SQUEEZE TO PLAN: LEARNING COMPACT VISUAL REPRESENTATIONS FOR EFFICIENT PLANNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Visual foundation models offer robust perceptual priors for robotic manipulation, but it is computationally expensive to plan over their high-dimensional outputs. This inefficiency arises from the fact that for any given task, foundation models tend to capture irrelevant information. For example, a robot performing a pick-and-place need not reason about the visual texture of the table on which it operates. This leads us to ask – how can we develop planners which both benefit from these robust visual priors, and remain fast enough for real-time planning? Towards this goal, we introduce Squeeze-to-Plan (S2P), a framework for compressing the outputs of frozen foundation visual models into compact, task-specific state representations suitable for real-time planning. S2P achieves this using a novel Bisimulation Information Bottleneck objective, to encourage the compressed representations to discard task-irrelevant visual information while retaining task-relevant reward and dynamics information. S2P solves two real-time visual manipulation tasks where prior methods for planning in pretrained visual feature spaces generally fail, while achieving an orders-of-magnitude reduction in time needed to generate a plan. Additionally, we show that S2P's compression does not sacrifice the task-relevant perceptual priors despite its large compression factor, achieving similar robustness to prior methods against test-time perturbations to lighting and object-colour. Our results show that our method can enable efficient planning with pretrained visual features, while preserving their robustness to certain visual changes.

1 INTRODUCTION

Enabling robots to act in unstructured, real-world environments requires control algorithms which can (1) infer the states of complex environments, (2) model the environment dynamics, and (3) decide control actions quickly enough to act before the environment changes. Recent progress in self-supervised visual representation learning and large-scale world modelling has produced pretrained visual encoders that can extract control-relevant information from pixels (Zhou et al., 2025; Assran et al., 2025; Agarwal et al., 2025; Bruce et al., 2024).

While these foundation vision models provide a promising direction to address challenges (1) and (2), it is computationally expensive to incorporate them into world models, and predict how their high-dimensional features evolve over time (Assran et al., 2025; Zhou et al., 2025; Kim et al., 2026). Optimizing a single action plan with a world model can require thousands of calls to the dynamics model. A computationally expensive model can therefore dramatically decrease the speed of action decisions, making it difficult to satisfy requirement (3).

Motivated by this challenge, we propose Squeeze-to-Plan (S2P), a method for compressing the high-dimensional feature spaces of foundation vision encoders into low-dimensional state representations enabling efficient planning. To achieve this, S2P learns an attention-pooling encoder over DINOv3 ViT-S's visual features, trained with a novel bisimulation information bottleneck loss inspired by both bisimulation (Ferns et al., 2012; Gelada et al., 2019; Zhang et al., 2021) and Deep Variational Information Bottlenecks (Alemi et al., 2017). Our loss encourages S2P's compression to extract only information which is predictive of the environment dynamics and rewards (Figure 1). This design is inspired by a line of work on learning compressed representations of state spaces for Markov Decision Processes (MDPs) (Zhang et al., 2021; Gelada et al., 2019; Toso et al., 2026).

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

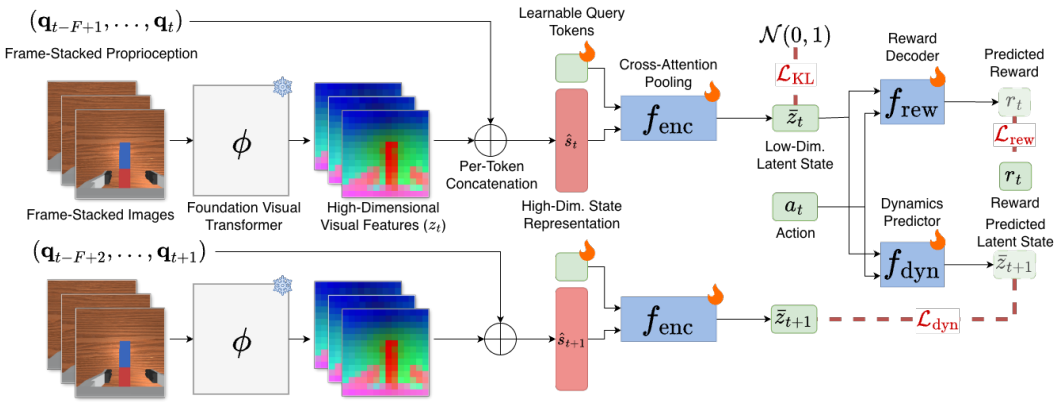

Figure 1: **S2P Architecture** On each training step, we sample a minibatch of transitions, apply the DINOv3 foundation model ϕ over the image observations, and concatenate each patch feature with the proprioceptive information to derive our high-dimensional state representations $\hat{s}_t$ and $\hat{s}_{t+1}$. A cross-attention pooling encoder transforms $\hat{s}_t, \hat{s}_{t+1}$ into low-dimensional state representations $\bar{z}_t, \bar{z}_{t+1}$, which are used to compute the three components of our Bisimulation Information Bottleneck loss (equation 5.)

Since S2P represents the environment state as a single compact vector, our dynamics model can be implemented as a simple MLP. This allows S2P to plan between 1 and 2 orders of magnitude faster than prior methods which apply pretrained vision encoders for planning, while improving success on three simulated real-time (RT) manipulation tasks based on the ManiSkill3 (Tao et al., 2025) task suite. On these tasks, we evaluate the time and memory complexity of planning with S2P and baselines, and the resulting planning performance of each method with and without applying test-time lighting and colour perturbations.

We summarize our contributions below:

1. We introduce a new framework for incorporating pretrained foundation visual encoders into world models, compressing their feature spaces into task-relevant representations to enable efficient planning.
2. We demonstrate that S2P's compact representations allow us to plan over an order of magnitude faster than baseline methods using pretrained encoders.
3. Separately from our method, we introduce an adaptation of prior works using the DINO family of visual models for planning, allowing them to perform longer-horizon manipulation tasks. We then demonstrate that Squeeze-to-Plan matches or exceeds the performance of these augmented baselines in simulated RT planning tasks.
4. We demonstrate that Squeeze-to-Plan's representation retains the robustness of its pretrained encoder's visual features to several test-time visual variations.

2 Related Work

2.1 Planning with Latent World Models

Action-conditioned world models (ACWMs) predict how an environment evolves based on actions taken by an agent, typically by encoding sensor observations to some latent representation of the environment state, and learning a dynamics model to predict the next state representation given actions (Hafner et al., 2025; Hansen et al., 2024; Micheli et al., 2023). ACWMs can be used as simulators with Model-Predictive Control (Schwenzer et al., 2021), to implement closed-loop robot controllers (**?**Schrittwieser et al., 2020; Hafner et al., 2019; Bhardwaj et al., 2020). Given candidate action plans, the dynamics model can simulate the resulting latent-state trajectories. An iterative optimization algorithm such as Model Predictive Path Integral control (MPPI) (Williams et al., 2015)

or the Cross-Entropy Method (CEM) (Botev et al., 2013) can refine the action plan to optimize a learned or heuristic planning objective. One benefit of applying ACWMs for MPC is that allocating more computation to the optimization often improves plan quality (Muppidi et al., 2026). For MPPI and CEM, this is realized by increasing the horizon length of the plan, or increasing the number of optimization iterations. Many approaches additionally learn a value function to consider rewards beyond the planning horizon (Hansen et al., 2024; Bhardwaj et al., 2020).

While MPC-style control with ACWMs can be a modular approach to learning robot controllers, it requires the ability to quickly simulate action sequences in latent space. When the dynamics and/or reward models are computationally expensive, planning becomes slow, negatively impacting task performance (Muppidi et al., 2026).

2.2 Planning with Foundation Vision Models

Recent efforts have integrated foundation visual models into world models, to benefit from their general feature representations (Bruce et al., 2024; Agarwal et al., 2025; Assran et al., 2025). Foundation world models like Cosmos3 (Agarwal et al., 2025) and Genie (Bruce et al., 2024) use encoders trained on internet-scale datasets, to extract state representations that generalize to unseen environments. Similarly, DINO-WM (Zhou et al., 2025) uses the DINOv2 visual foundation model (Oquab et al., 2024) directly as a latent state space to model the dynamics of a particular task. While the features of large, pretrained vision models can enable detailed and generalized perception, it is computationally expensive to use their high-dimensional features as latent states for planning.

2.3 Improving the Efficiency of DINO-WM

Several works have aimed to reduce this computational complexity, particularly in the case of DINO-WM. Sparse Imagination (Chun et al., 2026) takes the simple yet effective approach of randomly dropping DINO patch features when forming the state representation. In contrast, TC-WM (Fu et al., 2026) and DINO-Bisim (Toso et al., 2026) both reduce dimensionality by compressing DINO-WM's latent states with a learned, per-token compression. TC-WM trains this compression to retain predictive information for future compressed states, and the proprioceptive state of the robot. DINO-Bisim instead maps the patch tokens into a space where the L2 distance between tokens approximates the reward-free π-bisimulation metric (Zhang et al., 2021).

