ColloDreamer: Latent Dynamics with Collocation and Value Networks

Michael Wan[1], Justin Chen[2]

[1]) m.wan@berkeley.edu
[2]) j.t.chen23@berkeley.edu

1 Abstract

While model-based RL has enjoyed significantly higher sample efficiency in comparison to model-free RL, there are a couple drawbacks to model-based RL that greatly limit its use case and potential. Most state-of-the-art learned dynamics models lack gradient-based optimization, and instead resort to gradient-free approaches such as random-shooting. However, these shooting methods grow extremely exhaustive with exponentially large state spaces from complex environments, which is largely impractical. On the other hand, gradient-based dynamics models fail to handle long-term horizon trajectory rollouts, as they lead to compounding errors. Still, there is great incentive to explore gradient-based solutions that can handle the compounding gradient errors. However, there has been relatively little work done in this space that has led to extraordinary results.

Training autonomous and intelligent agents to complete complex tasks in a meaningful way requires effective and efficient planning over the state and action spaces. Recently, we have seen both Dreamer and LatCo propose novel solutions to this planning problem through utilization of latent spaces and collocation, respectively. In this study, we explore the various optimization techniques leveraged by both Dreamer and LatCo in hopes of further improving long-horizon planning in model-based reinforcement learning.

We present a method that attempts the marriage of Dreamer and LatCo, as we believe a hybrid approach that combines elements from both algorithms can not only maintain the strengths of Dreamer and LatCo, but also mitigate the various drawbacks of both algorithms. Specifically, we implement LatCo's latent collocation algorithm along with Dreamer's value network and latent dynamics model learning process. Replacing the actor network in Dreamer allows optimization over both the state and action space, while using Dreamer's value network for reward calculation gives access to infinite time horizon reward sums, in contrast to LatCo's finite horizon reward sums. In doing so, we aim to accurately plan sequences of states and actions with a stronger signal from a better reward model. Ultimately, our proposed approach achieves comparable results to Dreamer and LatCo for a variety of DeepMind Control environments (namely Cheetah, Quadruped, and Reacher tasks). With components from both Dreamer and Latco, our proposed approach shows great promise in being an algorithm robust to distributional shifts and unseen data.

2 Introduction

Training autonomous and intelligent agents is a long-standing problem in reinforcement learning, and a prerequisite to solving intricate tasks in various high-dimensional and unknown environments. Much work has been done in model-based reinforcement learning to learn complex dynamics models, but classically, these model-based methods suffer from two drawbacks. On one hand, many of these methods rely on random-shooting to explore and optimize action selection, which is infeasible for exponentially high-dimensional environments and action spaces. On the other hand, gradient-based methods suffer from compounding errors that make long-term horizon planning infeasible.

Recently, two new model-based methods have pushed the frontier on long-term horizon planning: Dreamer [1] and LatCo [4]. Both these models utilize gradient-based exploration, using various techniques to overcome the classical drawbacks. Both models encode input image data into latent representations that uphold the Markovian property, allowing for training and planning to occur directly in the latent state space. Furthermore, LatCo optimizes over both actions and states, rather than only actions – this collocation technique diminishes compounding errors and local minima traps, giving birth to a robust long-term horizon planning method.

In our research, we examine our hypothesis that Dreamer and LatCo may significantly benefit from borrowing implementation details from the other. We aim to combine aspects of both LatCo and Dreamer to mitigate the downsides of the individual models. Specifically, we aim to replace Dreamer's actor network with LatCo's collocation technique, while retaining the Dreamer value network to better predict the value of latent states. Dreamer's actor network only optimizes over the action space, which may result in a high variance policy fragile against compounding errors and distributional shifts. LatCo calculates rewards rather naively and simply takes $\sum_{k=t}^{t+H} r(z_k)$, where $z_{t:t+H}$ is a trajectory of H latent states, and $r(\cdot)$ is the learned reward dynamics model. Instead, implementing Dreamer's value network allows for reward calculations that sum to $t = \infty$ rather than over a horizon of H time-steps, and this may prove to be a much stronger signal for training. Lastly, LatCo is able to explicitly explore high-reward areas by starting with relaxed dynamics constraints and slowly enforcing them through iterative updates to the Lagrangian equation, which we believe is a vital component that will benefit Dreamer.

Furthermore, with both models being built on a Markovian latent spaces, we investigate this space and its properties. Specifically, we study the effect of adjusting the size of the latent space, and test an additional loss term that aims to more efficiently explore high-reward regions in the latent space.

We apply our newly proposed method to several challenging tasks within the DeepMind Labs Control suite and show that it can produce intelligent agents capable of exploring and modeling their environments to achieve autonomous skill-acquisition. We find that this method achieves relatively comparable results when compared to those of Dreamer and LatCo in controlled settings.

