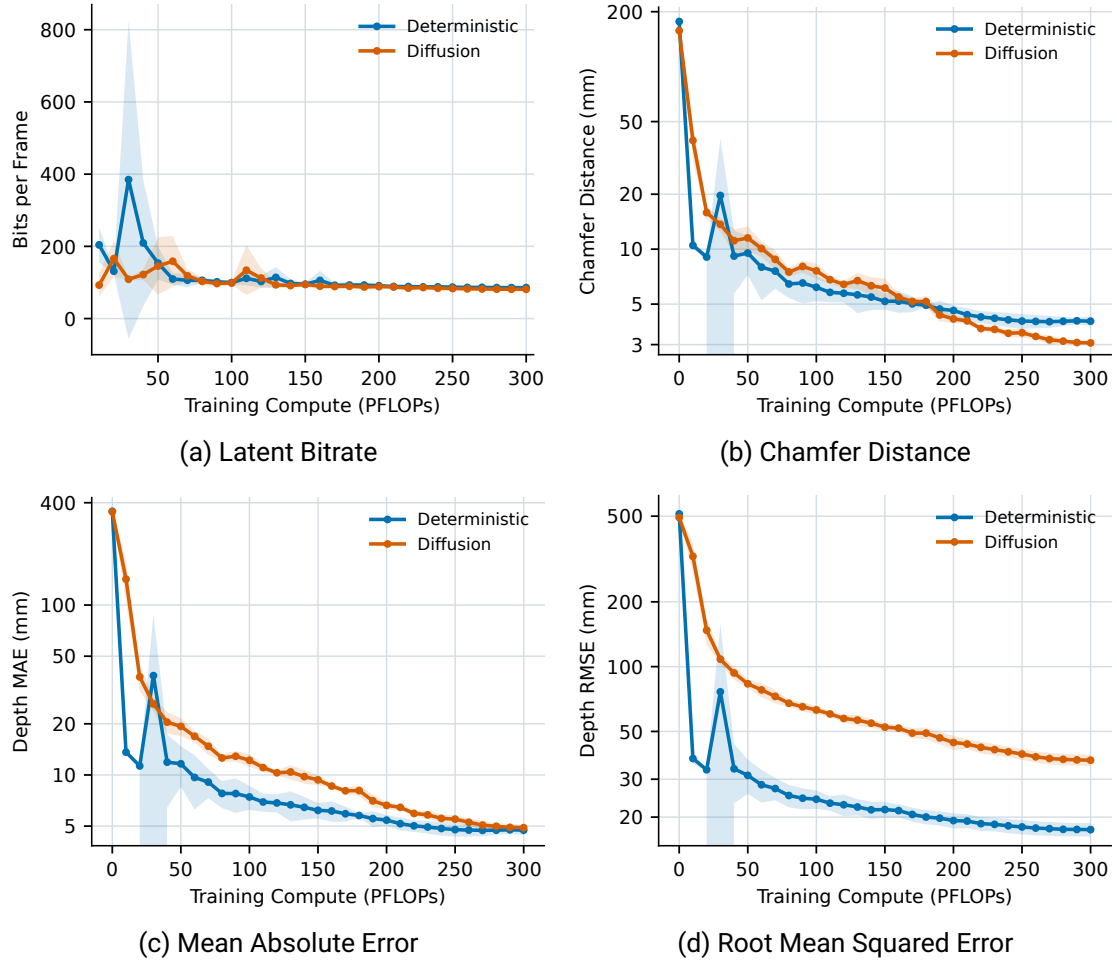


Figure 5.4.: Validation metrics and latent bitrate over Stage 1 training FLOPs for the deterministic decoder and the diffusion decoder. We compare three test seeds per model type and report the mean and standard deviation. We report the latent bitrate, CD, MAE, and RMSE for the reconstructed depth images. To compute CD, we convert the depth images to 3D point clouds and compute the Chamfer Distance between the reconstructed and ground truth point clouds. All metrics beside the latent bitrate are computed in millimeters.

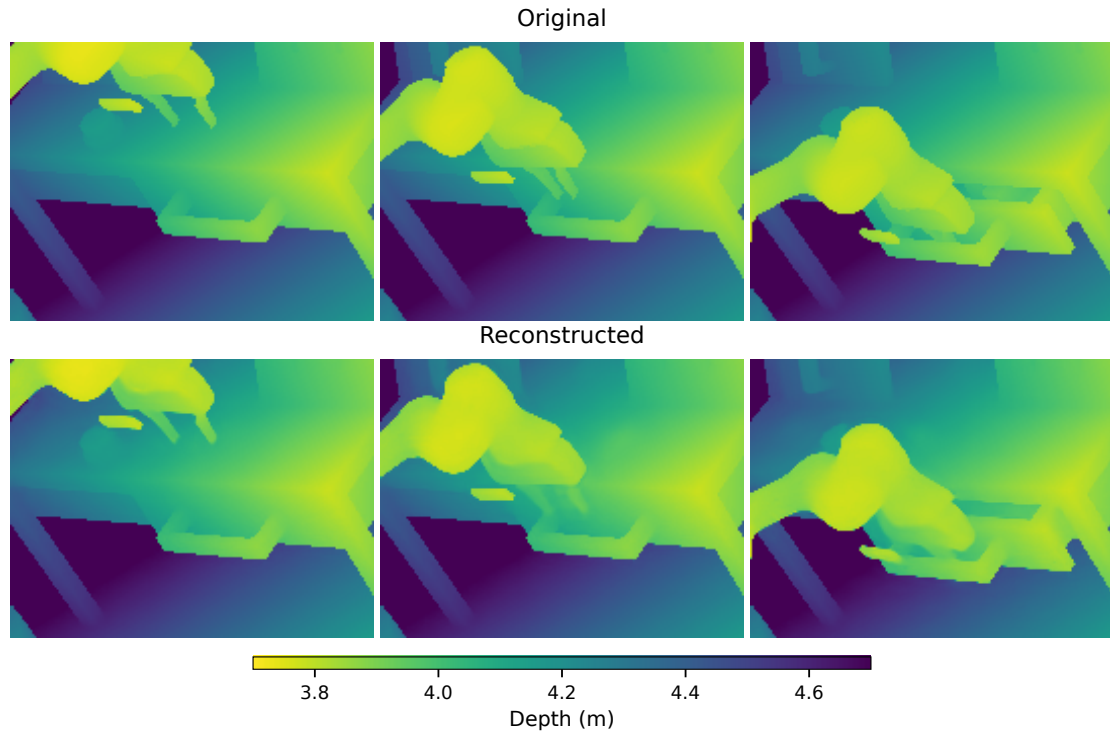


Figure 5.5.: Original depth images (first row) and reconstructed depth images (second row) for one *Drawer Open* demonstration from the held-out test set, using the deterministic decoder.

and position of the robot and drawer, with differences at the edge of objects.

Downstream Effects of Latent Bitrates

Here we evaluate the effect of the latent bitrate on the Stage 2 diffusion model beyond the FID metrics in Tables 5.2 and 5.3.

As discussed in Section 4.2, we want the diffusion model to be capable of generating trajectories over the entire task horizons with a receding horizon approach. Therefore, it is important that sampled latent representations are consistent across time. Manual inspection of the reconstructed depth images, as shown in Figure 5.6, suggests that the

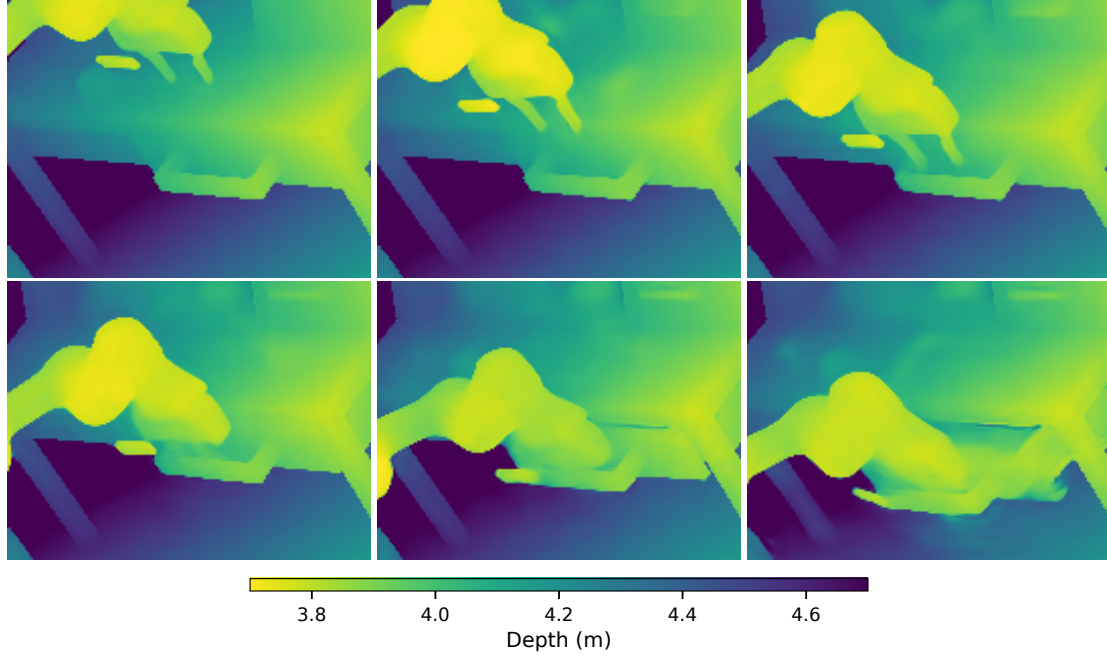


Figure 5.6.: Decoded depth images from a Stage 2 trajectory for *Drawer Open*, generated with CFG scale $s = 2$ and a sampling horizon of $H = 10$. Frames 0, 9, 18, 27, 37, and 46 are shown from left to right and then from the first row to the second row. The first image is decoded from the inpainted initial latent.

Stage 2 diffusion model is able to generate temporally consistent depth images. But it is not clear how the latent bitrate of the Stage 1 system relates to temporal consistency.

To evaluate the impact of the latent bitrate on temporal consistency and receding horizon planning, we sample trajectories from the Stage 2 model and execute them in the CALVIN simulator. Then we compare the generated depth images with the ground truth depth images from the simulator.