These prior methods achieve dimensionality reduction by either dropping information indiscriminately, or applying compression on a per-token basis. These choices prevent prior methods from compressing the latent state dimensionality to a single vector, and necessitate using expensive dynamics models like vision transformers (Dosovitskiy et al., 2021). This motivates developing a method which can compress the patch features of DINOv3 to a compact state vector where dynamics can be modeled efficiently and accurately.

3 Preliminaries

3.1 Notation and Assumptions

Let $M = (S, O, A, T, R, \gamma)$ represent a Partially Observable Markov Decision process (POMDP), where $s \in S$ are states, $o \in O$ are observations, $a \in A$ are actions and γ the discount factor. $T : S \times A \to \Delta(S)$ is the probabilistic state transition function, and $R : S \times A \to \mathbb{R}$ is the reward function. We denote with subscript t the value of a quantity in the t'th transition, i.e. $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$, and $r_t = r(s_t, a_t)$.

We assume that observations are $o_t = (i_t, q_t)$, where $i_t \in I$ is a visual input and q_t are proprioceptive measurements. We also assume access to a pretrained foundation visual encoder $\phi : I \to Z$, where $z \in Z$ is a high-dimensional feature space, such that $z_t = \phi(i_t)$ extracts useful geometric and semantic information from i_t. This enables us to assume that stacking together the last F observations of z_t provides a sufficient representation $\hat{s}_t$ of the current MDP state (i.e. $\hat{s}_t = (z_{t-F+1}, q_{t-F+1}, \ldots, z_t, q_t)$). We note that for most choices of foundation visual encoders, z_t, and therefore $\hat{s}_t$, can be very high dimensional. As our experiments show, planners using lightly-compressed versions of $\hat{s}_t$ are extremely slow and perform poorly at tasks that reward execution

speed. This motivates the search for strategies to substantially compress $\hat{s}_t$ while retaining useful perceptual information in z_t. One way to view the compression desired is as an *aliasing of states that contain the same useful information about rewards and transitions yet may be otherwise different*, a concept captured by *bisimulation relations*.

3.2 Bisimulation Relations and Metrics

A stochastic bisimulation relation $\sim$ is an equivalence relation on the state space S such that

$$s \sim s' \implies \forall a \in A,\ r(s,a) = r(s',a) \text{and } \forall \bar{s} \in S,\ \sum_{c \in [\bar{s}]_\sim} p(c \mid s,a) = \sum_{c \in [\bar{s}]_\sim} p(c \mid s',a), \quad (1)$$

where $[\bar{s}]_\sim = \{c \in S : c \sim \bar{s}\}$ denotes the equivalence class containing $\bar{s}$. Following Ferns et al. (2012), two states are bisimulation equivalent if they are related by some stochastic bisimulation relation. Such states yield the same rewards and have the same probability of transitioning into each equivalence class under every action. In general, it is difficult to find (s, s') for which equation 1 holds. Bisimulation metrics were introduced as a relaxation of equation 1, by quantifying how far two states are from being bisimulation-equivalent (Ferns et al., 2012). One formulation of a bisimulation metric is the π-bisimulation metric $d_\pi(s, s')$ defined as follows (Zhang et al., 2021):

$$d_\pi(s,s') = (1-c)\,|r(s,\pi(s)) - r(s',\pi(s'))| + c\ D(T(s,\pi(s)), T(s',\pi(s'))), \quad (2)$$

where π is some policy, $c \in [0, 1)$ controls the relative contribution of the two terms, and D represents some measure of the difference between the transition distributions, often $W_1(d_\pi)$, the 1-Wasserstein distance under the metric itself. Given an optimal policy π^*, we can derive a compression of S by aggregating states s and s' wherever $d_{\pi^*}(s, s') < \epsilon$. If we identify s by the cluster $\bar{z}$ that it is aggregated into, then we derive a new state space $\bar{Z}$, with associated optimal value $\bar{V}^*$, whose error is bounded by the coarseness ϵ of the aggregation (Zhang et al., 2021):

$$|V^*(s) - \bar{V}^*(\bar{z})| \leq {}^{2\epsilon}/_{[(1-\gamma)(1-c)]}. \quad (3)$$

Equation 2 and equation 3 indicate that if we can learn a compression function $\psi : S \to \bar{Z}$ which condenses states that are nearly bisimulation-equivalent under the visitation distribution of an optimal policy, then the optimal value function will be approximately preserved in the compressed space. We next discuss a framework we will use to formulate such a compression. Deep Variational Information Bottlenecks (DVIBs) Alemi et al. (2017) are a technique for learning a compression from a random variable X to a latent variable Z, such that Z is informative for reconstructing some output variable Y. Mathematically, DVIBs are a paired stochastic encoder $p_{\text{enc}} : X \times Z \to [0, 1]$ and decoder $p_{\text{dec}} : Z \times Y \to [0, 1]$ which maximize the following objective, where $I(\cdot;\cdot)$ is the mutual information of two random variables: $\mathcal{J}_{\text{DVIB}} \doteq I(Z;Y) - \beta I(Z;X)$. Optimizing this objective encourages z_i to become a minimal sufficient statistic of x_i, for predicting y_i (Alemi et al., 2017).

4 Squeeze-to-Plan: Pruning irrelevant information from a foundational encoder

We now outline Squeeze-to-Plan (S2P), our proposed framework for compressing the feature spaces of pretrained visual encoders. Our compression aims to reduce the dimensionality of the feature space to a single small vector, while retaining in it the information needed to model rewards and dynamics for planning (*Contribution 1*).

4.1 Bisimulation-Bottleneck State Compression

Motivated by bisimulation relations, our method uses DVIBs (Alemi et al., 2017) to learn an encoder to align state representations to be predictive of rewards and next representations, and otherwise compress information from the input. In particular, we construct a DVIB in which the input variable X is an observation sequence $\hat{s}_t = (o_{t-F+1}, o_{t-F+2}, \ldots, o_t)$, and the output variable Y is *the reward and next compressed representation*, $(R_{t+1}, \bar{Z}_{t+1})$:

$$\mathcal{J}_{\text{S2P}} \doteq I(\bar{Z}_t; (R_t, \bar{Z}_{t+1}) \mid A_t) - \beta I(\bar{Z}_t; \hat{S}_t), \quad (4)$$

After deriving the lower bound for this DVIB objective (see section A.1), adding per-term weighting coefficients $\lambda_{\text{rew}}, \lambda_{\text{dyn}}, \lambda_{H_{\text{enc}}}$, and writing $x_t \doteq (\hat{s}_t, a_t, r_t, \hat{s}_{t+1})$, we arrive at the surrogate objective $\mathcal{L}_{\text{S2P}}$ which we seek to maximize,

$$\mathcal{L}_{\text{S2P}} \doteq \mathop{\mathbb{E}}_{\substack{x_t \sim \mathcal{D} \\ z_t \sim p_{\text{enc}}(z_t \mid s_t) \\ z_{t+1} \sim p_{\text{enc}}(z_{t+1} \mid s_{t+1})}} [\underbrace{\lambda_{\text{rew}} \log p_{\text{rew}}(r_t \mid \bar{z}_t, a_t)}_{\mathcal{L}_{\text{rew}}} \tag{5}$$
$$\underbrace{+\lambda_{\text{dyn}} \log p_{\text{dyn}}(\bar{z}_{t+1} \mid \bar{z}_t, a_t)}_{\mathcal{L}_{\text{dyn}}} \underbrace{-\beta \,\text{KL}\left(p_{\text{encoder}}(\cdot \mid \hat{s}_t) \,\|\, \mathcal{N}(0,1)\right)}_{\mathcal{L}_{\text{KL}}} \underbrace{+\lambda_{H_{\text{enc}}} H[p_{\text{enc}}(\cdot \mid \hat{s}_t)]}_{\mathcal{L}_{H_{\text{enc}}}}].$$

We learn each of p_{enc}, p_{dyn}, and p_{rew} using gradient ascent. Empirically, we find that optimization is more stable when parameterizing p_{enc}, p_{dyn}, and p_{rew} as Gaussian distributions with fixed, small covariance. In this regime, p_{enc}, p_{dyn}, and p_{rew} become deterministic functions f_{enc}, f_{dyn}, f_{rew}. $\mathcal{L}_{\text{rew}}$, $\mathcal{L}_{\text{dyn}}$ and $\mathcal{L}_{\text{KL}}$ reduce to L2 losses, and $\mathcal{L}_{\text{H}_{\text{encoder}}}$ reduces to a constant (Appendix A.1):

$$\mathcal{L}^{\text{deterministic}}_{\text{S2P}} \doteq \mathop{\mathbb{E}}_{\substack{x_t \sim \mathcal{D} \\ z_t = f_{\text{enc}}(s_t) \\ z_{t+1} = f_{\text{enc}}(s_{t+1})}} [\underbrace{-\lambda_{\text{rew}} |r_t - f_{\text{rew}}(\bar{z}_t, a_t)|^2}_{\mathcal{L}^{\text{deterministic}}_{\text{rew}}} \underbrace{-\lambda_{\text{dyn}} |\bar{z}_{t+1} - f_{\text{dyn}}(\bar{z}_t, a_t)|^2}_{\mathcal{L}^{\text{deterministic}}_{\text{dyn}}} \underbrace{-\beta |f_{\text{enc}}(\hat{s}_t)|^2}_{\mathcal{L}^{\text{deterministic}}_{\text{KL}}}]. \tag{6}$$

Intuitively, the losses in equation 5 and equation 6 act as a bottleneck to compress our high-dimensional state representation $\hat{s}_t$ to a minimal statistic for predicting the compressed $\bar{Z}$-space dynamics and rewards. This has the effect of aliasing states together if they result in the same rewards and transitions, to achieve a bisimulation-inspired compression.