Ultimately, our approach inherits the benefits of both collocation planning (as in LatCo) and effective planning through long-horizon reward models (as in Dreamer), allowing our agents to better learn and understand their environment.

3 Related Work

Dreamer

Open-Loop planning is a long-standing problem in model-based reinforcement learning. Dreamer [1] proposes a new approach by encoding images into and operating within a latent state-space, reducing the dimensionality of the input image and making image sequences a Markovian state space. Dreamer is able to do this by learning the separate functions:

$$\text{Representation model} : p(z_{t+1}|z_t, a_t, o_{t+1}) \tag{1}$$
$$\text{Transition model} : q(z_{t+1}|z_t, a_t) \tag{2}$$
$$\text{Reward model} : q(r_t|s_t) \tag{3}$$

Dreamer is then able to optimize the ELBO variational lower bound to avoid the intractability of exact likelihood:

$$\mathcal{J}_{\text{REC}} = \mathbb{E}\Big[\sum_t \Big(\ln q(z_t|s_t) + \ln q(r_t|s_t) - \beta KL\big(p(s_t|s_{t-1}, a_{t-1}, z_t)\big)||q(s_t|s_{t-1}, a_{t-1})\big)\Big)\Big] \tag{4}$$

In being able to directly predict and train in the latent space, the world model becomes much more efficient and practical.

At every timestep t, the world model encodes the input image I_t into the latent representation z_t, which is passed through to the actor-critic to produce an action a_t. The dynamics model is used to predict z_{t+1} from z_t, a_t, and the tuple $(a_t, z_{t+1}, r(z_{t+1}))$ is stored into the replay buffer, where further datapoints are sampled from to train the world model. The world model then generates imagined trajectories in the latent space to train the actor-critic model on.

An interesting detail to note that improves the algorithm's effectiveness lies in how the world model takes in stochastic image embeddings at every time step, making it more robust to unseen inputs. Our project aims to further augment this algorithm with the collocation planning component from LatCo [4], which we hope will produce more realistic and higher-reward trajectories for exploration.

DreamerV2

```
kl_loss =         alpha * compute_kl_loss(stop_grad(approx_posterior), prior)
        + (1 - alpha) * compute_kl_loss(approx_posterior, stop_grad(prior))
```

Listing 1: KL Balancing pseudocode

Using the previous work Dreamer [1] as a foundation, DreamerV2 [3] further improves the world model and adds Atari games compatibility to the agent. In particular, the authors claim that the algorithm is the first to achieve human-level performance on such games. The main technical improvements compared to the original Dreamer algorithm include (1) allowing for categorical latents rather than Gaussian latents and (2) implementing KL "balancing". The KL "balancing" technique attributes a heavier weight towards fitting the prior distribution to that of the imagined representations. Through empirical studies, DreamerV2 shows that using categorical representations rather than Gaussian ones performs much better on most tasks. The KL balancing algorithm pseudocode is described in Listing 1.

LatCo

LatCo [4] also builds off of the original Dreamer paper, using the same CNN image encoder and learned dynamics model with a RSSM (Recurrent State Space Model) [2]. Specifically, the dynamics model over the latent state-space is given by

$$q_\phi(z_{t+1}|z_t, a_t) \sim \mathcal{N}(\mu(z_t, a_t), \sigma(z_t, a_t)) \tag{5}$$

However, using the results from the revised DreamerV2 [3], we have seen that using Gaussian distributions to model the transition distributions performs worse than using a mixture of categorical distributions; our project aims to implement these ideas in LatCo. The main contribution involves utilizing a collocation-based planning approach in replacement of the actor-network one present in Dreamer [1]. Collocation only requires learning a dynamics model and a reward function, allowing the agent to optimize over states rather than just actions and achieve high rewards while ensuring that the predicted sequence conforms to model dynamics, i.e. a realistic and feasible trajectory. LatCo is able to do this by rewriting the dynamics as a Lagrangian, constraining the dynamics and action violations to a negligible value ϵ:

$$\begin{aligned}\min_{\lambda} \max_{p(z_{2:T}), a_{1:T-1}} \sum_t & \mathbb{E}_{p(z_t)}[r(z_t)] \\ & - \lambda(\|\text{mean}[q_\phi(z_t)] - \text{mean}[p(z_t)]\|^2 - \epsilon) \\ & - \lambda(\|\text{var}[q_\phi(z_t)] - \text{var}[p(z_t)]\|^2 - \epsilon)\end{aligned} \tag{6}$$