Specifically, for each Stage 1 system in Table 5.3, we train Stage 2 models with the same network dimensions and optimization settings as for gFID evaluation, but add task conditioning with CFG. We use guidance scale $s = 2$ and a dropout probability of $p = 0.1$ for the task conditioning. We use the initial states from our held-out test set and condition the Stage 2 diffusion model on the respective depth image and robot state from the simulator. Then we generate task horizon trajectories open loop with a receding

window approach and execute the actions in the simulator. We then report the success rate of the generated trajectories in Table 5.4 and compute the differences between the ground truth depth images from the simulator and the decoded depth images from the generated latent representations. Since differences in ground truth depth images and decoded depth images can be due to planning errors as well as decoding errors, we also report the decoding error at the evaluated time steps. Figure 5.7 shows both the depth image MAE between planned and ground truth states as well as the decoding error for the *Switch On*, *Drawer Open*, and *Slider Left* tasks. We compare Stage 2 models trained on latent representations with different bitrates, as indicated by the loss factor values. The Stage 1 systems are the same as in Table 5.3. Per Stage 1 system, we train 3 Stage 2 models with different test seeds and evaluate them over 8 noise seeds per initial state. All Stage 2 models have a sampling horizon of $H = 10$; thus, open loop generation requires 6 receding horizon steps for the evaluated tasks. Each sample from the diffusion model is conditioned on the last state of the previously generated trajectory segment. Figure 5.7 shows that for all latent bitrates, the prediction error increases over time. Further we can see that the differences between latent bitrates are small.

Since generated depth errors could be explained by a trajectory sampler that generates latent trajectories that are actually valid for the sampled robot states, but the sampled robot states are not executed correctly by the controller in the simulator, we also measure position and orientation errors of the end effector between generated and executed trajectories. Figure 5.8 shows the tracking error in the end effector position and orientation for the same rollouts. Here we can see that the tracking errors do indeed increase over time, but differences between latent bitrates remain small as well.

In Table 5.4 we summarize the generated depth MAE and end effector tracking error for the three evaluated tasks, averaged over all time steps and all seeds. Additionally, we report the success rate of the generated trajectories. Based on this we conclude that the latent bitrates are a poor predictor of planning accuracy and task success. Due to the generally low bitrates and FID scores for our dataset and model families, we believe that larger differences in planning accuracy and task success could be observed for more diverse and challenging datasets.

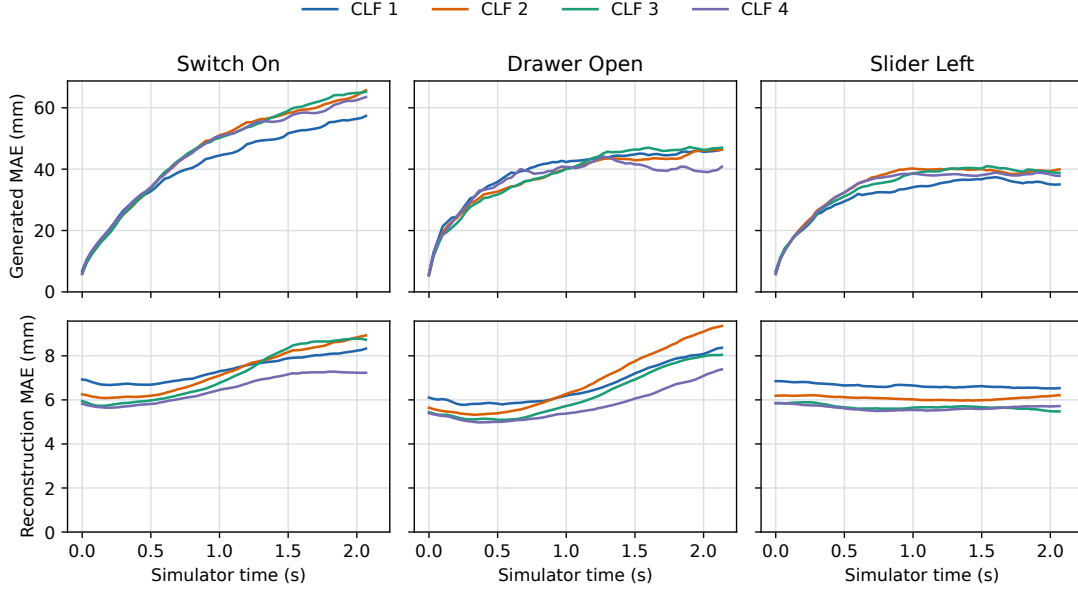


Figure 5.7.: Latent trajectory prediction results for three tasks using open loop receding horizon planning in the CALVIN simulator. We compare three Stage 2 models per Stage 1 system, trained on held-out test seeds and evaluated over 8 noise seeds per task and initial state. The Stage 1 systems are the same as in Table 5.3. CLF values indicate the loss factor c_{lf} used to train the Stage 1 system. All Stage 2 models have a sampling horizon of $H = 10$ and have an overlap between receding horizon segments of one frame. The last frame of each receding horizon segment is used to condition the next segment.

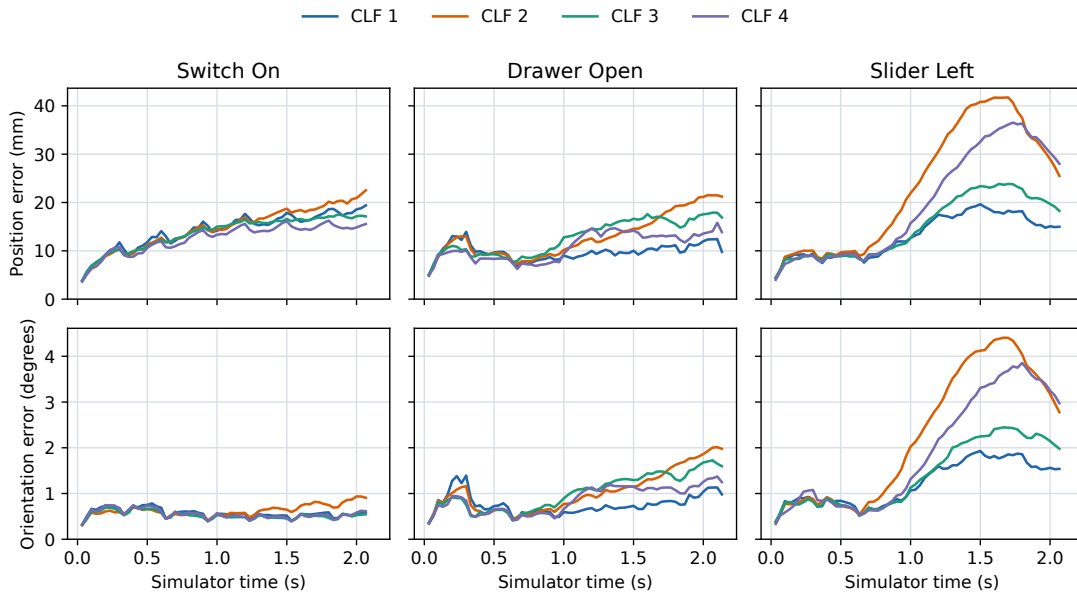


Figure 5.8.: Tracking errors for the CALVIN Stage 2 benchmark. The first row shows the end effector position error in mm, the second row shows the end effector orientation error in degrees. Errors are computed between the generated and executed trajectories. The evaluated models and rollouts are the same as in Figure 5.7.

c_{lf}	Latent Bitrate	Generated Depth MAE (mm)	Position Error (mm)	Success (%)
1	68.77	37.79 $\pm$ 1.28	12.38 $\pm$ 1.07	64.17 $\pm$ 12.64
2	86.77	39.85 $\pm$ 6.59	16.96 $\pm$ 6.2	70.69 $\pm$ 5.21
3	100.71	39.95 $\pm$ 7.25	13.87 $\pm$ 1.41	73.75 $\pm$ 9.75
4	112.48	39.14 $\pm$ 3.27	14.43 $\pm$ 5.21	63.06 $\pm$ 3.78

Table 5.4.: Latent bitrates, average generated depth MAE, end effector position error, and success rate, over *Switch On*, *Drawer Open* and *Slider Left* tasks. Each task-level result is averaged over all time steps and all seeds from Figures 5.7 and 5.8. Errors and success rates are reported as mean $\pm$ standard deviation over model seeds.

Effect of the Diffusion Prior

In this section we investigate the effect of the diffusion prior on the Stage 1 latent representation on the Stage 2 diffusion model.

We train the same Stage 1 system as in Table 5.3 with $c_{lf} = 1$, but use a stop gradient on the prior loss for the optimization of the encoder and decoder. This way, the encoder and decoder are optimized only through the reconstruction objective, while the diffusion prior is trained separately to estimate the latent bitrate.

Table 5.5 shows that without the prior loss contribution, the latent bitrate increases significantly. As expected, the reconstruction quality improves over all previous models shown in Table 5.3. Interestingly, the gFID of the Stage 2 diffusion model is better than for the models trained with $c_{lf} = 3$ and $c_{lf} = 4$, but worse than for the model trained with $c_{lf} = 1$ and $c_{lf} = 2$.

setting	bits/frame	MAE (mm)	CD (mm)	rFID	gFID
with prior	68.77	6.24	7.21	0.0016	0.1464
without prior	693.17	4.35	5.69	0.0002	0.2438

Table 5.5.: Latent bitrates, reconstruction quality, and FID metrics for Stage 1 models trained with and without the prior loss. For the model trained with the prior loss, we use $c_{lf} = 1$. We report the same metrics as in Table 5.2.

Table 5.6 shows the planning accuracy and task success of the Stage 2 diffusion model trained on the latent representation without prior loss, compared to the models trained with prior loss and different values of c_{lf} . From this we can see that the generated depth MAE is significantly worse than previous models, even though the tracking error is the lowest. The average task success is also the lowest for the model trained without the prior loss. This suggests Stage 2 diffusion does indeed suffer from the increased latent bitrate in terms of planning accuracy. Additionally, the results suggest that the gFID of the Stage 2 diffusion model is not a good predictor of planning accuracy and task success.

Setting	Latent Bitrate	Generated Depth MAE (mm)	Position Error (mm)	Success (%)
$c_{lf} = 1$	68.77	37.79 $\pm$ 1.28	12.38 $\pm$ 1.07	64.17 $\pm$ 12.64
$c_{lf} = 2$	86.77	39.85 $\pm$ 6.59	16.96 $\pm$ 6.2	70.69 $\pm$ 5.21
$c_{lf} = 3$	100.71	39.95 $\pm$ 7.25	13.87 $\pm$ 1.41	73.75 $\pm$ 9.75
$c_{lf} = 4$	112.48	39.14 $\pm$ 3.27	14.43 $\pm$ 5.21	63.06 $\pm$ 3.78
without prior	693.17	46.11 $\pm$ 4.03	12.24 $\pm$ 3.75	60.83 $\pm$ 9.20

Table 5.6.: Average generated depth MAE, end effector position error, and success rate, over *Switch On*, *Drawer Open* and *Slider Left* tasks. Results are averaged over all time steps and all model seeds. We compare the Stage 2 diffusion model trained on the latent representation with and without the prior loss. Errors and success rates are reported as mean $\pm$ standard deviation over model seeds.

5.4. Trajectory Sampling

In this section we analyze the impact of different design choices for the Stage 2 diffusion model, regarding the robot state representation and the sampling horizon.