The minimal sufficient statistic for $(r_t, \bar{z}_{t+1})$ depends on the data distribution, and therefore the policy. Since we are interested in learning a bisimulation-based compression over the state distribution of the optimal policy, we frame our method as an *online* learning algorithm. This way, we ensure that the compression is trained over a data distribution which approaches an optimal policy.

Since we would like to apply p_{dyn} and p_{rew} for planning, we also propose learning a value function $V : \bar{Z} \to \mathbb{R}$ using an off-the-shelf value learning method. Alg. 1 illustrates our training procedure, agnostic to the particular instantiations of each model.

4.2 Instantiating S2P with DINOv3

We implement S2P with a frozen DINOv3 (Siméoni et al., 2025) backbone as the pretrained visual encoder ϕ. Following the example of DINO-WM (Zhou et al., 2025), we form $\hat{s}_t$ by concatenating proprioceptive information to each DINOv3 patch token, and stacking the past $F = 3$ observations.

We instantiate f_{enc} as a multi-head cross-attention pooling transformer, with a small set of learnable query tokens which attend to the frame-stacked and concatenated tokens in $\hat{s}_t$. After cross-attention, the transformed query tokens are concatenated and projected to the target latent dimensionality. In this compressed latent space, we learn f_{dyn} and f_{rew} as simple MLPs. Finally, we train $V(\bar{z})$ using TD-MPC2 (Hansen et al., 2025). Our full training architecture is illustrated in Figure 1.

5 Experiments

Our experiments aim to answer the following questions:

- **RQ1:** How much faster can we perform MPPI planning with Squeeze-to-Plan, compared with other methods which plan over DINO-based state representations? (*Contribution 2*)
- **RQ2:** How does Squeeze-to-Plan's efficient planning translate to task performance, in an evaluation environment where latency matters? (*Contribution 3*)
- **RQ3:** Does S2P's compression preserve the robustness of its pretrained features to test-time visual perturbations? (*Contribution 4*)
- **RQ4:** How does each term of the bisimulation-bottleneck loss, and the use of a pretrained backbone, contribute to S2P's performance?

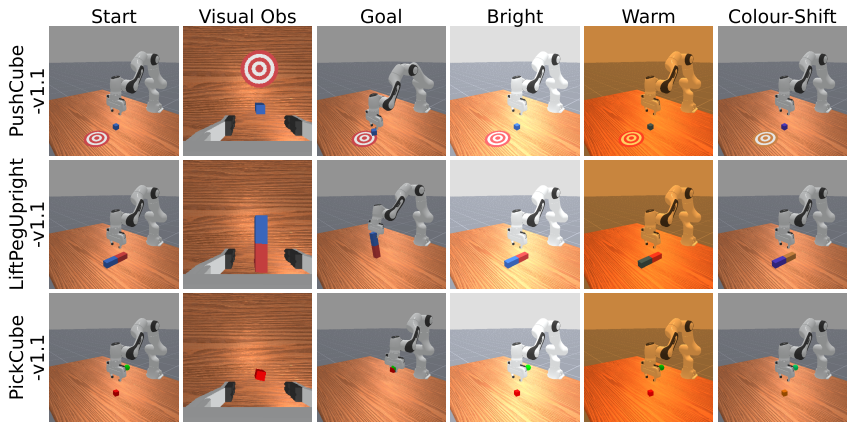

Figure 2: **Our Tasks** Our task suite consists of three custom tasks: PushCube-v1.1, LiftPegUpright-v1.1, and PickCube-v1.1. At test time, we apply real-time latency effects and refer to these settings as the *-RT variants of each task. We can apply lighting and colour perturbations to increase brightness, shift the lighting temperature, or alter object colours.

5.1 Baselines

For our baselines, we compare against the four prior approaches to using the DINOv3 encoder for planning, discussed in Section 2.3 – DINO-WM, Sparse Imagination, TC-WM, and DINO-Bisim. For fair comparison, we implement S2P and all baselines using DINOv3.

5.2 Real-Time Planning Tasks with Visual Perturbations

We train our method and baselines on three modified tasks from ManiSkill3 (Tao et al., 2025), which we refer to as PushCube-v1.1, LiftPegUpright-v1.1, and PickCube-v1.1 (section A.2). Figure 2 depicts the three environments.

Since we are interested in analyzing RT manipulation performance with different planning budgets, we train all methods in the standard *-v1.1 environments, and evaluate in environment variants where we model RT latency effects, called PushCube-v1.1-RT, LiftPegUpright-v1.1-RT, and PickCube-v1.1-RT. To model an RT planning scenario, we benchmark the time required to generate a plan with each method, under a variety of MPPI planner configurations (see Appendix A.6). We divide this interval by the environment's control interval of 0.05s to obtain the number of environment steps that the agent requires to plan. Every time the agent generates a plan, we first step the environment with no action for this many steps, before executing the actions of the plan.

We are also interested in understanding how S2P's frozen foundation encoder may provide robustness against test-time visual variations. To test this, we additionally evaluate each method in *-Bright, *-Warm, and *-ColourShift variants of each RT environment, where we apply different lighting and object-colour perturbations, as shown in Figure 2 and described in Appendix A.2.

5.3 Training Protocols

We train all methods with minimal modifications, for 1M gradient descent steps or 72 hours, whichever comes first. Due to the differences in the scale of computation required for training S2P versus the prior DINO-based methods, separate training protocols were employed. For S2P, we train online, as discussed in Section 4.1. Because DINO-WM, DINO-Bisim, TC-WM, and Sparse Imagination can take orders of magnitude longer to generate plans compared to S2P (as we show in Section 5.4.1), we find it infeasible to train these methods online. We instead train them using an offline dataset containing a mixture of expert and exploratory actions (Appendices A.3 and A.4).

In their original formulations, DINO-WM, DINO-Bisim, TC-WM, and Sparse Imagination are applied to short-horizon reward-free planning, using the ℓ_2 distance between the current and goal latent states as a planning objective. For our tasks we find this to be insufficient, as the ℓ_2 distance in

Table 1: **Planner Configurations.** We benchmark the time and memory complexity for a wide range of MPPI planner configurations, by grid-sweeping across parameters.

Planning Parameter	Sweep Values
Horizon (H)	$\{1, 2, 3, 4, 5\}$
Iterations (I)	$\{1, 2, 3, 4, 5\}$
Num. Trajectories (N), Num. Elites (E)	$\{(64, 8), (128, 16), (256, 32)\}$
Replanning Interval (K)	$\{1, \ldots, H\}$

DINO-based latent space can be dominated by visual similarity between the current and goal visual observations, which rarely leads the agent to interact with the task's object as intended.

To achieve non-zero performance, we train decoders $r(\hat{s}_t, a_t)$ over the latent spaces of our baseline methods, providing a task-targeted objective during planning. We implement this decoder with a cross-attention pooling architecture matching S2P's bisimulation compression model. This augmentation allows three of these methods to achieve non-zero success rates on evaluation in PushCube-v1.1-RT, and two in PickCube-v1.1-RT. Furthermore, we set the *frameskip* parameter of these methods to 1, allowing us to compare their planning performance at a variety of planning horizons.

5.4 Results

5.4.1 Planning Efficiency

For each method on each task, we benchmark the time and memory complexity of generating one plan with MPPI under different planner configurations. We sweep the MPPI planning hyperparameters across the ranges listed in Table 1. For each combination of hyperparameters, we evaluate the wall-clock time and peak GPU memory required to generate one plan (Appendix A.6).