In this expression, we aim to learn a stochastic distribution $p(z_t)$, where $q_\phi(z_t)$ is shorthand for $\mathbb{E}_{p(z_t)}[q_\phi(z_{t+1}|z_t, a_t)]$. Additionally, $p(\cdot)$ is parameterized as a diagonal-covariance Gaussian $p(z_{1:T}) = \mathcal{N}(\mu_{1:T}, \sigma^2_{1:T})$. This modified approach finds high reward regions much faster than shooting methods, and overall leads to better optimization. Unfortunately, the formulation of this new optimization with the dynamics constraints means that gradient-based methods are no longer plausible and thus another key idea in LatCo involves the employment of dual-descent Lagrangian optimization to escape local minima in the saddle region. To optimally update the lambda values, we follow the technique introduced in the LatCo paper:

$$\lambda \leftarrow \lambda + \alpha \log\Big(\frac{\|z_{t+1} - \mu(z_t, a_t)\|^2}{\epsilon} + \eta\Big)\lambda \tag{7}$$

4 Dreamer Value Network with LatCo

In addition to implementing collocation planning onto Dreamer [1], we also aimed to add the Dreamer Value Network into the LatCo [4] logic. Specifically, for states $s_{t:t+H}$, LatCo aims to maximize the reward

$$R(t) = \sum_t^{t+H} r(z_t) \tag{8}$$

An early observation we made is that the return only considers a finite time horizon rather than an infinite horizon, in which case $R(t)$ may not be the optimal reward signal to train on. As such, using Dreamer's value network will allow us to estimate the total episodic reward. We can use our dynamics model in tandem to rewrite $R(t)$ as

$$R(t) = \sum_{k=t}^{t+H} r(z_k) + V(q_\phi(z_{t+H}, a_{t+H})) \tag{9}$$

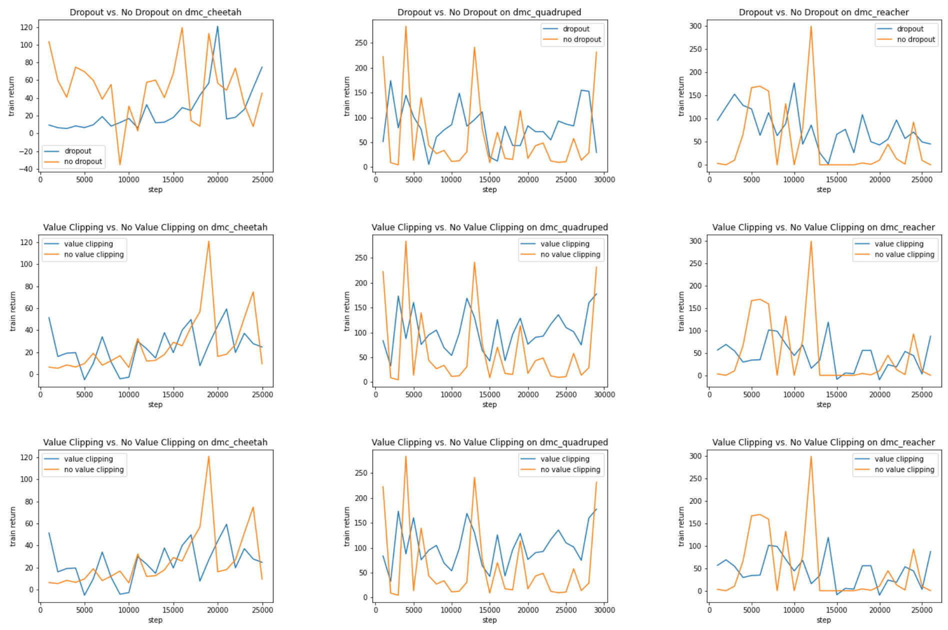

Figure 1: Our proposed method performance with or without dropout, value clipping, and batch normalization (rows 1, 2, 3), on environments DeepMind Control Cheetah, Quadruped (walk), Reacher (easy) (cols 1, 2, 3). Otherwise, all models were trained with the same configurations as in Figure 13.

We can also derive an expression for discounted rewards,

$$R(t) = \sum_{k=t}^{t+H} \gamma^{k-t} r(z_k) + \gamma^{t+H} V(q_\phi(z_{t+H}, a_{t+H})) \tag{10}$$

In the collocation training loop, because we have access to $z^{\text{plan}}_{t:t+H}, a^{\text{plan}}_{t:t+H}$, and $r(\cdot)$, we can continually use $\sum_{k=t}^{t+H} r(z^{\text{plan}}_k) + V(q_\phi(z^{\text{plan}}_{t+H}, a^{\text{plan}}_{t+H}))$ to set up the Bellman equation to train our value network $V(z_t)$. To add further stability to the performance of our value network, we experimented with various regularization methods, including:

- Batch Normalization: adding batch normalization layer either before or after the activation layers.
- Dropout: adding dropout layers to the value network, with probability $p_{\text{drop},i}$ of setting values to zero in the ith dropout layer.
- Value Clipping: clipping the total reward prediction to be between $[r_{\text{low}}, r_{\text{high}}]$.

Nevertheless, none of these made a significant impact on the effectiveness of the value network and model.