We compare two different representations of the robot state for the Stage 2 diffusion model. Since the robot states are not used during Stage 1 training, we can compare Stage 2 models trained on the same Stage 1 latent representation. We use the same network dimensions and optimization settings as in the preceding CALVIN simulation experiments, with task conditioning through CFG. Here we compare two action representations for the diffusion model. The first representation uses absolute task space positions, orientations, and gripper commands. The second is a relative representation, where the robot states after the first inpainted frame are relative to that first absolute frame.

Table 5.7 shows the success rates for both representations, on the eight evaluated tasks. Task success is evaluated in the CALVIN simulator, using initial states from the held-out test set. We train three Stage 2 models per representation on held-out test seeds with CFG. We then evaluate each model in the simulator with eight different noise seeds per task and initial state. We find that averaged success rates are similar for both representations, with $93.91 \pm 3.03\%$ for the absolute representation and $92.34 \pm 18.29\%$ for the relative

Task	Absolute (%)	Relative (%)
Button On	88.75	91.25
Button Off	93.33	72.08
Switch On	98.33	95.83
Switch Off	99.17	93.75
Drawer Open	95.00	97.50
Drawer Close	93.75	98.75
Slider Left	83.33	92.08
Slider Right	99.58	97.50
Aggregated	93.91 $\pm$ 3.03	92.34 $\pm$ 18.29

Table 5.7.: Success rates for absolute and relative action representations. Task rows show the mean success rate over the three model seeds and eight noise seeds per initial state. The aggregated row reports the mean over tasks and seeds with a 95% Student- t confidence interval computed across the seed-level task averages. All evaluated models have a sampling horizon of $H = 10$ and are trained on the same Stage 1 latent representation.

representation. High variance in the relative representation is due to the *Button Off* task, which has a low success rate of 72.08%, which is dominated by one of the three training seeds, which achieved only 40.2% success on this task.

Next, we evaluate the effect of the sampling horizon on the Stage 2 diffusion model. A longer sampling horizon allows the model to plan over longer time horizons within a single diffusion rollout. While this can improve the inference speed of the model, increasing the sampling horizon linearly increases the number of values the model must predict, which can make the task more difficult. Since the Stage 2 model also acts as a planner, we not only evaluate task success but also the accuracy of the generated latent trajectories. We measure MAE between the generated depth images and the ground truth depth images from the simulator and the position tracking error between the generated and executed end effector positions. For the planning metrics, we use the same receding horizon

Horizon	Success (%)	Generated Depth MAE (mm)	Position Tracking Error (mm)
10	92.3 ± 18.3	42.58 ± 14.82	18.71 ± 14.21
16	99.0 ± 1.2	35.14 ± 10.89	14.63 ± 12.70
24	96.8 ± 2.0	39.21 ± 7.79	15.25 ± 12.81
32	87.7 ± 21.5	44.61 ± 13.47	16.10 ± 4.46

Table 5.8.: Mean success rate, generated depth MAE, and position tracking error over the eight evaluated tasks for each horizon. We evaluate the same tasks as for Table 5.7. Each task-level value aggregates three training seeds and eight noise seeds per initial state. The reported intervals are 95% Student- t intervals.

approach as in the previous section, where we use the last frame of each receding horizon segment to condition the next segment. For the task success evaluation, we use the same approach but with feedback from the environment. This means every sample from the diffusion model is conditioned on the observation from the environment and robot state after the previous action chunk has been executed. We compare the performance of Stage 2 models trained with sampling horizons of $H = 10, 16, 24, 32$ frames. The models are trained on the same Stage 1 latent representation and with the same training settings and CFG as in the robot-representation comparison in Table 5.7. Table 5.8 shows the success rates, generated depth MAE, and position tracking error for each sampling horizon.

5.5. Predicate Guidance

As described in Section 4.3, we are especially interested in evaluating diffusion guidance with predicates that only use scene observations or single depth latents as inputs, instead of entire trajectories. For this experiment, we use our custom Isaac Lab Environment and demonstration dataset, with the scene configurations described in Section 5.1 and Table 5.1.

Let $\hat{\tau} = ((z_1, r_1)^T, \dots, (z_H, r_H)^T)$ denote the clean data prediction of the diffusion model D_θ , such that $\hat{\tau} = D_\theta(\tau^{(t)}, t)$ for a noisy trajectory $\tau^{(t)}$ at diffusion time t . We train a predicate classifier closed_ψ that receives (parts) of clean data trajectories and a query

vector as inputs. For simplicity, we omit the parameters ψ of the predicate classifier in the following. The classifier is trained on the same dataset as the Stage 2 diffusion model and thus uses the same latent space as inputs. The query vector is 4-dimensional and concatenates one-hot encodings of the target object type and slot.

As motivated in Section 4.3, we are interested in steering the diffusion trajectory generator from predicates that are evaluated only on parts of the trajectory. This is because such classifiers are generalizable beyond the collected robot demonstrations, and the samples for training them are easier to obtain.

We train four variants of a closed classifier that differ in what part of the trajectory τ they get as inputs. The first variant takes the entire trajectory $\hat{\tau}$ as an input. The second variant only takes the latent variables of the trajectory $(z_1, \dots, z_H)$ as inputs. The third variant takes the terminal latent variable and robot state (z_H, r_H) as inputs. The fourth variant only takes the terminal latent (z_H) as inputs. Table A.7 in Appendix A specifies their shared network architecture, and Table A.8 summarizes the four input variants.

We train unconditional diffusion models on the demonstration data. To guide the unconditional diffusion model from the closed predicate with only the terminal latent variable, we standardize the length of the demonstrations and set the sampling horizon to the same length, as described in Section 4.2. At test time, we use deterministic DDIM sampling, as shown in Algorithm 1.

As described in Section 4.3, we compute gradients through the classifier and the diffusion model, because with standard classifier guidance, we had no direct flow of information from the classifier to the entire trajectory if the classifier only takes part of the trajectory as inputs. While our approach allows for gradients to flow from the classifier to the entire trajectory, we want to evaluate if this actually works out in practice. For the classifier that takes only the terminal latent variable of the trajectory as inputs, Figure 5.9 shows the gradient magnitudes with respect to the entire trajectory over diffusion times. We compare the gradients $\nabla_{\hat{\tau}} \text{closed}(P\hat{\tau}, q)$ against the gradients $\nabla_{\tau^{(t)}} \text{closed}(PD_{\theta}(\tau^{(t)}, t), q)$, which are the gradients taken through the denoiser. While the gradients $\nabla_{\hat{\tau}} \text{closed}$ are zero everywhere besides the terminal latent variable, gradient magnitudes for the gradients taken through the denoiser are non-zero.

While this shows that information from the classifier can affect the entire trajectory, we also want to evaluate if this actually improves task success. Therefore, we compare the task success of the four classifier input variants against an unguided baseline. We use three development seeds for the predicate classifiers and Stage 2 diffusion models to optimize model guidance hyperparameters. We select the best hyperparameters based

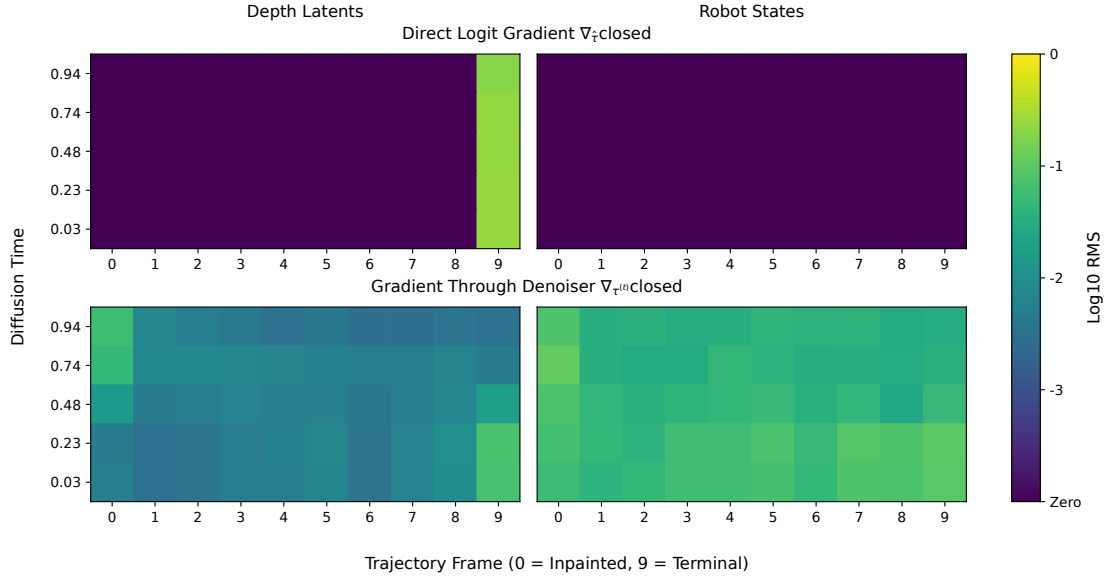


Figure 5.9.: Gradient magnitudes for the closed predicate classifier that only uses the terminal latent variable as input. The first row shows the gradient magnitudes of the classifier with respect to the prediction from the denoiser, $\nabla_{\hat{\tau}} \text{closed}(P\hat{\tau}, q)$. The second row shows the gradient magnitudes of the classifier with respect to the noisy trajectory $\nabla_{\tau^{(t)}} \text{closed}(PD_{\theta}(\tau^{(t)}, t), q)$. The first column shows the gradient magnitudes for the latent variables, the second column shows the gradient magnitudes for the robot states. Magnitudes are computed as the RMS of the gradients over the feature dimensions in log space. We report the median magnitudes over 50 rollouts and different diffusion times.

on task success in the simulator. The selected classifier and guidance settings are given in Appendix B.2. Then we train five Stage 2 models and three classifiers per input type on held-out test seeds and evaluate every Stage 2 model with every input type in the simulator over 8 noise seeds per scene configuration. For every Stage 2 model, we also evaluate the unguided task success as a baseline. We evaluate the models on mixed-object scene configurations with one microwave and one stove at either slot and every slot-object combination as the target.

Table 5.9 shows the aggregated success of each classifier input type against the unguided baseline. A rollout is counted as successful if the robot closes the selected object at the selected slot without interacting with the distractor. This shows that all classifier input types significantly improve success over the unguided baseline. On average, the best performing classifier input type is the terminal latent variable, which achieves 96.0% success.

During development, we also experimented with Universal Guidance [28] style recurrence and backward steps but did not find settings in which it improved success metrics.

Method	Success (%)	Model-Seed Range (%)
Unguided	48.4	46.9–51.6
Terminal Latent	96.0	87.5–100.0
Terminal Latent + Terminal Robot	95.5	90.6–100.0
Latent Trajectory	94.1	79.7–100.0
Latent + Robot Trajectory	91.7	76.6–100.0

Table 5.9.: Comparison of training-free classifier guidance inputs for the closed predicate. Terminal latent guidance uses only the last latent variable of the sampled trajectory as input to the classifier; latent trajectory guidance uses the entire latent trajectory; latent + Robot trajectory guidance uses the entire model output of latent and robot states. Success is averaged over five Stage 2 model seeds and three classifier seeds per guided method. Ranges give the minimum and maximum success rates over model-classifier seed pairs for guided methods and over model seeds for the unguided baseline.