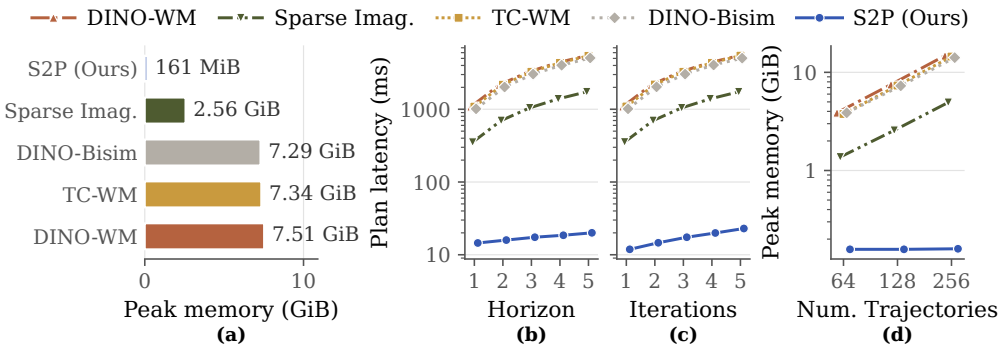

Figure 3: **Time and Memory Cost of Planning** Using a standard planner configuration (H=3, I=3, N=128, E=16, K=1), the methods have vastly different memory footprints when planning. Furthermore, the time required to plan increases as we increase horizon (b), MPPI iterations (c), and the number of sampled trajectories (d). We note that K has no impact on time or memory required to generate one plan and H and I have no impact on peak memory utilization. S2P plans in an efficient latent space, and is therefore able to plan faster and with smaller memory budgets for a given planning configuration.

Figure 3 shows the memory complexity of planning with each method, and how time complexity scales with H, I, and N. Since S2P plans in a low-dimensional latent space with an efficient dynamics model, it can plan between 1 and 2 orders of magnitude faster than baselines, with similarly reduced memory footprint. Sparse Imagination achieves lower time and memory complexity than the other baselines, indicating that the number of tokens to represent the latent state has a stronger impact on computational complexity of planning than the dimensionality of individual tokens.

5.5 Real-Time Planning with Compressed Features

Next, we investigate whether S2P's *efficient* planning is also *effective*. To this end, we train all methods in the standard *-v1.1 environments, and evaluate with various planner configurations in the *-RT environments, giving each 10s of simulated time to solve the task. We use the planning latencies recorded in Section 5.4.1 to model the RT latency effects. We note that the RT evaluation setting is generally more difficult than the non-RT setting, particularly in the case of the grasp-and-lift tasks, LiftPegUpright-v1.1-RT and PickCube-v1.1-RT. To keep the number of evaluations feasible, we consider a subset of 54 configurations from the sweep listed in Table 1, where we restrict $H \in \{1, 3, 5\}$, $I \in \{1, 3, 5\}$, $R \in \{1, 3, 5\}$, and $R \leq H$.

As shown in Section 5.4.1, certain planning configurations with the baseline methods can take over 1s to generate a single plan. Given the 10s budget to solve the task, not all planner configurations can feasibly solve this task. To assess which configurations *are* feasible for each baseline, we find the minimum number of actions required to solve the task using a PPO-trained expert, and do not evaluate any planning configuration which cannot generate this many actions in 10s (Appendix A.7).

For each feasible planning configuration, we evaluate the success rate in the RT variants of each task, and plot this as a function of the planner configuration's wall-clock time budget for generating a plan. For each method, we also track the best success rate achieved within a given planning budget. The "Default" column in Figure 4 shows the success rate of S2P and baselines across a range of planner configurations, along with the best achieved success rate within a given time budget.

S2P achieves competitive or higher success rate than the baselines across all tasks, and can solve the grasp-and-lift tasks LiftPegUpright-v1.1-RT and PickCube-v1.1-RT while the baselines generally cannot.

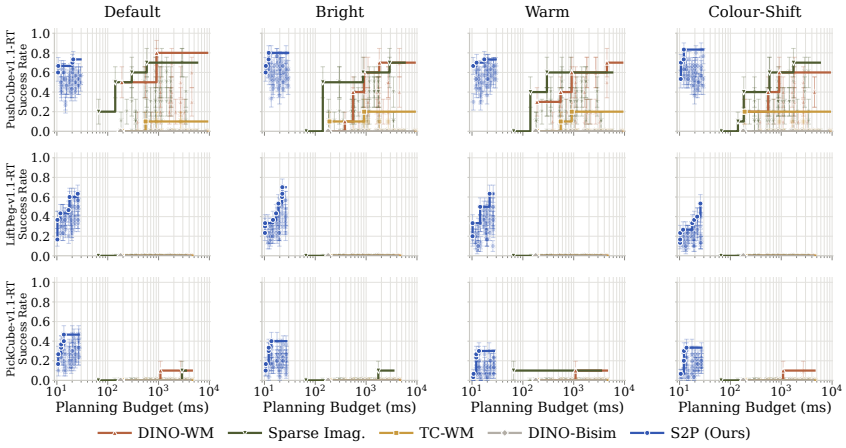

Figure 4: **Success rate vs planning budget** Success rates (mean ± SEM) for all methods as a function of planning budget, evaluated in the RT environments under each visual setting (columns). S2P plans faster, achieves higher success rates, and shows similar robustness to other DINO-based methods. Methods were trained under 1 seed each. Evaluation uses 10 episodes per planning configuration for baselines, 30 for S2P.

5.6 Preserving Perceptual Priors

It is conceivable that S2P's training procedure could destroy the useful visual prior offered by its foundation encoder model when forming its compressed latent space, which would make S2P more brittle against test-time visual variations. To test this, we perform the same evaluation described in Section 5.5 on the visually perturbed variants of our RT environments described in Section 5.2. Columns 2, 3, and 4 of Figure 4 show the performance of each method under these variations and Table 2 summarizes the success rate of all methods on each task, averaging across all visual variants. The results show that S2P performs consistently across the different visual variants of each task, matching the behaviour of the DINO-WM and Sparse Imagination baselines, which plan directly in

DINOv3's latent space. This indicates that S2P's training procedure does not destroy the robustness of the original DINOv3 features, on these specific visual perturbations.

5.7 Ablations

Finally, analyze the impact of turning off the gradients from each of our loss terms to S2P's bisimulation compression encoder. Additionally, we include a *Fresh Encoder* ablation, where we re-initialize DINOv3's pretrained weights and train it from scratch alongside S2P. We train each ablation in the LiftPegUpright-v1.1 environment, and evaluate the best-planner success rates in LiftPegUpright-v1.1-RT (Table 3), adding visual perturbations for the *Fresh Encoder* ablation (Table 4).

The results indicate that $\mathcal{L}_{\text{rew}}$ and $\mathcal{L}_{\text{KL}}$ play a role in encouraging S2P's compact latent space to retain relevant information for effective planning, while $\mathcal{L}_{\text{dyn}}$ has neutral effect on LiftPegUpright-v1.1-RT. Furthermore, while training an encoder from scratch yields similar performance to S2P (Full) in the task it was trained, on it shows very poor generalization across visual perturbations. This indicates that the pretrained DINOv3 weights may play a role in allowing S2P to remain robust across the visual perturbations tested here.

Method	Push Cube	Lift Peg	Pick Cube
DINO-WM	0.700 ± 0.072	0.000 ± 0.000	0.075 ± 0.041
Sparse Imag.	0.675 ± 0.074	0.000 ± 0.000	0.075 ± 0.041
TC-WM	0.175 ± 0.060	0.000 ± 0.000	0.000 ± 0.000
DINO-Bisim	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
S2P (Ours)	0.775 ± 0.038	0.625 ± 0.044	0.375 ± 0.044

Table 2: **Success Rate Summary** Success rate (mean $\pm$ SEM) for the best-performing planner configuration for each task and method, averaged over the 4 visual perturbation settings. Training uses 1 seed, evaluation uses 10 seeds per planner configuration for baselines, 30 for S2P.

Ablation	LiftPegUpright-v1.1-RT SR
S2P (Full)	0.63 ± 0.09
Fresh Encoder	0.87 ± 0.06
No Dynamics Grads.	0.67 ± 0.09
No KL Grads.	0.57 ± 0.09
No Reward Grads.	0.40 ± 0.09

Table 3: **Loss Ablations**: Success rate (mean $\pm$ SEM) for best-performing planner of each ablation upon training in LiftPegUpright-v1.1 and being evaluated in LiftPegUpright-v1.1-RT. Training uses 1 seed, evaluation uses 30 seeds per planner configuration.

Visual Perturbation	Fresh Enc. SR
Default	0.87 ± 0.06
Bright	0.00 ± 0.00
Warm	0.00 ± 0.00
ColourShift	0.00 ± 0.00

Table 4: **Fresh Encoder Ablation**: Success rate (mean $\pm$ SEM) for best-performing planner of the *Fresh Encoder* ablation, evaluated in LiftPegUpright-v1.1-RT under visual perturbations. Evaluation uses 30 seeds per planner configuration.