5 Collocation Planning with Dreamer

We aim to add LatCo's [4] collocation mechanism to Dreamer [1], replacing the actor network that Dreamer uses to optimize action selection. Instead of using the actor network to simply sample an action $a_\tau \sim \pi(a_\tau|s_\tau)$, we implement the collocation optimization logic.

There are two styles in which we train the Dreamer value network. The first method we attempted was to simply train the value network after taking all K optimization steps in the innermost for loop. Because we execute the planned actions $a_{t:H}^{\text{plan}}$, we thus have access to the observations $o_{t:H}$ as well as the ground truth latent representations $z_{t:H} \sim e_\phi(\cdot|o_{t:H})$, where e_ϕ is the encoder network used to encode observations into latent representations. We can then train $V(z_t)$ by calculating reward as described in Equations (9) and (10).

The second method is to continually train $V(z_t)$ in the innermost for loop. Although the $z_{t+1:H}^{\text{plan}}$ values are inferred, our hypothesis is that they are still suitable for reward calculation and value network training. Ultimately, this method gave a value network that performed slightly better than that of the first method. We hypothesize that the value network benefits largely from more training iterations in the second method, and is able to converge better. Algorithm 1 below describes the collocation procedure using the second method for value network training.

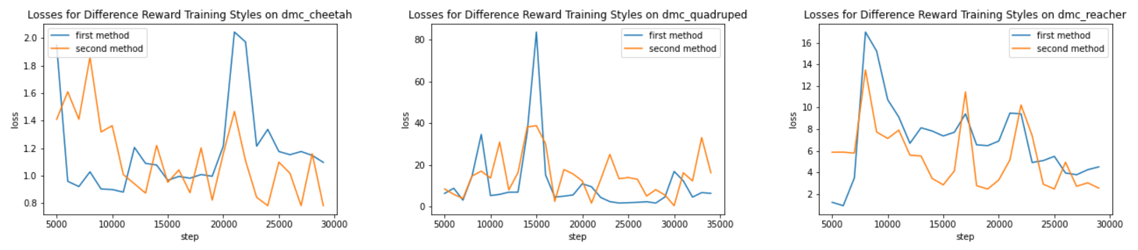

Figure 2: Our proposed method value losses for both reward training styles on DeepMind Control Cheetah, Quadruped (walk), Reacher (easy) environments, respectively. All other configurations are the same as those specified in Figure 13.

To learn the dynamics model, we use the Dreamer ELBO objective described in Equation (4).

6 Augmented Latent Objective for Improved Exploration

From running our model on the DeepMind Control Quadruped (walk) and Reacher environments, we hoped to improve the model's ability to efficient explore high-reward regions in the latent space to facilitate optimal planning. Drawing inspiration from Trust-Region Policy Optimization [5] and Proximal Policy Optimization [6], we design a new objective term that aims to maximize the probability of transitions between states z_{t-1} and z_t proportional to the gain in value determined by $V(z_t) - V(z_{t-1})$. As the training matures, we slowly decrease the weight of this objective while increasing the dynamics violation constraint in the LatCo [4] collocation logic. Specifically, the new

Algorithm 1 Collocation with Dreamer

```
Require: $\mathcal{D}, q_\phi(\cdot), e_\phi(\cdot), r_\phi(\cdot)$
  Sample $o_t \sim \mathcal{D}$
  Initialize $z_{1:H}^{\text{plan}}, a_{1:H}^{\text{plan}}$
  for $t \in [1, H]$ do
      Get predicted latent state $z_t \sim e_\phi(z_t|o_t)$
      $z_t^{\text{plan}} \leftarrow z_t$
      Define Lagrangian $\mathcal{L}$ from Equation. (6)
      for $k \in [1, K]$ do
          $z_{t+1:H}^{\text{plan}}, a_{t+1:H}^{\text{plan}} \leftarrow z_{t+1:H}^{\text{plan}}, a_{t+1:H}^{\text{plan}} + \nabla\mathcal{L}$
          Update $\lambda$ using Equation. (7)
          Update $V(z_t)$ using $\sum_{k=t}^{H} r_\phi(z_k^{\text{plan}}) + V'(q_\phi(z_H^{\text{plan}}, a_H^{\text{plan}}))$
      end for
      Execute $a_{t:H}^{\text{plan}}$, observe $o_{t:H}^{\text{env}}$
      Store $(o_{t:H}^{\text{env}}, a_{t:H}^{\text{env}}) \in \mathcal{D}$
      Learn $q_\phi(z_{t+1}|z_t, a_t), e_\phi(z_t|o_t), r_\phi(z_t)$ with Dreamer ELBO optimization
  end for
```