As hinted at in Section 5.3, using the deterministic decoder allows us to use predicate

guidance in depth-image space, instead of latent space. To do this, we only need to change line four in Algorithm 1 to compute the gradient as

$$g = \nabla_{\tau(t_i)} \log p_{\psi}(f_{\text{dec}}(P\hat{\tau}), q), \quad (5.2)$$

where $\hat{\tau}$ is the prediction of the Stage 2 diffusion model, f_{dec} is the Stage 1 decoder, and P is a projection that selects the terminal latent from the generated trajectory. In this case, the predicate classifier p_{ψ} is trained in depth image space. Table A.9 specifies the depth-image classifier architecture.

To evaluate how guidance through the decoded depth images performs, we use the same evaluation protocol as before, with the depth-image classifier settings given in Appendix B.2.4. We use the closed predicate classifier with the same query vector q , but train it on depth images directly with the same underlying dataset. Then we evaluate the same unconditional Stage 2 diffusion models as in Table 5.9 but use predicate guidance with gradients as in Equation 5.2. Table 5.10 shows that using the decoded depth images for guidance significantly reduces success to 70.3%, compared to 96.0% when using the terminal latent variable for guidance. Additionally, we find that there is substantial variance in success between different model-classifier seed pairs.

Method	Success (%)	Model-Seed Range (%)
Unguided	48.4	46.9–51.6
Terminal Latent	96.0	87.5–100.0
Decoded Depth Image	70.3	54.7–84.4

Table 5.10.: Success for guidance with the closed predicate evaluated on the terminal latent or its decoded depth image, compared with the unguided baseline. Decoded depth image guidance propagates classifier gradients through the Stage 1 decoder and the Stage 2 denoiser. Results are averaged over five Stage 2 model seeds and three classifier seeds per guided method. Ranges give the minimum and maximum success rates over model-classifier seed pairs for guided methods and over model seeds for the unguided baseline.

6. Discussion

Our experiments on latent representations show that we can control the information content of the latent representation through the loss factor and decoder weighting, similar to Heek et al. [6]. Increasing the bitrate improves reconstruction while making the latent distribution more difficult for the Stage 2 diffusion model to generate, as measured by the FID metrics. The same effect is particularly clear with the deterministic decoder in Table 5.3, where generative FID consistently becomes worse as the bitrate increases. This suggests that the bitrate provides a useful way of controlling the tradeoff between reconstruction and diffusion sampling also for depth representations. Our ablation on the diffusion prior in Section 5.3.1 further shows that the diffusion prior has an effect on the planning accuracy and task success of the Stage 2 model.

However, we do not observe the same clear relation we see between the latent bitrate and gFID in downstream planning performance. For the evaluated CALVIN tasks, differences in generated depth error and task success remain relatively small across the tested latent bitrates. The latent bitrate and generative FID are therefore poor predictors of downstream task success in our current setting. One explanation is that the CALVIN tasks and scene observations used in these experiments do not require enough information for differences between the latent representations to become important. The Stage 1 systems already achieve low reconstruction errors over most of the evaluated bitrate range, and the manipulation tasks depend on relatively few object states. We believe that the tradeoff could become more important for larger and more diverse datasets with more difficult tasks and scenes. In such settings, the latent representation must preserve more information while the trajectory model has to model a more complex latent distribution.

The CALVIN experiments further show that a single diffusion model can represent different manipulation behaviors and can be steered effectively with CFG. Both absolute and relative robot state representations achieve high success rates over the eight evaluated tasks. The experiments with different sampling horizons further show that the same model architecture can generate useful trajectories over different horizons. CFG is therefore a

simple and effective steering method when the tasks are known during training. However, the task condition is part of the diffusion model itself. The model is trained with task identifiers and learns the corresponding manipulation behaviors from the demonstration data. Adding a new symbolic action or changing the available guidance signals therefore generally requires training or fine-tuning the diffusion model again. This limits the flexibility of CFG when the diffusion model is used as the motion planning component of a symbolic planning system whose domain may change.

The predicate guidance experiments address a different setting. The strongest result is that the predicate classifier does not need to receive the complete generated trajectory. A classifier that receives only the terminal latent variable as trajectory input increases success from 48.4% without guidance to 96.0%. It performs at least as well as classifiers that additionally receive robot states or the complete latent trajectory. At the same time, the gradient analysis shows that propagating the predicate gradient through the denoiser produces non-zero gradients for earlier latent variables and robot states, even though the predicate itself only depends on the terminal latent. The diffusion model therefore provides the connection between the desired final environment state and the robot motion required to reach that state.

This is useful for steering from symbolic states because the predicate classifier does not have to learn what a complete robot trajectory for a symbolic action looks like. It only has to determine whether the generated environment state satisfies the predicate. Such predicates can be trained from labeled environment observations instead of labeled robot trajectories, while the unconditional diffusion model can remain unchanged. In this sense, predicate guidance provides a more flexible interface between symbolic planning and trajectory generation than CFG. A new predicate does not have to be included as a condition during training of the diffusion model. However, this flexibility concerns the steering signal, not the behaviors represented by the diffusion model. Guidance can steer the model toward trajectories that satisfy a new predicate only if suitable robot motions are already represented by the unconditional trajectory distribution. Predicate guidance therefore does not remove the need for training data that contains the relevant objects and manipulation behaviors.

Our experiments also show an important limitation of the current predicate guidance approach. When the predicate is evaluated on the decoded terminal depth image instead of directly on the terminal latent variable, selective success decreases from 96.0% to 70.3%, with substantial differences between model and classifier seeds. The result is still better than the unguided baseline, but the strongest performance is obtained when the predicate classifier is trained directly on the same latent representation used by the trajectory model.

This means that our experiments show that guidance from observation-only predicates is possible, but they do not yet show the same performance for general classifiers operating directly on environment observations. Using a differentiable decoder provides a way to connect such classifiers to the latent trajectory model, but the additional decoder makes the guidance less effective in our experiments.

Finally, the predicate guidance experiments benefit from the fact that the complete task can be represented within a fixed sampling horizon. This allows the terminal predicate to influence the complete generated robot trajectory directly. For longer tasks, using a receding window makes this more difficult because the desired final state is not contained in the first generated trajectory segments. Uniformly reducing complete demonstrations to a fixed number of frames works for the simple tasks considered here, but can remove important parts of more complex motions. The current results therefore establish predicate guidance for full-task trajectories in a controlled setting. Extending the same mechanism to more complex and diverse tasks remains an important part of scaling the approach.

7. Summary

In this thesis, we investigate how a single multi-task diffusion model can generate robot motions for different symbolic actions and be steered from symbolic plans. We focus on the motion planning component of TAMP and assume that a symbolic planner provides a sequence of symbolic actions and states. Our approach uses two stages. First, we learn latent representations of depth images with an encoder, diffusion prior, and decoder. We then train a diffusion model to jointly generate trajectories of latent environment representations and robot states, conditioned on the initial environment observation and robot state.

We compare deterministic and diffusion decoders and investigate how the information content of the latent representations affects reconstruction and downstream diffusion sampling. We find that the latent bitrate can be controlled through the Stage 1 loss weighting. Higher bitrates improve reconstruction and generally lead to worse generative FID, but we find no clear relation between bitrate and downstream task success. We also compare robot state representations and sampling horizons on CALVIN. With CFG and absolute robot states, the model achieves an aggregated success rate of 93.91% over eight tasks.

Finally, we investigate guidance from predicates that describe the desired symbolic environment state. We train an unconditional diffusion model and predicate classifiers that are evaluated on parts of the generated trajectory. Propagating the classifier gradients through the denoiser allows a predicate evaluated only on the terminal latent variable to influence the complete trajectory. In our Isaac Lab environment, guidance using only the terminal latent increases success from 48.4% without guidance to 96.0%. Guidance from decoded depth images achieves 70.3%.

8. Outlook

Our guidance experiments use only two manipulation tasks and a small number of controlled scene configurations. Future work should evaluate whether the same approach remains effective when the diffusion model represents more object types, more manipulation behaviors, and more diverse scenes.

A second open problem is the difference between the task horizon and the sampling horizon. Our predicate guidance experiments use a fixed task horizon and uniformly downsample demonstrations to sparse trajectories. This allows a predicate on the final state to influence the complete generated trajectory, but sparse trajectories can omit important parts of the motion. More sophisticated approaches could generate trajectories at multiple temporal resolutions or first generate a sparse long-horizon trajectory and subsequently generate denser intermediate states. This could preserve the ability to guide from predicates on distant final states without requiring the diffusion model to generate a dense complete task trajectory at once.

Another important step is integrating the proposed components into a complete TAMP system. In this thesis, we assume that a symbolic planner provides the desired symbolic actions and states. A complete system should connect the symbolic planner, predicate classifiers, and diffusion trajectory model directly.

Symbolic states and task descriptions do not need to be represented by a single predicate. More complex tasks can require several predicates to hold at the same time. Future work should therefore investigate how multiple predicate guidance signals can be combined. An important question is how guidance signals interact when their gradients support or conflict with each other.

Finally, the current experiments obtain the strongest predicate guidance when the classifier operates directly on the latent representation. Guidance through a predicate classifier trained on decoded depth images is less reliable in our experiments. Future work should



investigate whether more accurate decoders or stronger observation-based predicate models can close this gap.

Bibliography

- [1] Tom Silver et al. *Learning Neuro-Symbolic Skills for Bilevel Planning*. Oct. 12, 2022. DOI: 10.48550/arXiv.2206.10680. arXiv: 2206.10680 [cs]. URL: <http://arxiv.org/abs/2206.10680> (visited on 11/26/2025). Pre-published.
- [2] Rohan Chitnis et al. *Learning Neuro-Symbolic Relational Transition Models for Bilevel Planning*. June 30, 2022. DOI: 10.48550/arXiv.2105.14074. arXiv: 2105.14074 [cs]. URL: <http://arxiv.org/abs/2105.14074> (visited on 01/29/2026). Pre-published.
- [3] Tsung-Wei Ke, Nikolaos Gkanatsios, and Katerina Fragkiadaki. *3D Diffuser Actor: Policy Diffusion with 3D Scene Representations*. arXiv.org. Feb. 16, 2024. URL: <https://arxiv.org/abs/2402.10885v3> (visited on 02/17/2026).
- [4] Huy Ha, Pete Florence, and Shuran Song. *Scaling Up and Distilling Down: Language-Guided Robot Skill Acquisition*. Oct. 1, 2023. DOI: 10.48550/arXiv.2307.14535. arXiv: 2307.14535 [cs.R0]. URL: <http://arxiv.org/abs/2307.14535> (visited on 10/03/2026). Pre-published.
- [5] Xiaolin Fang et al. *DiMSam: Diffusion Models as Samplers for Task and Motion Planning under Partial Observability*. arXiv.org. June 22, 2023. URL: <https://arxiv.org/abs/2306.13196v3> (visited on 04/13/2026).
- [6] Jonathan Heek et al. *Unified Latents (UL): How to Train Your Latents*. Feb. 19, 2026. DOI: 10.48550/arXiv.2602.17270. arXiv: 2602.17270 [cs.LG]. URL: <http://arxiv.org/abs/2602.17270> (visited on 07/02/2026). Pre-published.
- [7] Corinna Cortes and Vladimir Vapnik. “Support-Vector Networks”. In: *Machine Learning* 20.3 (Sept. 1, 1995), pp. 273–297. ISSN: 1573-0565. DOI: 10.1007/BF00994018. URL: <https://doi.org/10.1007/BF00994018> (visited on 08/01/2026).