6 Conclusion

We have introduced Squeeze-to-Plan, a novel framework for building efficient, task-targeted world models around foundation visual encoders. While prior works model the environment dynamics directly in the encoder's feature space, S2P applies a novel loss function motivated by both bisimulation compressions and information bottlenecks, to learn a compression from the encoder's high-dimensional features to a compact state representation suitable for planning. In contrast with prior methods, S2P can be trained online, and can generate plans with orders of magnitude less time and memory, all while retaining perceptual priors of its pretrained encoder that allow it to be robust against three test-time visual perturbations. Going forward, we believe that S2P points to a possible framework to benefit from both the generality of large, pretrained foundation models and the efficiency of small, task-specific planners.

AI USE STATEMENT

In this work, we used generative AI tools to assist in the writing of proofs related to our loss derivation, implement methods, provide feedback on research methodology or experiments, draft sections of the paper, and interpret results. We have not used generative AI tools to formulate mathematical claims, provide critical ingredients for proving mathematical claims, propose or refine hypotheses, or cleaning and reformatting datasets. Assisting with translation, qualitative and thematic data analysis are not applicable to this work. Additionally, we used generative AI tools to suggest experimental parameters, create or edit software code, and summarize or analyze existing literature. We have reviewed all AI-assisted work. AI facilitated derivations have been double-checked, and AI-generated code was reviewed and tested. Literature review was human-conducted, with occasional AI assistance to identify gaps. All prior work recommended by AI was human-reviewed and assessed for relevance. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

We do not believe that our work raises any questions regarding the Code of Ethics.

REPRODUCIBILITY STATEMENT

We release our full code, containing all software used to run our training and evaluations, define environments, and run analyses. In addition, we include full details for our task implementations in Appendix A.2, and dataset generation methodology in Appendix A.3. Finally, we list the details for how we ran benchmarking for each of our planners in Appendix A.6.

REFERENCES

Niket Agarwal, Arslan Ali, Maciej Bala, Yogesh Balaji, Erik Barker, Tiffany Cai, Prithvijit Chattopadhyay, Yongxin Chen, Yin Cui, Yifan Ding, et al. Cosmos world foundation model platform for physical ai. *arXiv preprint arXiv:2501.03575*, 2025.

Alexander A. Alemi, Ian Fischer, Joshua V. Dillon, and Kevin Murphy. Deep variational information bottleneck. 2017. URL https://openreview.net/forum?id=HyxQzBceg.

Mido Assran, Adrien Bardes, David Fan, Quentin Garrido, Russell Howes, Matthew Muckley, Ammar Rizvi, Claire Roberts, Koustuv Sinha, Artem Zholus, et al. V-jepa 2: Self-supervised video models enable understanding, prediction and planning. *arXiv preprint arXiv:2506.09985*, 2025.

Mohak Bhardwaj, Ankur Handa, Dieter Fox, and Byron Boots. Information theoretic model predictive q-learning. In *Learning for Dynamics and Control*, pp. 840–850. PMLR, 2020.

Zdravko I. Botev, Dirk P. Kroese, Reuven Y. Rubinstein, and Pierre L'Ecuyer. Chapter 3 - The Cross-Entropy Method for Optimization. In C.R. Rao and Venu Govindaraju (eds.), *Handbook of Statistics*, volume 31, pp. 35–59. Elsevier, January 2013. ISBN 0169-7161. doi: 10.1016/B978-0-444-53859-8.00003-5.

Jake Bruce, Michael D Dennis, Ashley Edwards, Jack Parker-Holder, Yuge Shi, Edward Hughes, Matthew Lai, Aditi Mavalankar, Richie Steigerwald, Chris Apps, et al. Genie: Generative interactive environments. In *Forty-first International Conference on Machine Learning*, 2024.

Junha Chun, Youngjoon Jeong, and Taesup Kim. Sparse imagination for efficient visual world model planning. In *International Conference on Learning Representations*, volume 2026, pp. 102160–102188, 2026.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image Is Worth 16x16 Words: Transformers for Image Recognition at Scale. 2021. URL https://mlanthology.org/iclr/2021/dosovitskiy2021iclr-image/.

Norman Ferns, Prakash Panangaden, and Doina Precup. Metrics for finite markov decision processes. *arXiv preprint arXiv:1207.4114*, 2012.

Minghao Fu, Fan Feng, Nicklas Hansen, and Biwei Huang. Back to parsimonious latents: Learning task-centric world models from visual foundations. *arXiv preprint arXiv:2605.25620*, 2026.

Carles Gelada, Saurabh Kumar, Jacob Buckman, Ofir Nachum, and Marc G. Bellemare. DeepMDP: Learning continuous latent space models for representation learning, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/gelada19a.html.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International conference on machine learning*, pp. 2555–2565. PMLR, 2019.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse control tasks through world models. *Nature*, 640(8059):647–653, April 2025. ISSN 1476-4687. doi: 10.1038/s41586-025-08744-2.

Nicklas Hansen, Hao Su, and Xiaolong Wang. TD-MPC2: Scalable, robust world models for continuous control. 2024. URL https://openreview.net/forum?id=Oxh5CstDJU.

Nicklas Hansen, Hao Su, and Xiaolong Wang. Learning massively multitask world models for continuous control. *arXiv preprint arXiv:2511.19584*, 2025.

Moo Jin Kim, Yihuai Gao, Tsung-Yi Lin, Yen-Chen Lin, Yunhao Ge, Grace Lam, Percy Liang, Shuran Song, Ming-Yu Liu, Chelsea Finn, and Jinwei Gu. Cosmos policy: Fine-tuning video models for visuomotor control and planning. 2026. URL https://openreview.net/forum?id=wPEIStHxYH.

Vincent Micheli, Eloi Alonso, and François Fleuret. Transformers are sample-efficient world models. 2023. URL https://openreview.net/forum?id=vhFu1Acb0xb.

Aneesh Muppidi, Firas Darwish, Dylan Cope, João F Henriques, and Jakob Nicolaus Foerster. Finding the time to think: Learning planning budgets in real-time rl. *arXiv preprint arXiv:2606.26463*, 2026.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy V. Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Mido Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Herve Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. DINOv2: Learning Robust Visual Features Without Supervision. *Transactions on Machine Learning Research*, 2024. URL https://mlanthology.org/tmlr/2024/oquab2024tmlr-dinov2/.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

Max Schwenzer, Muzaffer Ay, Thomas Bergs, and Dirk Abel. Review on model predictive control: An engineering perspective. *The international journal of advanced manufacturing technology*, 117(5):1327–1349, 2021.

Oriane Siméoni, Huy V Vo, Maximilian Seitzer, Federico Baldassarre, Maxime Oquab, Cijo Jose, Vasil Khalidov, Marc Szafraniec, Seungeun Yi, Michaël Ramamonjisoa, et al. Dinov3. *arXiv preprint arXiv:2508.10104*, 2025.

Stone Tao, Fanbo Xiang, Arth Shukla, Yuzhe Qin, Xander Hinrichsen, Xiaodi Yuan, Chen Bao, Xinsong Lin, Yulin Liu, Tse-Kai Chan, Yuan Gao, Xuanlin Li, Tongzhou Mu, Nan Xiao, Arnav Gurha, Viswesh Nagaswamy Rajesh, Yong Woo Choi, Yen-Ru Chen, Zhiao Huang, Roberto Calandra, Rui Chen, Shan Luo, and Hao Su. Demonstrating GPU Parallelized Robot Simulation and Rendering for Generalizable Embodied AI with ManiSkill3. June 2025. doi: 10.15607/RSS.2025.XXI.021.

Leonardo F Toso, Davit Shadunts, Yunyang Lu, Nihal Sharma, Donglin Zhan, Nam H Nguyen, and James Anderson. Learning invariant visual representations for planning with joint-embedding predictive world models. *arXiv preprint arXiv:2602.18639*, 2026.

Grady Williams, Andrew Aldrich, and Evangelos Theodorou. Model predictive path integral control using covariance variable importance sampling. *arXiv preprint arXiv:1509.01149*, 2015.

Amy Zhang, Rowan Thomas McAllister, Roberto Calandra, Yarin Gal, and Sergey Levine. Learning Invariant Representations for Reinforcement Learning Without Reconstruction, 2021. URL `https://mlanthology.org/iclr/2021/zhang2021iclr-learning-d/`.

Gaoyue Zhou, Hengkai Pan, Yann Lecun, and Lerrel Pinto. DINO-WM: World Models on Pre-Trained Visual Features Enable Zero-Shot Planning. 267:79115–79135, 2025. URL `https://mlanthology.org/icml/2025/zhou2025icml-dinowm/`.