objective to be augmented is

$$\mathcal{J}_{\text{latent}} = \frac{1}{\lambda} m_\phi(t) A(t) \tag{11}$$

$$\text{where } m_\phi(t) = \frac{q_\phi(z_t|z_{t-1}, a_{t-1}) q_\phi(z_{t-1}|z_{t-2}, a_{t-2})}{q_\phi^{\text{old}}(z_t|z_{t-1}, a_{t-1}) q_\phi^{\text{old}}(z_{t-1}|z_{t-2}, a_{t-2})}$$

$$A(t) = r(z_t, a_t) + V(q_\phi(z_{t+1}|z_t, a_t)) - V(z_t)$$

The term $m_\phi(t)$ reflects the change in likelihood of the transition (z_{t-1}, z_t). When this change is large, then there is greater incentive to either minimize a negative advantage $A(t)$ or maximize a large advantage $A(t)$. Lastly, we notice that we need a reference to q_ϕ^{old}, which requires saving a target model.

We test the performance of our algorithm with and without $\mathcal{J}_{\text{latent}}$. Unfortunately, when trained with the augmented objective, the model did not perform any better.

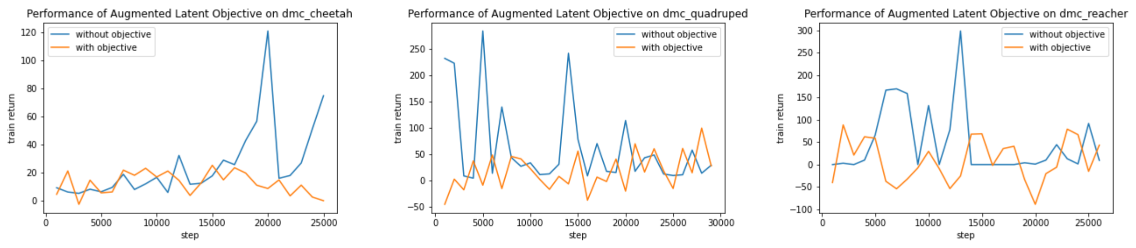

Figure 3: Our proposed method train returns with and without the augmented latent objective on DeepMind Control Cheetah, Quadruped (walk), Reacher (easy) environments, respectively. All other configurations are the same as those specified in Figure 13.

7 Results

Implementation

Our implementation uses TensorFlow and TensorFlow Probability. We trained our models using Google Colab and the provided free-tier NVIDIA Tesla K80 GPU with 12.7GB of RAM and 78.2GB of disk storage.

Environments

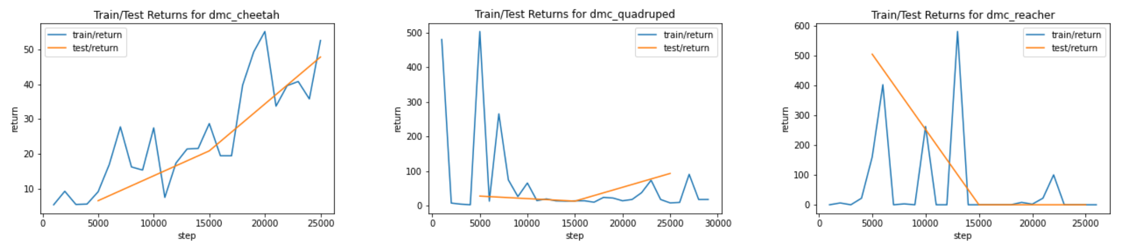

Figure 4: Baseline LatCo [4] performance on DeepMind Control Cheetah, Quadruped (walk), and Reacher environments, respectively. The LatCo model was trained up to 25k steps, with testing metrics collected at 5k steps and every 10k steps subsequently. The default hyperparameters were used, which can be found in the define_config methods located in planning_agent.py and base_agent.py as well as the yaml file config.yaml.