-
-
- [8] D. E. Rumelhart and D. Zipser. “Feature Discovery by Competitive Learning”. In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*. Cambridge, MA, USA: MIT Press, Jan. 3, 1986, pp. 151–193. ISBN: 978-0-262-68053-0.
- [9] Mark A. Kramer. “Nonlinear Principal Component Analysis Using Autoassociative Neural Networks”. In: *AIChE Journal* 37.2 (1991), pp. 233–243. ISSN: 1547-5905. DOI: 10.1002/aic.690370209. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/aic.690370209> (visited on 08/01/2026).
- [10] Jascha Sohl-Dickstein et al. *Deep Unsupervised Learning Using Nonequilibrium Thermodynamics*. Nov. 18, 2015. DOI: 10.48550/arXiv.1503.03585. arXiv: 1503.03585 [cs]. URL: <http://arxiv.org/abs/1503.03585> (visited on 04/13/2026). Pre-published.
- [11] Jonathan Ho, Ajay Jain, and Pieter Abbeel. *Denoising Diffusion Probabilistic Models*. Dec. 16, 2020. DOI: 10.48550/arXiv.2006.11239. arXiv: 2006.11239 [cs]. URL: <http://arxiv.org/abs/2006.11239> (visited on 04/13/2026). Pre-published.
- [12] W. Feller. “On the Theory of Stochastic Processes, with Particular Reference to Applications”. In: *Proceedings of the First Berkeley Symposium on Mathematical Statistics and Probability (August, 1945 and January, 1946)* (1945). URL: <https://digicoll.lib.berkeley.edu/record/112810> (visited on 08/02/2026).
- [13] Calvin Luo. *Understanding Diffusion Models: A Unified Perspective*. Aug. 25, 2022. DOI: 10.48550/arXiv.2208.11970. arXiv: 2208.11970 [cs]. URL: <http://arxiv.org/abs/2208.11970> (visited on 04/14/2026). Pre-published.
- [14] Alex Nichol and Prafulla Dhariwal. *Improved Denoising Diffusion Probabilistic Models*. Feb. 18, 2021. DOI: 10.48550/arXiv.2102.09672. arXiv: 2102.09672 [cs]. URL: <http://arxiv.org/abs/2102.09672> (visited on 04/13/2026). Pre-published.
- [15] Tim Salimans and Jonathan Ho. *Progressive Distillation for Fast Sampling of Diffusion Models*. June 7, 2022. DOI: 10.48550/arXiv.2202.00512. arXiv: 2202.00512 [cs]. URL: <http://arxiv.org/abs/2202.00512> (visited on 04/13/2026). Pre-published.
- [16] Bradley Efron. “Tweedie’s Formula and Selection Bias”. In: *Journal of the American Statistical Association* 106.496 (Dec. 2011), pp. 1602–1614. ISSN: 0162-1459, 1537-274X. DOI: 10.1198/jasa.2011.tm11181. URL: <http://www.>

-
- tandfonline.com/doi/abs/10.1198/jasa.2011.tm11181 (visited on 08/07/2026).
- [17] Yang Song and Stefano Ermon. *Generative Modeling by Estimating Gradients of the Data Distribution*. Oct. 10, 2020. DOI: 10.48550/arXiv.1907.05600. arXiv: 1907.05600 [cs.LG]. URL: <http://arxiv.org/abs/1907.05600> (visited on 08/02/2026). Pre-published.
- [18] Yang Song et al. *Score-Based Generative Modeling through Stochastic Differential Equations*. Feb. 10, 2021. DOI: 10.48550/arXiv.2011.13456. arXiv: 2011.13456 [cs]. URL: <http://arxiv.org/abs/2011.13456> (visited on 04/13/2026). Pre-published.
- [19] Jiaming Song, Chenlin Meng, and Stefano Ermon. *Denoising Diffusion Implicit Models*. Oct. 5, 2022. DOI: 10.48550/arXiv.2010.02502. arXiv: 2010.02502 [cs]. URL: <http://arxiv.org/abs/2010.02502> (visited on 04/13/2026). Pre-published.
- [20] Prafulla Dhariwal and Alexander Nichol. “Diffusion Models Beat GANs on Image Synthesis”. In: *Advances in Neural Information Processing Systems*. Vol. 34. Curran Associates, Inc., 2021, pp. 8780–8794. URL: <https://papers.nips.cc/paper/2021/hash/49ad23d1ec9fa4bd8d77d02681df5cfa-Abstract.html> (visited on 02/25/2026).
- [21] Andreas Lugmayr et al. *RePaint: Inpainting Using Denoising Diffusion Probabilistic Models*. Aug. 31, 2022. DOI: 10.48550/arXiv.2201.09865. arXiv: 2201.09865 [cs.CV]. URL: <http://arxiv.org/abs/2201.09865> (visited on 08/23/2026). Pre-published.
- [22] Jonathan Ho and Tim Salimans. *Classifier-Free Diffusion Guidance*. July 26, 2022. DOI: 10.48550/arXiv.2207.12598. arXiv: 2207.12598 [cs.LG]. URL: <http://arxiv.org/abs/2207.12598> (visited on 08/02/2026). Pre-published.
- [23] Robin Rombach et al. *High-Resolution Image Synthesis with Latent Diffusion Models*. Apr. 13, 2022. DOI: 10.48550/arXiv.2112.10752. arXiv: 2112.10752 [cs]. URL: <http://arxiv.org/abs/2112.10752> (visited on 04/13/2026). Pre-published.
- [24] Michael Janner et al. *Planning with Diffusion for Flexible Behavior Synthesis*. arXiv.org. May 20, 2022. URL: <https://arxiv.org/abs/2205.09991v2> (visited on 04/13/2026).

-
-
- [25] Cheng Chi et al. *Diffusion Policy: Visuomotor Policy Learning via Action Diffusion*. Mar. 14, 2024. DOI: 10.48550/arXiv.2303.04137. arXiv: 2303.04137 [cs.R0]. URL: <http://arxiv.org/abs/2303.04137> (visited on 08/21/2026). Pre-published.
- [26] Alec Radford et al. *Learning Transferable Visual Models From Natural Language Supervision*. Feb. 26, 2021. DOI: 10.48550/arXiv.2103.00020. arXiv: 2103.00020 [cs]. URL: <http://arxiv.org/abs/2103.00020> (visited on 02/17/2026). Pre-published.
- [27] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. *PDDLStream: Integrating Symbolic Planners and Blackbox Samplers via Optimistic Adaptive Planning*. Mar. 23, 2020. DOI: 10.48550/arXiv.1802.08705. arXiv: 1802.08705 [cs]. URL: <http://arxiv.org/abs/1802.08705> (visited on 02/25/2026). Pre-published.
- [28] Arpit Bansal et al. *Universal Guidance for Diffusion Models*. arXiv.org. Feb. 14, 2023. URL: <https://arxiv.org/abs/2302.07121v1> (visited on 06/02/2026).
- [29] Maximilian Du and Shuran Song. *DynaGuide: Steering Diffusion Polices with Active Dynamic Guidance*. Nov. 9, 2025. DOI: 10.48550/arXiv.2506.13922. arXiv: 2506.13922 [cs.R0]. URL: <http://arxiv.org/abs/2506.13922> (visited on 08/21/2026). Pre-published.
- [30] Maxime Oquab et al. *DINOv2: Learning Robust Visual Features without Supervision*. Feb. 2, 2024. DOI: 10.48550/arXiv.2304.07193. arXiv: 2304.07193 [cs.CV]. URL: <http://arxiv.org/abs/2304.07193> (visited on 09/01/2026). Pre-published.
- [31] Emiel Hoogeboom et al. *Simpler Diffusion (SiD2): 1.5 FID on ImageNet512 with Pixel-Space Diffusion*. Mar. 22, 2025. DOI: 10.48550/arXiv.2410.19324. arXiv: 2410.19324 [cs.CV]. URL: <http://arxiv.org/abs/2410.19324> (visited on 09/01/2026). Pre-published.
- [32] Kaiming He et al. *Deep Residual Learning for Image Recognition*. Dec. 10, 2015. DOI: 10.48550/arXiv.1512.03385. arXiv: 1512.03385 [cs.CV]. URL: <http://arxiv.org/abs/1512.03385> (visited on 09/29/2026). Pre-published.
- [33] Chang Chen et al. *Simple Hierarchical Planning with Diffusion*. Version 1. Jan. 5, 2024. DOI: 10.48550/arXiv.2401.02644. arXiv: 2401.02644 [cs.LG]. URL: <http://arxiv.org/abs/2401.02644> (visited on 08/05/2026). Pre-published.

-
-
- [34] Ranjay Krishna et al. *Visual Genome: Connecting Language and Vision Using Crowdsourced Dense Image Annotations*. arXiv.org, Feb. 23, 2016. URL: <https://arxiv.org/abs/1602.07332v1> (visited on 04/09/2026).
- [35] Sami Haddadin et al. “The Franka Emika Robot: A Reference Platform for Robotics Research and Education”. In: *IEEE Robotics & Automation Magazine* 29.2 (June 2022), pp. 46–64. ISSN: 1558-223X. DOI: 10.1109/MRA.2021.3138382. URL: <https://ieeexplore.ieee.org/document/9721535> (visited on 10/08/2026).
- [36] NVIDIA. *Isaac Sim*. Version 6.0.1. Oct. 1, 2026. URL: <https://github.com/isaac-sim/IsaacSim> (visited on 10/01/2026).
- [37] Mayank Mittal et al. *Isaac Lab - A GPU-Accelerated Simulation Framework for Multi-Modal Robot Learning*. Version 3.0.0. 2025. DOI: 10.48550/arXiv.2511.04831. arXiv: 2511.04831. URL: <https://arxiv.org/abs/2511.04831> (visited on 10/01/2026).
- [38] *Isaac-Sim/IsaacLab-Arena*. NVIDIA Isaac Sim, Oct. 1, 2026. URL: <https://github.com/isaac-sim/IsaacLab-Arena> (visited on 10/01/2026).
- [39] Oier Mees et al. *CALVIN: A Benchmark for Language-Conditioned Policy Learning for Long-Horizon Robot Manipulation Tasks*. arXiv.org, Dec. 6, 2021. URL: <https://arxiv.org/abs/2112.03227v4> (visited on 10/01/2026).
- [40] Martin Heusel et al. *GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium*. Jan. 12, 2018. DOI: 10.48550/arXiv.1706.08500. arXiv: 1706.08500 [cs.LG]. URL: <http://arxiv.org/abs/1706.08500> (visited on 10/08/2026). Pre-published.