A Appendix

A.1 Loss Derivation

We begin with $\hat{S}_t$, the random variable corresponding to our high-dimensional, uncompressed state representation at time t. Motivated by bisimulation compressions, we wish to learn a mapping to a relatively low-dimensional latent random variable $\bar{Z}_t$, such that $\bar{Z}_t$ is predictive of the MDP's rewards R_t, and $\bar{Z}_{t+1}$, given the action A_t. Following Alemi et al. (2017), we formalize this objective using mutual information $I(\cdot;\cdot)$ as

$$\mathcal{J}_{\mathrm{S2P}} = I((R_t, \bar{Z}_{t+1}); \bar{Z}_t \mid A_t) - \beta I(\bar{Z}_t; \hat{S}_t), \quad \beta \geq 0. \tag{7}$$

We wish to learn a stochastic encoder $p_{\mathrm{enc},\theta} : \hat{S} \times \bar{Z} \to [0,1]$ with parameters θ to maximize this objective, over a data distribution $\mathcal{D}$ defining a distribution $p_{\mathcal{D}}(\hat{s}_t, a_t, r_t, \hat{s}_{t+1})$ over the random variable $(\hat{S}_t, A_t, R_t, \hat{S}_{t+1})$. The joint distribution over data and latents is then

$$p_\theta(\hat{s}_t, a_t, r_t, \hat{s}_{t+1}, \bar{z}_t, \bar{z}_{t+1}) = p_{\mathcal{D}}(\hat{s}_t, a_t, r_t, \hat{s}_{t+1})\, p_{\mathrm{enc},\theta}(\bar{z}_t|\hat{s}_t)\, p_{\mathrm{enc},\theta}(\bar{z}_{t+1}|\hat{s}_{t+1}).$$

In what follows, expectations are taken with respect to $p_\theta(\hat{s}_t, a_t, r_t, \hat{s}_{t+1}, \bar{z}_t, \bar{z}_{t+1})$, and we denote with subscripts θ any quantities which depend on the parameters θ.

A.1.1 Lower Bound for $I((R_t, \bar{Z}_{t+1}); \bar{Z}_t \mid A_t)$

We first rewrite the first term of equation 7 as

$$\begin{aligned} I_\theta((R_t, \bar{Z}_{t+1}); \bar{Z}_t \mid A_t) =& H_\theta(R_t, \bar{Z}_{t+1}|A_t) - H_\theta(R_t, \bar{Z}_{t+1}|\bar{Z}_t, A_t) \\ =& H(R_t|A_t) + H_\theta(\bar{Z}_{t+1}|R_t, A_t) - H_\theta(R_t, \bar{Z}_{t+1}|\bar{Z}_t, A_t) \\ =& H(R_t|A_t) + H_\theta(\bar{Z}_{t+1}|R_t, A_t) + \mathbb{E}_\theta[\log p_\theta(R_t, \bar{Z}_{t+1}|\bar{Z}_t, A_t)], \end{aligned}$$

and note that the conditional entropy $H(R_t|A_t)$ is a constant, depending purely on the data distribution $\mathcal{D}$. Focusing next on the $H_\theta(\bar{Z}_{t+1}|R_t, A_t)$ term, we note that the encoder $p_{\mathrm{enc},\theta}(\bar{Z}_{t+1} \mid \hat{S}_{t+1})$ depends only on $\hat{S}_{t+1}$, so:

$$\begin{aligned} H_\theta(\bar{Z}_{t+1} \mid R_t, A_t) &= H_\theta(\bar{Z}_{t+1} \mid \hat{S}_{t+1}, R_t, A_t) + \underbrace{I_\theta(\bar{Z}_{t+1}; \hat{S}_{t+1} \mid R_t, A_t)}_{\geq 0} \\ &\geq H_\theta(\bar{Z}_{t+1} \mid \hat{S}_{t+1}, R_t, A_t) \\ &= H_\theta(\bar{Z}_{t+1} \mid \hat{S}_{t+1}) \\ &= \mathbb{E}_{\hat{S}_{t+1}} H[p_{\mathrm{enc},\theta}(\cdot \mid \hat{S}_{t+1})]. \end{aligned}$$

We now turn to the remaining term $\mathbb{E}_\theta[\log p_\theta(R_t, \bar{Z}_{t+1} \mid \bar{Z}_t, A_t)]$ and approximate the conditional $p_\theta(R_t, \bar{Z}_{t+1} \mid \bar{Z}_t, A_t)$ with a factorized *variational decoder* with its own parameters η, chosen by design to factorize into a reward predictor and a dynamics predictor:

$$p_\eta(r_t, \bar{z}_{t+1} \mid \bar{z}_t, a_t) = p_{\mathrm{rew},\eta}(r_t \mid \bar{z}_t, a_t)\, p_{\mathrm{dyn},\eta}(\bar{z}_{t+1} \mid \bar{z}_t, a_t).$$

The KL divergence between the true conditional $p_\theta(\cdot \mid \bar{Z}_t, A_t)$ and the decoder $p_\eta(\cdot \mid \bar{Z}_t, A_t)$ must be non-negative, so:

$$\begin{aligned}\mathbb{E}_\theta[\log p_\theta(R_t, \bar{Z}_{t+1} \mid \bar{Z}_t, A_t)] &= \mathbb{E}_\theta[\log p_\eta(R_t, \bar{Z}_{t+1} \mid \bar{Z}_t, A_t)] \\ &\quad + \mathbb{E}_{\bar{Z}_t, A_t}\mathrm{KL}\big(p_\theta(\cdot \mid \bar{Z}_t, A_t) \,\|\, p_\eta(\cdot \mid \bar{Z}_t, A_t)\big) \\ &\geq \mathbb{E}_\theta[\log p_\eta(R_t, \bar{Z}_{t+1} \mid \bar{Z}_t, A_t)] \\ &= \mathbb{E}_\theta\big[\log p_{\mathrm{rew},\eta}(R_t \mid \bar{Z}_t, A_t) + \log p_{\mathrm{dyn},\eta}(\bar{Z}_{t+1} \mid \bar{Z}_t, A_t)\big].\end{aligned}$$

Combining the two bounds derived above, we obtain the following lower bound on the predictive term:

$$I_\theta((R_t, \bar{Z}_{t+1}); \bar{Z}_t \mid A_t) \geq H(R_t \mid A_t) + \mathbb{E}_{\hat{S}_{t+1}} H[p_{\mathrm{enc},\theta}(\cdot \mid \hat{S}_{t+1})] \tag{8}$$

$$+ \mathbb{E}_\theta\big[\log p_{\mathrm{rew},\eta}(R_t \mid \bar{Z}_t, A_t) + \log p_{\mathrm{dyn},\eta}(\bar{Z}_{t+1} \mid \bar{Z}_t, A_t)\big]. \tag{9}$$

A.1.2 Upper Bound for $I(\bar{Z}_t; \hat{S}_t)$

We now bound the compression term of equation 7. Let $\rho = \mathcal{N}(0, I)$ be a fixed prior over the latent space, and let $q_{\theta,t}(\bar{z}) := \mathbb{E}_{\hat{S}_t}\big[p_{\mathrm{enc},\theta}(\bar{z} \mid \hat{S}_t)\big]$ denote the aggregate encoder distribution at time t. Then

$$\begin{aligned}\mathbb{E}_{\hat{S}_t}\,\mathrm{KL}\big(p_{\mathrm{enc},\theta}(\cdot \mid \hat{S}_t) \,\|\, \rho\big) &= \mathbb{E}_{\hat{S}_t, \bar{Z}_t} \log \frac{p_{\mathrm{enc},\theta}(\bar{Z}_t \mid \hat{S}_t)}{\rho(\bar{Z}_t)} \\ &= \mathbb{E}_{\hat{S}_t, \bar{Z}_t} \log \frac{p_{\mathrm{enc},\theta}(\bar{Z}_t \mid \hat{S}_t)}{q_{\theta,t}(\bar{Z}_t)} + \mathbb{E}_{\bar{Z}_t} \log \frac{q_{\theta,t}(\bar{Z}_t)}{\rho(\bar{Z}_t)} \\ &= I_\theta(\bar{Z}_t; \hat{S}_t) + \mathrm{KL}(q_{\theta,t} \,\|\, \rho) \\ &\geq I_\theta(\bar{Z}_t; \hat{S}_t).\end{aligned}$$

Since $\beta \geq 0$, this upper bound on $I_\theta(\bar{Z}_t; \hat{S}_t)$ yields a lower bound once multiplied by $-\beta$:

$$-\beta I_\theta(\bar{Z}_t; \hat{S}_t) \geq -\beta\, \mathbb{E}_{\hat{S}_t}\,\mathrm{KL}\big(p_{\mathrm{enc},\theta}(\cdot \mid \hat{S}_t) \,\|\, \rho\big). \tag{10}$$

A.1.3 Combining the Bounds

Substituting equation 9 and equation 10 into equation 7 gives

$$\mathcal{J}_{\mathrm{S2P}} \geq H(R_t \mid A_t) + \mathcal{L}, \tag{11}$$

where

$$\begin{aligned}\mathcal{L} := \mathbb{E}_\theta\Big[&\log p_{\mathrm{rew},\eta}(R_t \mid \bar{Z}_t, A_t) + \log p_{\mathrm{dyn},\eta}(\bar{Z}_{t+1} \mid \bar{Z}_t, A_t) \\ &- \beta\,\mathrm{KL}\big(p_{\mathrm{enc},\theta}(\cdot \mid \hat{S}_t) \,\|\, \rho\big) + H[p_{\mathrm{enc},\theta}(\cdot \mid \hat{S}_{t+1})]\Big].\end{aligned} \tag{12}$$

As $H(R_t \mid A_t)$ is a constant independent of θ and η, maximizing $\mathcal{L}$ maximizes a valid lower bound on $\mathcal{J}_{\mathrm{S2P}}$.