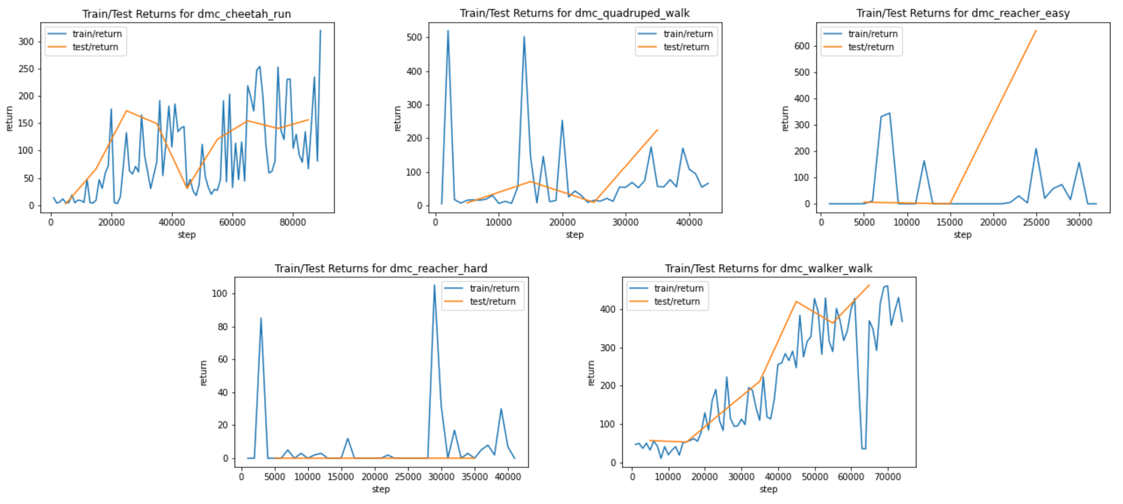

Figure 5: Baseline Dreamer [1] performance on DeepMind Control Cheetah, Quadruped (walk), Reacher (easy), Reacher (hard), and Walker (walk) environments, respectively. The model was trained up to 80k, 40k, 30k, 40k, and 70k steps, respectively. Testing metrics were collected at 5k steps and every 10k steps subsequently. The default hyperparameters were used (dreamer.py).

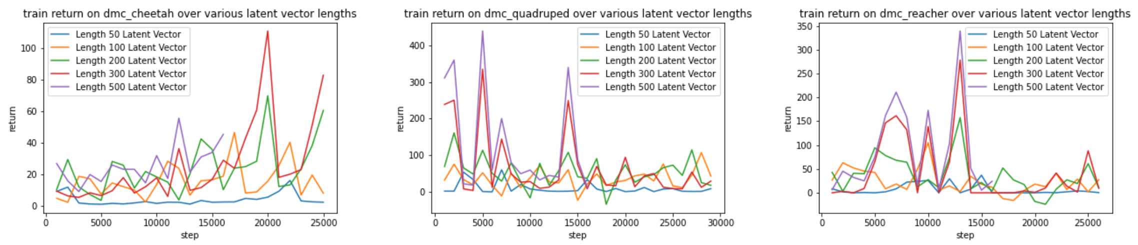

Figure 6: Our proposed method on DeepMind Control Cheetah, Quadruped (walk), and Reacher (easy) environments, respectively, with varying latent vector lengths of 50, 100, 200, 300, and 500. All other configurations are the same as those specified in Figure 13. We had difficulty training the model with length 500 latent vectors due to compute limits as we had to truncate our experiments early. Since length 300 latent vectors proved to perform at a similar level and did not significantly impact our ability to train models in Colab, it was chosen as the default parameter for all experiments.

In the figure above, we ran a hyperparameter search on the length of the latent vector representation used to train the dynamics model.

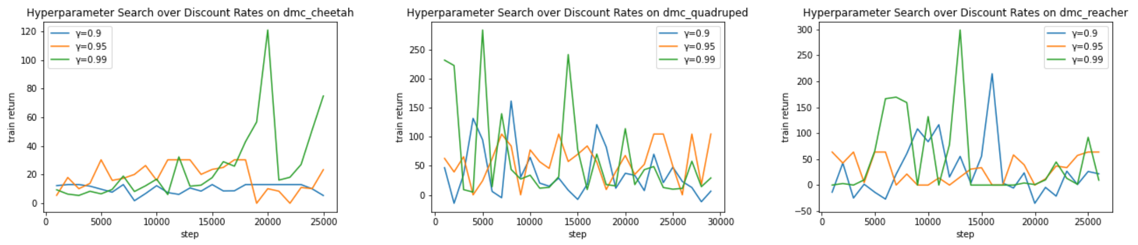

Figure 7: Our proposed method on DeepMind Control Cheetah, Quadruped (walk), and Reacher (easy) environments, respectively, with varying discount rates. All other configurations are the same as those specified in Figure 13.

In the figure above, we ran a hyperparameter search on the discount rate. Out of $\gamma = \{0.9, 0.95, 0.99\}$, 0.99 was found to be the optimal value which is consistent with Dreamer.