A. Network Architectures

This appendix describes the networks used for latent representation learning, trajectory sampling, predicate guidance, and state estimation. We specify the operations and their connectivity separately from the channel widths and block counts, which are denoted symbolically. Widths c_i and block counts n_i are ordered in the direction of computation in each table. Repeated blocks have independent parameters. Symbols for widths, block counts, and diffusion-time embedding dimensions are local to each network. The stage counts S_e , S_d , S_q , and S_s equal the lengths of the corresponding channel-width tuples in Appendix B.

A.1. Notation and Building Blocks

We omit the batch dimension and write tensor shapes with the channel dimension last. A depth image has shape $H_x \times W_x \times 1$, a spatial latent has shape $h_z \times w_z \times c_z$, and a robot state has dimension d_r . A trajectory contains H frames. Convolutions and linear layers include a bias. Normalization layers have learned affine parameters. Unless stated otherwise, convolutions use stride one and zero padding that preserves the input resolution. Spatial stride-two convolutions use a 3×3 kernel and padding one, so successive resolutions satisfy

$$h_{i+1} = \lceil h_i/2 \rceil, \quad w_{i+1} = \lceil w_i/2 \rceil. \quad (\text{A.1})$$

All bilinear resizing uses the target tensor size with `align_corners=False`. Concatenation is along the channel or feature dimension. Spatial flattening follows channel, height, and width order.

Block	First convolution K_1	Second convolution K_2
Spatial	3×3	3×3
Spatial prior	3×3 , dilation δ	3×3
Latent trajectory	$1 \times 3 \times 3$, spatial dilation δ_s	$3 \times 1 \times 1$, temporal dilation δ_τ
Robot trajectory	Temporal kernel 3, dilation δ_τ	Temporal kernel 3, dilation δ_τ

Table A.1.: Residual-block convolutions. For latent trajectories, kernel dimensions are ordered as trajectory time, image height, and image width. Padding equals dilation along each convolved dimension. All blocks preserve their input shape.

The convolutional residual blocks use group normalization (GN) and SiLU activations. For an input feature tensor v , their common form is

$$u = K_1(\text{SiLU}(\text{GN}(v))), \quad (\text{A.2})$$

$$R(v; e) = v + K_2(\text{SiLU}((1 + s(e)) \odot \text{GN}(u) + b(e))). \quad (\text{A.3})$$

Here K_1 and K_2 are the convolutions in Table A.1. In conditioned blocks, a linear projection of the diffusion-time embedding e produces a scale $s(e)$ and shift $b(e)$ for each channel. These are broadcast over the remaining dimensions. Unconditioned blocks omit this projection and use $s = b = 0$. There is no activation after the residual addition. GN uses the largest divisor of the channel width in $\{32, 16, 8, 4, 2, 1\}$, except for the depth-image predicate classifier, which uses $\{8, 4, 2, 1\}$.

Dilation cycles are read from their first entry at the first residual block and repeated in block order. For Stage 2, both streams use the temporal dilation assigned to the same joint block.

The diffusion networks embed the continuous diffusion time t with a sinusoidal embedding of even dimension d_t . Its entries are $\sin(at\omega_j)$ and $\cos(at\omega_j)$, where $\omega_j = 10000^{-j/\max(d_t/2-1, 1)}$ for $j = 0, \dots, d_t/2 - 1$. The embedding is followed by a linear layer, SiLU, and a second linear layer, each preserving dimension d_t . Stage 1 uses $a = 1$, while Stage 2 uses $a = 1000$. Diffusion time is distinct from the frame index of a trajectory.

Stage	Operation	Output shape	Repetitions
Input	3×3 convolution	$H_x \times W_x \times c_0$	1
Stage 0	Unconditioned spatial residual block	$H_x \times W_x \times c_0$	n_0
Downsample i	Stride-two convolution, GN, SiLU	$h_i \times w_i \times c_i$	1
Stage i	Unconditioned spatial residual block	$h_i \times w_i \times c_i$	n_i
Align scales	Stride-two convolution, GN, SiLU; retain width c_i	$h_z \times w_z \times c_i$	$S_e - 1 - i$
Aggregate	Concatenate aligned features	$h_z \times w_z \times \sum_i c_i$	1
Latent	1×1 convolution, GN, SiLU	$h_z \times w_z \times c_z$	1

Table A.2.: Depth encoder. The alignment operation is applied separately to each retained feature map. The latent resolution is the coarsest encoder resolution, $(h_z, w_z) = (h_{S_e-1}, w_{S_e-1})$, with $(h_0, w_0) = (H_x, W_x)$.

A.2. Stage 1: Latent Representation Learning

A.2.1. Depth Encoder

The encoder maps a depth image x to a spatial latent z . It extracts features at progressively lower resolutions and combines features from all resolutions at the coarsest scale. Table A.2 describes the network. The rows for downsampling and residual processing are applied alternately for $i = 1, \dots, S_e - 1$, where S_e is the number of encoder stages. Features are retained after the residual blocks at every stage.

A.2.2. Latent Diffusion Prior

The prior denoiser takes a noised latent $z^{(t)}$ and diffusion time t and predicts the clean latent. Its spatial resolution remains constant throughout the network. Table A.3 gives the architecture. The dilation δ of the first convolution in each residual block follows a repeated dilation sequence.

Stage	Operation	Output shape	Repetitions
Input	3×3 convolution	$h_z \times w_z \times c_p$	1
Processing	Time-conditioned spatial prior residual block	$h_z \times w_z \times c_p$	N_p
Output	GN, SiLU, 3×3 convolution	$h_z \times w_z \times c_z$	1

Table A.3.: Stage 1 latent prior denoiser with hidden width c_p and N_p residual blocks. The diffusion-time embedding conditions every block.

A.2.3. Diffusion Decoder

The diffusion decoder predicts a clean depth image from a noised image $x^{(t)}$, a spatial latent $z^{(0)}$, and diffusion time t . It maintains four parallel resolution branches, including a full-resolution branch throughout the network. Figure A.1 shows their connectivity, and Table A.4 specifies the operations.

Fusion adds the feature at the target resolution to transformed features from each other resolution. A transformation from a finer branch j to a coarser branch i uses $i - j$ stride-two convolutions. Intermediate convolutions retain the source width and are followed by GN and SiLU; the last convolution maps to the target width without normalization or activation. A transformation from a coarser branch to a finer branch uses bilinear resizing followed by a 3×3 convolution. All contributions are computed from the features before fusion.

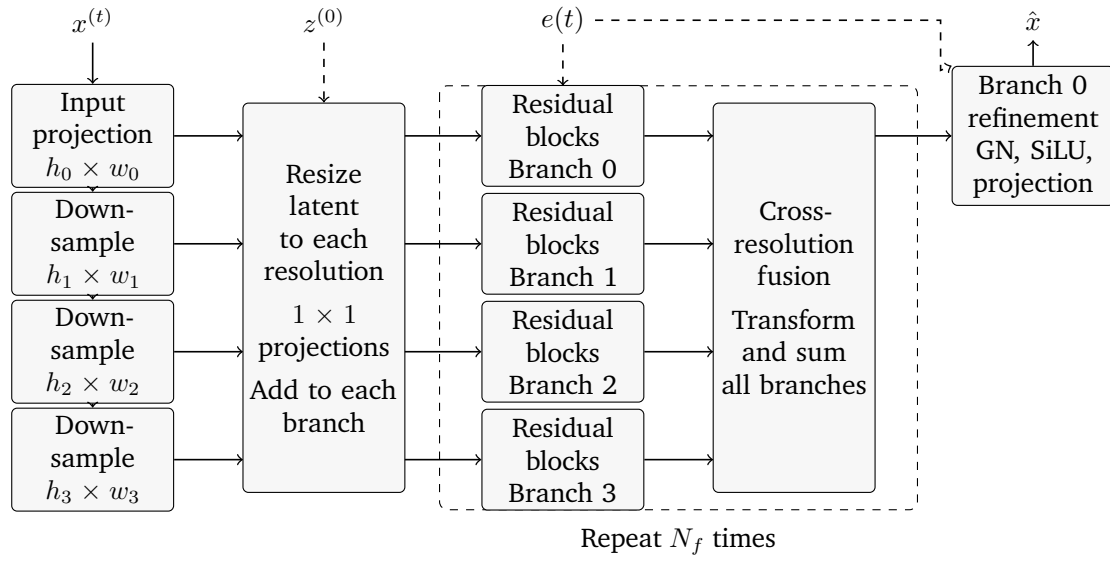


Figure A.1.: Diffusion decoder. The noised image initializes four resolution branches. Resized and projected latent features are added to each branch once, before the repeated processing and fusion stages. Every fusion stage exchanges information between all resolutions. Only the full-resolution branch is used for the final depth prediction.

Stage	Operation	Output shape	Repetitions
Input	3×3 convolution	$H_x \times W_x \times c_0$	1
Initialize branch i	Stride-two convolution, GN, SiLU from branch $i - 1$	$h_i \times w_i \times c_i$	1
Latent conditioning	Resize $z^{(0)}$, 1×1 convolution, add to branch i	$h_i \times w_i \times c_i$	1
Branch processing	Time-conditioned spatial residual block	$h_i \times w_i \times c_i$	n_f
Fusion	Sum transformed features from all branches	$h_i \times w_i \times c_i$	1
Refinement	Time-conditioned spatial residual block on branch 0	$H_x \times W_x \times c_0$	n_{out}
Output	GN, SiLU, 1×1 convolution	$H_x \times W_x \times 1$	1

Table A.4.: Diffusion decoder. Branches are indexed by $i = 0, \dots, 3$, with $(h_0, w_0) = (H_x, W_x)$. Initialization applies to $i = 1, 2, 3$. Branch processing and fusion together are repeated N_f times, giving $N_f n_f$ residual blocks per branch before output refinement.

A.2.4. Deterministic Decoder

The deterministic decoder maps $z^{(0)}$ directly to a depth image. It uses residual processing at each resolution and progressively upsamples the features, as specified in Table A.5. It receives neither a noised image nor diffusion time, and has no connections to intermediate encoder features. Its target resolutions reverse the stride-two resolution sequence for the output image, including the rounding defined in Section A.1.