A.1.4 Deterministic Variant

We now specialize equation 12 to the case where the encoder and both decoders are Gaussian with fixed covariance:

$$p_{\mathrm{enc},\theta}(\cdot \mid \hat{s}) = \mathcal{N}\big(f_{\mathrm{enc}}(\hat{s}),\ \sigma^2_{\mathrm{enc}} I\big), \qquad p_{\mathrm{rew},\eta}(\cdot \mid \bar{z}, a) = \mathcal{N}\big(f_{\mathrm{rew}}(\bar{z}, a),\ \sigma^2_{\mathrm{rew}} I\big),$$

$$p_{\mathrm{dyn},\eta}(\cdot \mid \bar{z}, a) = \mathcal{N}\big(f_{\mathrm{dyn}}(\bar{z}, a),\ \sigma^2_{\mathrm{dyn}} I\big),$$

where $\sigma^2_{\mathrm{enc}}, \sigma^2_{\mathrm{rew}}, \sigma^2_{\mathrm{dyn}}$ are constants. Specializing $\log p_{\mathrm{rew},\eta}(R_t \mid \bar{Z}_t, A_t)$, $\log p_{\mathrm{dyn},\eta}(\bar{Z}_{t+1} \mid \bar{Z}_t, A_t)$, and $\mathrm{KL}\big(p_{\mathrm{enc},\theta}(\cdot \mid \hat{S}_t) \,\|\, \rho\big)$ to this regime gives

$$\begin{aligned}\log p_{\mathrm{rew},\eta}(R_t \mid \bar{Z}_t, A_t) &= -\frac{1}{2\sigma^2_{\mathrm{rew}}}\big|R_t - f_{\mathrm{rew}}(\bar{Z}_t, A_t)\big|^2 - \underbrace{\frac{1}{2}\log(2\pi\sigma^2_{\mathrm{rew}})}_{=:c_{\mathrm{rew}}} \\ &= -\lambda_{\mathrm{rew}}\big|R_t - f_{\mathrm{rew}}(\bar{Z}_t, A_t)\big|^2 - c_{\mathrm{rew}},\end{aligned}$$

$$\log p_{\mathrm{dyn},\eta}(\bar{Z}_{t+1} \mid \bar{Z}_t, A_t) = -\frac{1}{2\sigma_{\mathrm{dyn}}^2}\big|\bar{Z}_{t+1} - f_{\mathrm{dyn}}(\bar{Z}_t, A_t)\big|^2 - \underbrace{\frac{d}{2}\log(2\pi\sigma_{\mathrm{dyn}}^2)}_{=:c_{\mathrm{dyn}}}$$

$$=: -\lambda_{\mathrm{dyn}}\big|\bar{Z}_{t+1} - f_{\mathrm{dyn}}(\bar{Z}_t, A_t)\big|^2 - c_{\mathrm{dyn}},$$

$$\mathrm{KL}\big(p_{\mathrm{enc},\theta}(\cdot \mid \hat{S}_t) \,\|\, \rho\big) = \frac{1}{2}\big|f_{\mathrm{enc}}(\hat{S}_t)\big|^2 + \underbrace{\frac{d}{2}\big(\sigma_{\mathrm{enc}}^2 - 1 - \log\sigma_{\mathrm{enc}}^2\big)}_{=:c_{\mathrm{KL}}},$$

where d the dimension of $\bar{Z}$, $\lambda_{\mathrm{rew}} := \frac{1}{2\sigma_{\mathrm{rew}}^2}$, $\lambda_{\mathrm{dyn}} := \frac{1}{2\sigma_{\mathrm{dyn}}^2}$, and $c_{\mathrm{rew}}, c_{\mathrm{dyn}}, c_{\mathrm{KL}}$ are constants.

Finally, since σ_{enc}^2 does not depend on the input, $\mathcal{L}_{\mathrm{H_{enc}}}$ becomes a constant:

$$H\big[p_{\mathrm{enc},\theta}(\cdot \mid \hat{S}_{t+1})\big] = \frac{d}{2}\log(2\pi e\,\sigma_{\mathrm{enc}}^2) =: c_H.$$

Plugging these four reductions into equation 12:

$$\mathcal{L} = \mathbb{E}_\theta\Big[-\lambda_{\mathrm{rew}}\big|R_t - f_{\mathrm{rew}}(\bar{Z}_t, A_t)\big|^2 - \lambda_{\mathrm{dyn}}\big|\bar{Z}_{t+1} - f_{\mathrm{dyn}}(\bar{Z}_t, A_t)\big|^2 - \frac{\beta}{2}\big|f_{\mathrm{enc}}(\hat{S}_t)\big|^2\Big] + \underbrace{\big(-c_{\mathrm{rew}} - c_{\mathrm{dyn}} - \beta c_{\mathrm{KL}} + c_H\big)}_{\text{independent of } \theta,\eta}.$$

A.2 Environments

Our three environments PushCube-v1.1, LiftPegUpright-v1.1, and PickCube-v1.1 are based on the ManiSkill3 (Tao et al., 2025) table-top manipulation tasks PushCube-v1, LiftPegUpright-v1, and PickCube-v1:

- **PickCube-v1**: The robot must grasp a 4 cm cube and carry it to a goal position in free space. The cube starts with a random position and a random rotation about the vertical axis. The goal is drawn independently: its horizontal position comes from the same region as the cube, and its height is up to 30 cm above the table. The episode succeeds when the cube is within 2.5 cm of the goal and the robot has come to rest. The reward encourages the agent to reach the cube, grasp it, carry it to the goal, and then hold still.
- **PushCube-v1**: The robot must push a 4 cm cube across the table into a circular target region, 10 cm in radius, whose centre sits 20 cm in front of the cube's starting position. The episode succeeds when the cube's centre lies within the target region and the cube is still resting on the table. The reward encourages the agent to move the gripper behind the cube, then push the cube towards the target while keeping it on the table.
- **LiftPegUpright-v1**: A peg 24 cm long with a 5 cm square cross-section lies on its side at a random position on the table, and the robot must stand it upright on its end. The episode succeeds when the peg's long axis is within 0.08 rad of vertical and its base rests on the table, with its centre within 5 mm of half the peg's length above the surface. The reward encourages the agent to reach the peg, rotate it towards vertical, and raise its centre to the upright height.

In all tasks, the actions are 4-dimensional and continuous – 3 dimensions control the end effector position as a delta-position command in task space, and the fourth controls the gripper position.

We apply the following modifications to the stock implementations of these tasks, to create our *-v1.1 variants:

- **No early termination:** The stock tasks end an episode as soon as the success condition is first met. Our variants never terminate early, so every episode runs the full 50-step horizon.
- **Wrist-Camera Observations:** The stock tasks observe the scene through a fixed third-person camera at 128×128. We replace it with a single wide-angle (108° field of view) RGB camera mounted on the gripper, rendered at 224×224.

Finally, our *-Bright, *-Warm, and *-ColourShift variants of the tasks apply the following changes to the *-v1.1 versions:

1. **Bright**: The scene is lit 2.5× more intensely. Ambient light is raised from 0.3 to 0.75, and both directional lights from intensity 1.0 to 2.5, with their directions and colour unchanged.
2. **Warm**: Every light is tinted from the stock neutral white (6500 K) to a warm colour temperature of about 2550 K. Luminance is held constant, so only the colour of the light changes, not its overall brightness.
3. **ColourShift**: The colours of all task objects (the cube or peg, and PushCube's target region) are rotated 30° around the hue wheel, turning red towards orange and blue towards violet. The robot, table and floor keep their original appearance, and the lighting is unchanged.

For all tasks, observations consist of the 224px×224px wrist camera view, plus proprioceptive information comprising the joint angles, joint velocities, the position and velocities of the two gripper fingers, and the position and quaternion defining the end-effector pose. For PickCube-v1.1, we additionally provide the 3-D target position of the cube as proprioceptive information.