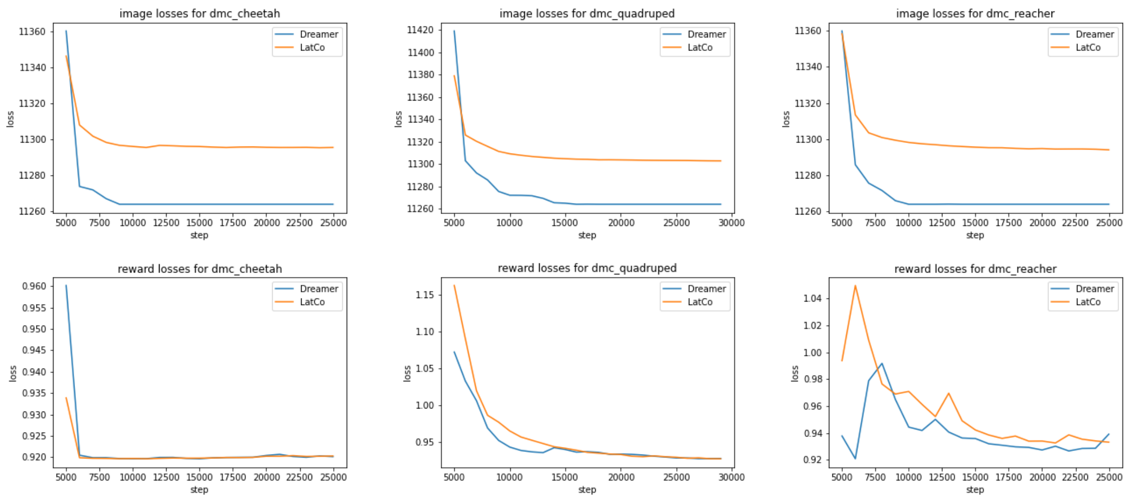

Figure 8: Baseline image and reward losses for dynamics models on DeepMind Control Cheetah, Quadruped (walk), and Reacher (easy) environments, respectively, for both Dreamer and LatCo models. All models were trained with the default parameters mentioned in the original papers.

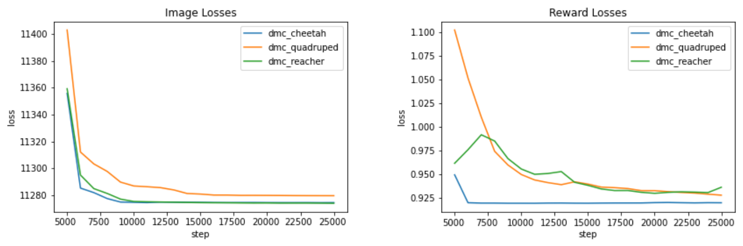

Figure 9: Our proposed method image and reward losses on DeepMind Control Cheetah, Quadruped (walk), and Reacher (easy) environments, respectively. All other configurations are the same as those specified in Figure 13.

As seen in the figure above, our method achieves similar image and reward losses and converges at a similar rate to both Dreamer and LatCo across each of the three tasks: DeepMind Control Cheetah, Quadruped (walk), and Reacher (easy).

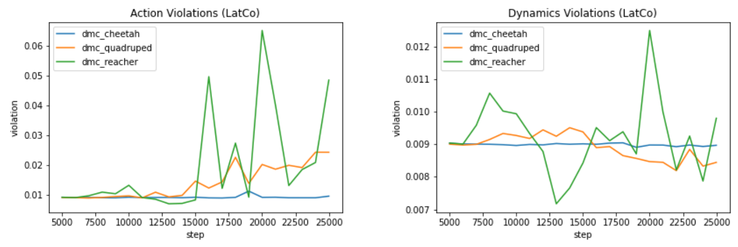

Figure 10: Baseline LatCo action and dynamics violations on DeepMind Control Cheetah, Quadruped (walk), and Reacher (easy) environments, respectively. All models were trained with the default parameters mentioned in the original paper.

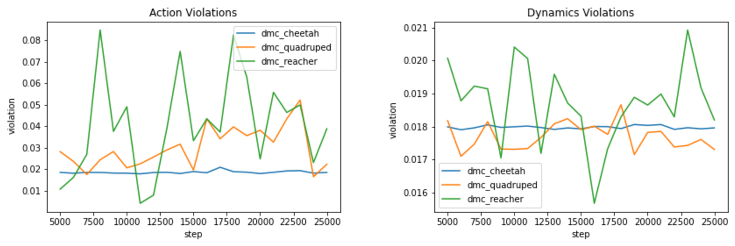

Figure 11: Our proposed method action and dynamics violations on DeepMind Control Cheetah, Quadruped (walk), and Reacher (easy) environments, respectively. All other configurations are the same as those specified in Figure 13.

As seen in the figure above, our method produces action and dynamic violations that are very similar to those of LatCo's across the same number of steps, although overall our method appears to have slightly higher violations.

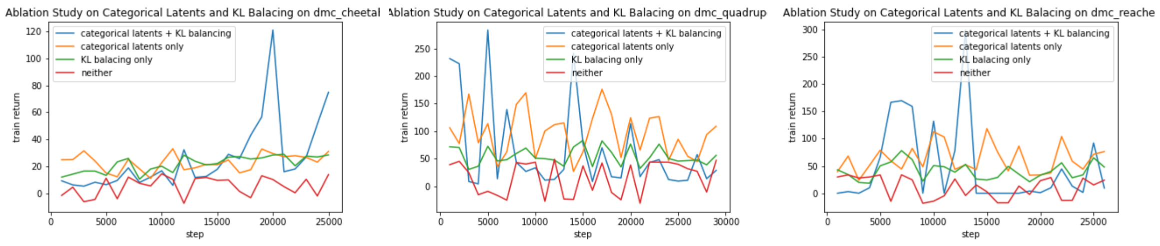

Figure 12: Our proposed method ablation study for categorical latent representations and KL balancing on DeepMind Control Cheetah, Quadruped (walk), and Reacher (easy) environments, respectively. All other configurations are the same as those specified in Figure 13.