Stage	Operation	Output shape	Repetitions
Input	3×3 convolution	$h_z \times w_z \times c_0$	1
Stage 0	Unconditioned spatial residual block	$h_z \times w_z \times c_0$	n_0
Upsample i	Bilinear resizing, 3×3 convolution	$\tilde{h}_i \times \tilde{w}_i \times c_i$	1
Stage i	Unconditioned spatial residual block	$\tilde{h}_i \times \tilde{w}_i \times c_i$	n_i
Output	3×3 convolution	$H_x \times W_x \times 1$	1

Table A.5.: Deterministic depth decoder. Upsampling and residual processing are applied alternately for $i = 1, \dots, S_d - 1$. The resolution sequence runs from $(\tilde{h}_0, \tilde{w}_0) = (h_z, w_z)$ to $(\tilde{h}_{S_d-1}, \tilde{w}_{S_d-1}) = (H_x, W_x)$. The output convolution has no activation.

A.3. Stage 2: Diffusion Trajectory Sampler

The Stage 2 denoiser jointly predicts clean latent and robot-state trajectories from their noised counterparts and diffusion time. Figure A.2 shows its two streams, and Table A.6 gives the architecture. The latent stream preserves the spatial latent grid and uses factorized spatial and temporal convolutions. The robot stream uses temporal convolutions. Neither stream changes the trajectory length or spatial resolution. The latents are flattened only when packing them together with robot states for the diffusion process. The widths c_ℓ and c_r describe hidden features; the input and output widths remain c_z and d_r . Each of the N_2 joint blocks contains one residual block per stream and one bidirectional fusion operation.

For latent features v and robot features u after residual processing, the fusion operation is

$$v' = (1 + s_r(u)) \odot v + b_r(u), \quad (\text{A.4})$$

$$u' = u + P_\ell(\text{mean}_{h,w}(v)). \quad (\text{A.5})$$

A temporal kernel-1 convolution produces s_r and b_r , which are broadcast over the spatial dimensions. Another temporal kernel-1 convolution P_ℓ projects the pooled latent features to the robot-stream width. Both directions use the features before fusion. These fusion projections are initialized with zero weights and biases.

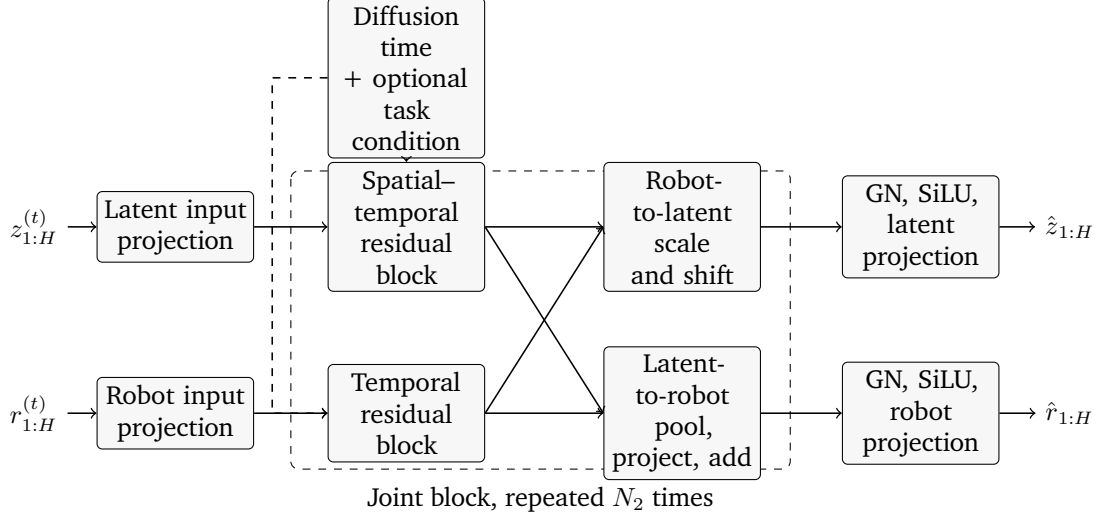


Figure A.2.: Stage 2 trajectory denoiser. Each joint block contains a residual block in each stream followed by bidirectional fusion. Robot features modulate latent features through channel-wise scale and shift, while spatially averaged latent features are projected and added to the robot stream. Diffusion time, together with an optional task condition, conditions both residual blocks.

Stage	Operation	Output shape	Repetitions
Latent input	$1 \times 1 \times 1$ convolution	$H \times h_z \times w_z \times c_\ell$	1
Robot input	Temporal kernel-1 convolution	$H \times c_r$	1
Joint block	Conditioned latent and robot residual blocks, then fusion	Unchanged	N_2
Latent output	GN, SiLU, $1 \times 1 \times 1$ convolution	$H \times h_z \times w_z \times c_z$	1
Robot output	GN, SiLU, temporal kernel-1 convolution	$H \times d_r$	1

Table A.6.: Stage 2 denoiser with latent-stream width c_ℓ and robot-stream width c_r . Each joint block uses the trajectory residual blocks in Table A.1. Spatial and temporal dilations follow their respective repeated sequences.

For task-conditioned models, the conditioning vector c is combined with the diffusion-time embedding as $e' = e + d_t^{-1/2} P_c(c)$. The projection P_c is linear, or the identity when c already has dimension d_t . CALVIN task identifiers are mapped to learned embeddings, while target conditions in the Isaac Lab Environment are supplied as encoded vectors. Unconditional models use $e' = e$. Conditioning on the initial observation and robot state is implemented by trajectory inpainting, as described in Section 4.2.

A.4. Predicate Classifiers

A.4.1. Latent Predicate Classifier

The four latent-input classifiers in Section 5.5 share the architecture in Table A.7. Table A.8 specifies their inputs. Each frame’s latent is flattened directly, without a convolutional backbone or spatial pooling. The classifier optionally concatenates the corresponding robot state before projecting each frame to a token of dimension d_c . The latent and robot inputs are supplied in their original representation, after reversing the Stage 2 normalization.

Each Transformer layer uses n_h attention heads and layer normalization before multi-head self-attention and before the feed-forward subnetwork. Attention operates over all input frames. The feed-forward subnetwork has hidden dimension $4d_c$ and GELU activation. Dropout is applied to attention weights, feed-forward hidden activations, and the outputs of both residual branches. The Transformer and head use dropout probability p_c . The learned position-embedding table has $H_{\max}$ rows of dimension d_c ; its first L rows are added to the input tokens, with $1 \leq L \leq H_{\max}$. A sigmoid converts the final logit to a predicate probability for guidance.

Stage	Operation	Output shape	Repetitions
Frame features	Flatten latent; optionally concatenate robot state	$L \times d_{\text{in}}$	1
Tokens	Linear projection, add learned position embeddings	$L \times d_c$	1
Temporal processing	Transformer encoder layer	$L \times d_c$	N_c
Readout	Select last token, concatenate query q	$d_c + d_q$	1
Head	Layer normalization, linear layer, GELU, dropout	d_c	1
Output	Linear layer	1	1

Table A.7.: Latent predicate classifier. The frame-feature dimension is $d_{\text{in}} = h_z w_z c_z$ for latent-only inputs and $d_{\text{in}} = h_z w_z c_z + d_r$ when robot inputs are included. L is the number of input frames and d_q is the query dimension. The scalar output is a logit.

Variant	Input frames	L
Terminal latent	z_H	1
Terminal latent and robot	(z_H, r_H)	1
Latent trajectory	$(z_1, \dots, z_H)$	H
Latent and robot trajectory	$((z_1, r_1), \dots, (z_H, r_H))$	H

Table A.8.: Input variants of the latent predicate classifier. The Transformer is retained for the single-frame variants. All variants also receive the object query q at the classification head.

A.4.2. Depth-Image Predicate Classifier

For guidance through the deterministic decoder, the predicate classifier receives a single depth image and the object query. Its convolutional backbone is independent of the Stage 1 encoder. Adaptive average pooling to a spatial grid retains coarse spatial information before classification. Table A.9 specifies the architecture. Input depth images use the same fixed normalization as the Stage 1 system. The head uses dropout probability p_q .

Stage	Operation	Output shape	Repetitions
Stage i	Stride-two convolution, GN, SiLU	$h_i \times w_i \times c_i$	1 per stage
Pooling	Adaptive average pooling	$g_q \times g_q \times c_{S_i}$	1
Features	Flatten, concatenate query q	$g_q^2 c_{S_i} + d_q$	1
Head	Linear layer, SiLU, dropout	d_q^{hid}	1
Output	Linear layer	1	1

Table A.9.: Depth-image predicate classifier with S_q convolutional stages, indexed by $i = 1, \dots, S_q$, and pooling-grid size g_q , with $(h_0, w_0) = (H_x, W_x)$. The scalar output is a logit, converted to a probability by a sigmoid when used for guidance.

A.5. CALVIN State Estimator

The state estimator predicts the continuous states of the drawer, slider, and switch from a depth image. It uses a convolutional backbone and spatial grid pooling, followed by a regression head, as specified in Table A.10. Its depth preprocessing is the same as for the CALVIN Stage 1 system. The output coordinates are multiplied by fixed characteristic-displacement scales to obtain predictions in meters. For the FID metrics, we extract the hidden vector after the regression head’s SiLU activation, before the final linear layer and output scaling.

Stage	Operation	Output shape	Repetitions
Stage i	Stride-two convolution, SiLU	$h_i \times w_i \times c_i$	1 per stage
Pooling	Adaptive average pooling, flatten	$g_s^2 c_{S_s}$	1
FID features	Linear layer, SiLU	d_s	1
Regression	Linear layer, coordinate-wise output scaling	3	1

Table A.10.: CALVIN state estimator with S_s convolutional stages, indexed by $i = 1, \dots, S_s$, pooling-grid size g_s , and hidden dimension d_s , with $(h_0, w_0) = (H_x, W_x)$. The backbone has no normalization layers. The same trained estimator provides features for reconstruction and generative FID.

B. Hyperparameters

This appendix lists the model dimensions and training settings used in the experiments. Network operations are specified in Appendix A; channel widths and block counts below follow the order of computation defined there. Each width tuple specifies one network’s stages or branches; block-count tuples assign the number of residual blocks to the corresponding entries. Batch sizes refer to the number of examples per optimizer step, including gradient accumulation. All models use AdamW with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$.

B.1. CALVIN Latent Representation Experiments

B.1.1. Stage 1

The diffusion-decoder and deterministic-decoder bitrate ablations use the same encoder and latent prior. Table B.1 gives their dimensions and the settings for each decoder, using the architectures in Section A.2. Depth images have resolution 200×200 and are normalized as $x = (d - 4.246993 \text{ m}) / (0.483500 \text{ m})$, where d is metric depth. Reconstruction losses exclude invalid depth pixels. The decoder receives latents perturbed at fixed log-SNR $\lambda = 10$. The prior and diffusion decoder predict clean data, with diffusion times sampled uniformly from $[0, 1]$ and schedule $\lambda(t) = 10 - 20t$. The five encoder stages have spatial side lengths (200, 100, 50, 25, 13). The deterministic decoder follows the reverse sequence, while the diffusion decoder’s four branches have side lengths (200, 100, 50, 25).