A.3 Datasets

For each task we collect an offline dataset of 30,000 50-step episodes, spanning a wide range of behaviour quality, from random to expert. We train a PPO agent on the dense reward with privileged state observations, using 1,024 parallel environments. We record three stages of 10,000 episodes, each under a different policy:

- **Random**: Actions sampled uniformly from the action space.
- **Mid-training**: During training, we periodically record 500 evaluation episodes with the current policy. Training stops once the policy's success rate across 500 evaluation episodes reaches a task-specific threshold (95% for PushCube-v1.1 and PickCube-v1.1, 90% for LiftPegUpright-v1.1). Any of this stage's trajectory budget not yet used at that point, is recorded with the converged policy.
- **Expert**: Actions taken by the converged policy.

Table 5 reports the success rate of each stage.

A.4 Training Compute

All methods are trained until they reach either 1 million gradient steps or 72 hours of wall-clock time, whichever comes first. S2P is trained online and reaches the 1 million step limit in under 24 hours on a single consumer-grade GPU (Nvidia GeForce RTX 4090). The DINO-based baselines are trained offline on the 30,000-episode PPO-generated datasets described in Appendix A.3, using Nvidia H100 Tensor Core GPUs. Each baseline trains for the full 72 hours, completing between 15 and 32 epochs over its dataset across the three tasks.

Table 5: Fraction of episodes in each stage of the PPO-staged datasets that reach success.

Task	Random	Mid-training	Expert
PushCube	0.6%	45.2%	97.5%
PickCube	0.0%	88.9%	97.3%
LiftPegUpright	0.0%	78.6%	90.4%

A.5 Squeeze-to-Plan Hyperparameters

All three tasks use the same hyperparameters (Table 6).

Algorithm 1 Squeeze-to-Plan (S2P) Training Procedure

Require: Empty replay buffer $\mathcal{D}$, frozen visual encoder ϕ, MPPI planner, value-learning method computing value_loss, training environment env with frame-stacked observations, learning rate α, loss weights $\lambda_{\text{rew}}, \beta, \lambda_{\text{dyn}}, \lambda_{H_{\text{enc}}}$

Initialize parameters θ of p_{enc} and η of $p_{\text{dyn}}, p_{\text{rew}}$; initialize parameters ν of value function V
function ENCODE(i, q)
  return concatenate($\phi(i), q$)
end function
function ACT_AND_STORE($\hat{s}, a$)
  $(i', q', r) \leftarrow$ env.step(a); $\hat{s}' \leftarrow$ ENCODE(i', q')
  $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\hat{s}, a, r, \hat{s}')\}$
  return $\hat{s}'$
end function
$(i_0, q_0) \leftarrow$ env.reset(); $\hat{s}_{\text{act}} \leftarrow$ ENCODE(i_0, q_0)
while seeding buffer **do**
  $a \leftarrow$ UNIFORM_RANDOM_ACTION()
  $\hat{s}_{\text{act}} \leftarrow$ ACT_AND_STORE($\hat{s}_{\text{act}}, a$)
end while
while not converged **do**
  Sample $\{(\hat{s}_t, a_t, r_t, \hat{s}_{t+1})\}_{i=1}^{B} \sim \mathcal{D}$
  encode $\bar{z}_t \sim p_{\text{enc}}(\cdot \mid \hat{s}_t)$, $\bar{z}_{t+1} \sim p_{\text{enc}}(\cdot \mid \hat{s}_{t+1})$
  $\mathcal{L}_{\text{rew}} \leftarrow \dfrac{1}{B} \sum_{i=1}^{B} -\log p_{\text{rew}}(r_t \mid \bar{z}_t, a_t)$
  $\mathcal{L}_{\text{dyn}} \leftarrow \dfrac{1}{B} \sum_{i=1}^{B} -\log p_{\text{dyn}}(\bar{z}_{t+1} \mid \bar{z}_t, a_t)$
  $\mathcal{L}_{\text{KL}} \leftarrow \dfrac{1}{B} \sum_{i=1}^{B} \mathrm{KL}\big(p_{\text{enc}}(\cdot \mid \hat{s}_t) \,\|\, \mathcal{N}(0,1)\big)$
  $\mathcal{L}_{\text{S2P}} \leftarrow \lambda_{\text{rew}}\mathcal{L}_{\text{rew}} + \lambda_{\text{dyn}}\mathcal{L}_{\text{dyn}} + \beta\,\mathcal{L}_{\text{KL}} + \lambda_{H_{\text{enc}}}\mathcal{L}_{H_{\text{enc}}}$
  $\mathcal{L}_{\text{V}} \leftarrow \dfrac{1}{B} \sum_{i=1}^{B} \text{value_loss}(\nu, \hat{s}_t, a_t, r_t, \hat{s}_{t+1})$
  $\theta, \eta \leftarrow \theta, \eta - \alpha\nabla\mathcal{L}_{\text{S2P}}$; $\nu \leftarrow \nu - \alpha\nabla\mathcal{L}_{\text{V}}$
  $a \leftarrow \text{MPPI}(\hat{s}_{\text{act}})$
  $\hat{s}_{\text{act}} \leftarrow$ ACT_AND_STORE($\hat{s}_{\text{act}}, a$)
end while
return θ, η, ν

A.6 Planner Benchmarking

We measure planning cost for every method, and every planner configuration listed in Table 1, on a single NVIDIA L40S GPU with no other jobs running. A single plan is the full cost a controller pays between receiving an observation and having an action to execute. Each configuration is benchmarked over 100 independent trials; we discard the first as CUDA warm-up and report the median of the rest. For timing, we measure the wall-clock time to encode one observation and run one complete planner call. For memory, we record the peak GPU memory allocated by PyTorch during each plan. This figure includes the model weights, since those must also reside on the device during planning.

A.7 Planner Configuration Pruning

To determine which planner configurations can feasibly solve each RT task, we first train an expert policy with PPO on the non-RT variant of each task. We record 10,000 successful episodes with this expert, and take the minimum number of actions it required to first reach a success state as an estimate of the minimum number of actions needed to solve the task. Since this is the fewest actions observed across all 10,000 successes, the resulting criterion is permissive: it excludes only configurations that could not match the expert's fastest episode. Any planner configuration that cannot generate at least this many actions within the 10 s budget, given the plan latency measured in Section 5.4.1, is therefore considered infeasible and is not evaluated. The replanning interval K is factored in, as a scaling of the number of actions generated for each plan. For example, while it

Table 6: S2P training hyperparameters, shared across all tasks.

Observations	
Camera	Wrist, RGB, 224×224
Frame stack	3
Proprioception	25 dims (28 for PickCube)
Foundation Vision Encoder	
Backbone	DINOv3 ViT-S/16 (frozen)
Tokens per observation	3×196, dim. $384+$proprioception
S2P bisimulation encoder	
Architecture	Cross-attention pooling, 1 block
Learned queries	4
Attention embedding dim. / heads	512 / 4
Feed-forward and output MLPs	3 hidden layers $\times$ 512, ReLU
Latent dimension	512
World model	
Dynamics / reward MLPs	3 hidden layers $\times$ 512, ReLU
Loss weights (dynamics / reward / latent ℓ_2)	1 / 10 / 0.1
Prediction horizon	1
Value and policy (TD-MPC2)	
Q-ensemble size	5
Return discretization	101 bins, symlog range $[-10, 10]$
Target network update rate τ	0.01
Policy entropy coefficient	10^{-4}
Discount	0.95
Loss	
Reward weight λ_{rew}	10
Dynamics weight λ_{dyn}	1
Latent regularization weight β	0.1
Planner (MPPI) for Training	
Horizon	3
Samples / elites	512 / 64
Policy-proposed samples	64 (12.5%)
Iterations	6
Temperature	0.5
Std. range	$[0.05, 2]$
Replanning	Every step
Training	
Parallel environments	256
Replay buffer size	100,000 transitions
Random-action seed data	10,000 transitions
Batch size	64 transitions
Gradient steps per collected window	1
Optimizer	AdamW, lr 10^{-5}, weight decay 0.01
Training steps	1M
Evaluation	50 episodes

may not be feasible to plan with DINO-WM using $H = 5, I = 5, N = 128, K = 1$, i.e. replanning on on every step, it may be feasible if we replan every 5 steps, i.e. setting $K = 5$. Table 7 lists the resulting minimum action counts and the number of feasible configurations for each method.

Table 7: **Feasible Planner Configurations.** We restrict evaluation to a reduced set of planner configurations with $H \in \{1, 3, 5\}$, $I \in \{1, 3, 5\}$, and $K \in \{1, 3, 5\}$. Given a known minimum number of actions needed to solve a task, and a set time of 10 seconds to solve it, we calculate for each method how many of the 54 considered planner configurations could feasibly solve the task.

Task	Min. Actions	S2P	DINO-WM	Sparse Imag.	TC-WM	DINO-Bisim
PushCube-v1.1-RT	5	54	30	48	30	30
PickCube-v1.1-RT	6	54	26	45	26	26
LiftPegUpright-v1.1-RT	10	54	18	38	18	18