As seen in the figure above, integration of both categorical latents and KL balancing from DreamerV2 outperforms either one individually, which in turn outperform an implementation with neither.

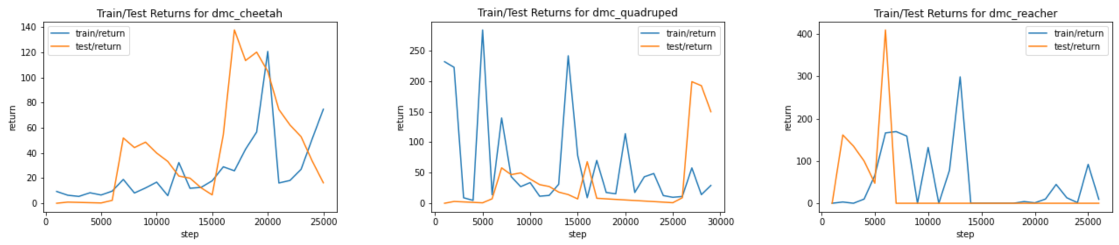

Figure 13: Our proposed method on DeepMind Control Cheetah, Quadruped (walk), and Reacher (easy) environments, respectively. The models were trained up to 25k, 30k, and 25k steps, respectively. Testing metrics were collected at 5k steps and every 10k steps subsequently. The model includes the optimal latent vector length (300), batch normalization, dropout, and value clipping configurations along with the KL regularization discussed earlier. The default hyperparameters from Dreamer (dreamer.py) and LatCo (planning_agent.py, base_agent.py, config.yaml) except for 'expl', which was changed to accommodate for the usage of mixed categorical latent representations instead of gaussians during exploration.

As seen in the figure above, our results are comparable to those of LatCo and Dreamer. For `dmc_cheetah_run`, our method achieves a train return of around 80 after 25k steps compared to around 50 and 100 for LatCo and Dreamer, respectively. For `dmc_quadruped_walk`, our method achieves a train return of around 50 after 30k steps compared to around 50 and 80 for LatCo and Dreamer, respectively. For `dmc_reacher_easy`, our method achieves a train return of around 100 after 25k steps compared to around 100 and 230 for LatCo and Dreamer, respectively.

8 Discussion

Conclusion

We have presented ColloDreamer, a new approach for model-based reinforcement learning that builds on Dreamer and LatCo. While our proposed method of joining LatCo's collocation planning mechanic with Dreamer's value network seemed promising in theory, our results show that significant further exploration of our work may be required to meaningfully surpass the benchmarks set by Dreamer and LatCo. Nevertheless, we were able to draw insightful conclusions from the various experiments and ablation studies we performed, which may be particularly useful for motivating future investigation. Comparing our proposed method's reward loss with the LatCo baselines shows that replacing the value network in LatCo with the one present in Dreamer improves performance as our proposed method's reward loss performs at least as well as Dreamer baselines. Additionally, we have shown that integrating categorical latents and KL balancing with LatCo collocation is viable as it achieves higher training return than LatCo baselines.

Future Work

Although our implemented method failed to yield significant improvements over Dreamer and LatCo, we believe that there are many promising avenues to explore from this project. One idea that we had initially proposed was to learn an amortized policy network that outputs actions to optimize the LatCo objective. We were not able to successfully explore or implement this idea, but in theory this would remove the need to conduct dual-descent Lagrangian optimization, providing a considerable speedup. Future approaches should also incorporate more environments, as we were only able to run our proposed model on three DeepMind Control environments (Cheetah, Quadruped, and Reacher) due to compatibility issues. Testing our model on extremely complex environments such as Starcraft II, Hanabi Learning, and Mujoco Soccer could better highlight the promise of our proposed model. Lastly, in the future, experiments should be performed with greater computational resources, as ours were immensely constrained by the base Google Colab resources.

9 Acknowledgments

We would like to thank Professor Sergey Levine and the graduate student instructors of CS285: Deep Reinforcement Learning. Our project underwent many changes from the proposal to the final paper, in both content and personnel. As a result, we would also like to thank all who submitted peer reviews for both the proposal and checkpoint for conceptual assistance, discussion, and paper review.

References

[1] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

[2] 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*, pages 2555–2565. PMLR, 2019.

[3] Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*, 2020.

[4] Oleh Rybkin, Chuning Zhu, Anusha Nagabandi, Kostas Daniilidis, Igor Mordatch, and Sergey Levine. Model-based reinforcement learning via latent-space collocation. In *International Conference on Machine Learning*, pages 9190–9201. PMLR, 2021.

[5] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[6] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.