Table B.2 lists the optimization settings. The decoder loss factors and sigmoid biases are varied as reported in Tables 5.2 and 5.3. The prior loss has weight one and includes the terminal KL term. The deterministic decoder uses MSE in normalized depth space, without diffusion-time weighting. For both decoder types, we select the checkpoint with the lowest validation loss, evaluated after each epoch.

Setting	Value
<i>Shared encoder and prior</i>	
Latent shape $h_z \times w_z \times c_z$	$13 \times 13 \times 2$
Encoder widths (c_i)	(32, 64, 128, 192, 256)
Encoder blocks (n_i)	(1, 1, 1, 2, 2)
Prior width c_p	128
Prior residual blocks N_p	8
Prior time-embedding dimension d_t	256
Prior dilation cycle	(1, 2, 4, 8)
<i>Diffusion decoder</i>	
Branch widths (c_i)	(32, 64, 128, 192)
Time-embedding dimension d_t	256
Fusion stages N_f	2
Residual blocks per branch and fusion stage n_f	2
Output refinement blocks n_{out}	1
<i>Deterministic decoder</i>	
Stage widths (c_i)	(128, 96, 64, 32, 16)
Residual blocks (n_i)	(2, 1, 1, 1, 1)

Table B.1.: Network dimensions for the two CALVIN Stage 1 bitrate ablations.

Setting	Diffusion decoder	Deterministic decoder
Initial learning rate	10^{-3}	10^{-3}
Weight decay	10^{-4}	0
Learning-rate schedule	Cosine	Cosine
Final learning rate	2×10^{-6}	2×10^{-6}
Optimizer steps	30,000	30,000
Batch size	72	64
Gradient norm limit	8	8
Numerical precision	Mixed bfloat16	Float32

Table B.2.: Stage 1 training settings for the CALVIN bitrate ablations. Cosine annealing spans the full training run. Diffusion-decoder models use batches of 24 with three accumulation steps, except for $c_{lf} = 40$, which uses batches of 12 with six accumulation steps. Deterministic-decoder models do not accumulate gradients.

B.1.2. Stage 2

The Stage 2 models used for gFID share the settings in Table B.3 across all eight Stage 1 systems. These dimensions instantiate the denoiser in Table A.6. They use unconditional clean-data prediction with the same noise schedule and uniform diffusion-time sampling as Stage 1. Training uses overlapping windows with stride one. Latent and robot features are standardized using training-set means and standard deviations. The first latent and robot state are inpainted and excluded from the loss. The latent and robot losses are means over their respective feature dimensions and receive weights 0.75 and 0.25, respectively. We select the checkpoint with the lowest validation loss, evaluated after each epoch, and use deterministic DDIM sampling with 32 steps.

The subsequent CALVIN simulation experiments retain these network dimensions and optimization settings and add task conditioning with CFG. Each of the eight task identifiers has a learned embedding of dimension $d_t = 32$, so the conditioning projection P_c is the identity. The robot-representation and horizon comparisons use the same settings, with the representation and horizon varied as specified in the main text.

Setting	Value
Latent-stream width c_ℓ	64
Robot-stream width c_r	32
Time-embedding dimension d_t	32
Joint blocks N_2	8
Spatial dilation cycle	(1, 2, 4, 8)
Temporal dilation cycle	(1, 2, 4, 1)
Sampling horizon H	10
Robot representation	Relative pose, 10 dimensions
Latent/robot loss weights	0.75/0.25
Initial learning rate	10^{-3}
Weight decay	10^{-4}
Learning-rate schedule	Cosine over 20,000 steps
Final learning rate	2×10^{-6}
Optimizer steps	20,000
Batch size	128
Gradient norm limit	2
Numerical precision	Float32

Table B.3.: Shared CALVIN Stage 2 settings for gFID evaluation. The relative robot representation uses position and 6D orientation relative to the first frame, together with the gripper state.

B.1.3. State Estimator for FID

The same depth-based state estimator supplies the features for all CALVIN rFID and gFID evaluations. It uses the depth normalization given above. Its architecture is specified in Table A.10. Table B.4 specifies the network dimensions and training settings. The regression loss is the mean squared error after dividing each target error by its characteristic displacement: 0.16 m for the drawer, 0.28 m for the slider, and 0.085 m for the switch. FID uses the 256-dimensional hidden features after the SiLU activation, rather than the three regression outputs. Validation is performed every 600 training steps, and the checkpoint with the lowest validation loss is selected. Training is limited to 1,800 optimizer steps or 12 minutes, whichever is reached first.

Setting	Value
Convolutional widths (c_i)	(32, 64, 128, 128)
Pooling grid $g_s \times g_s$	10×10
Hidden dimension d_s	256
Regression targets	Drawer, slider, switch
Learning rate	10^{-3} , constant
Weight decay	10^{-2}
Batch size	64
Numerical precision	Mixed float16

Table B.4.: State-estimator settings for CALVIN FID metrics.

B.2. Predicate Guidance

B.2.1. Isaac Lab Environment: Representation and Trajectory Models

The experiments in the Isaac Lab Environment use a separately trained Stage 1 system with a deterministic decoder. Table B.5 gives its settings. Depth images have resolution 200×200 and are normalized as $x = (d - 3.793764389 \text{ m}) / (0.705962775 \text{ m})$. The latent shape, fixed latent perturbation, prior dilation cycle, and diffusion schedule are the same as in the CALVIN Stage 1 experiments. The decoder loss factor is $c_{\text{lf}} = 2$. Training first uses cosine annealing for 80,000 steps and then continues for 20,000 steps at a constant learning rate of 2×10^{-6} . We use the checkpoint with the lowest validation loss from the continuation run, with validation after each epoch.

Unless listed in Table B.6, the unconditional Stage 2 models for the Isaac Lab Environment use the architecture and optimization settings in Table B.3. Training uses ten endpoint-preserving sparse trajectories per demonstration, each spanning the complete demonstration with $H = 10$ frames. The initial latent and robot state are inpainted. Latent and robot features are standardized from the training set. We select the checkpoint with the lowest validation loss, evaluated after each epoch.

Setting	Value
Encoder widths (c_i)	(16, 32, 64, 96, 128)
Encoder blocks (n_i)	(1, 1, 1, 1, 1)
Prior width c_p	64
Prior residual blocks N_p	6
Prior time-embedding dimension d_t	32
Decoder widths (c_i)	(128, 96, 64, 32, 16)
Decoder residual blocks (n_i)	(2, 2, 2, 2, 2)
Initial learning rate	10^{-3}
Learning-rate schedule	Cosine over 80,000 steps
Final and continuation learning rate	2×10^{-6}
Continuation steps	20,000
Weight decay	10^{-4}
Batch size	64
Gradient norm limit	8
Numerical precision	Mixed bfloat16

Table B.5.: Stage 1 settings for guidance in the Isaac Lab Environment. The continuation resumes the final checkpoint of the initial training run.

Setting	Value
Robot representation	Absolute position, 3 dimensions
Latent/robot loss weights	0.5/0.5
Batch size	64

Table B.6.: Stage 2 settings for the Isaac Lab Environment that differ from the CALVIN Stage 2 settings in Table B.3.

B.2.2. Latent Predicate Classifiers

The four input variants of the architecture in Table A.7 share the settings in Table B.7. Classifier training and guidance hyperparameters are optimized separately for each input variant. Table B.8 gives the selected learning rates, weight decay, dropout, and guidance scales. All classifiers receive unstandardized latents, and the robot-input variants also receive absolute end effector positions in three dimensions. Thus the frame-feature dimension d_{in} is 338 for latent-only inputs and 341 when robot states are included. Terminal-input variants use $L = 1$ token, while trajectory-input variants use $L = H = 10$ tokens. Each demonstration contributes ten endpoint-preserving sparse histories and up to ten additional histories ending before its final frame. Each history contains ten frames; terminal-input classifiers retain only its final state. Queries cover all objects present in the scene. An object is labeled closed when its normalized openness is at most 0.05.

Training uses binary cross-entropy, with sample weights that give equal weight to each supported query-label combination. We select the checkpoint with the lowest balanced validation loss. Validation is performed after each epoch, and training stops after 12 validation checks without improvement, or when the step or epoch limit is reached.

B.2.3. Guidance During Sampling

All four variants use forward guidance through the denoiser and deterministic DDIM sampling with 32 steps. Clean predictions are not clipped. Guidance is active over the full diffusion-time interval $[0, 1]$. We use the unnormalized gradient of the classifier’s log-probability, with no additional multiplication by $\sigma(t)$ beyond the factor in the clean-prediction update $\hat{\tau}_{\text{guided}} = \hat{\tau} + s\sigma(t)g/\alpha(t)$. Both latent and robot coordinates receive guidance; the initial inpainted state remains fixed. No backward refinement or recurrence is used.

Setting	Value
Token dimension d_c	128
Transformer layers N_c	2
Attention heads n_h	4
Feed-forward hidden dimension	512
Position-embedding length $H_{\max}$	10
Query dimension d_q	4
Learning-rate schedule	Constant
Batch size	256
Maximum optimizer steps	10,000
Maximum epochs	100
Gradient norm limit	1
Numerical precision	Float32

Table B.7.: Shared architecture and training settings for latent predicate classifiers. Terminal-input variants retain the same Transformer as the trajectory-input variants.

Classifier input	Learning rate	Weight decay	Dropout p_c	Guidance scale s
Terminal latent	10^{-4}	10^{-3}	0	2.5
Terminal latent and robot	10^{-4}	10^{-3}	0	0.5
Latent trajectory	5×10^{-4}	10^{-5}	0.2	2.5
Latent and robot trajectory	5×10^{-4}	10^{-5}	0	2.5

Table B.8.: Selected classifier training and guidance hyperparameters for each latent input variant.

B.2.4. Depth-Image Predicate Classifier

The depth-image classifier in Table A.9 uses the settings in Table B.9. It receives normalized 200×200 depth images and the same four-dimensional object query as the latent classifiers. Training uses original depth images, with the same history sampling, predicate labels, and balanced binary cross-entropy as above. The step and epoch limits, gradient clipping, and numerical precision are those in Table B.7; early-stopping patience remains 12 validation checks. However, checkpoint selection and early stopping use the balanced validation loss on images reconstructed by the frozen Stage 1 encoder and decoder. This evaluates the classifier on the inputs used for guidance. Sampling uses the same forward-guidance settings as the latent classifiers, with scale $s = 2.5$ and gradients propagated through the deterministic decoder.

Setting	Value
Convolutional widths (c_i)	(16, 32, 64, 64)
Pooling grid $g_q \times g_q$	6×6
Head hidden dimension d_q^{hid}	64
Dropout p_q	0.2
Learning rate	2×10^{-3} , constant
Weight decay	10^{-3}
Batch size	64
Guidance scale s	2.5

Table B.9.: Depth-image classifier and guidance settings for the decoded depth-image comparison.