GOAL-AWARE PREDICTION: LEARNING TO MODEL WHAT MATTERS

Suraj Nair, Silvio Savarese, Chelsea Finn
Stanford University

1 INTRODUCTION

Enabling artificial agents to learn from their prior experience and generalize their knowledge to new tasks and environments remains an open and challenging problem. Unlike humans, who have the remarkable ability to quickly generalize skills to new objects and task variations, current methods in multi-task reinforcement learning require heavy supervision across many tasks before they can even begin to generalize well. One way to reduce the dependence on heavy supervision is to leverage data that the agent can collect autonomously without rewards or labels, termed *self-supervision*. One of the more promising directions in learning transferable knowledge from this unlabeled data lies in learning the dynamics of the environment, as the physics underlying the world are often consistent across scenes and tasks. However learned dynamics models do not always translate to good downstream task performance, an issue which we study and attempt to mitigate in this work.

Our primary hypothesis is that "objective mismatch" between the training objective of the learned model (future state reconstruction), and the downstream planner or policy (completing a specified task) is one of the primary limitations in learning models of high dimensional states. In other words, the learned model is encouraged to model large portions of the state which may be irrelevant to the task at hand. Consider for example the task of picking up a pen from a cluttered desk. The standard training objective of the learned model would encourage it to equally weigh modeling the pen and all objects on the table, when given the downstream task, modeling the pen and adjacent objects precisely is clearly the most critical.

To that end, we propose goal-aware prediction (GAP), a framework for learning forward dynamics models that direct their capacity differently conditioned on the task, resulting in a model that is more accurate on trajectories most relevant to the downstream task. Specifically, we propose to learn a latent representation of not just the state, but both the state and goal, and to learn dynamics in this latent space. Furthermore, we can learn this latent space in a way that focuses primarily on parts of the state relative to achieving the goal, namely by reconstructing the *goal-state residual* instead of the full state. We find that this modification combined with training via goal-relabeling Andrychowicz et al. (2017) allows us to learn expressive, task-conditioned dynamics models in an *entirely self-supervised* manner. We observe that GAP learns dynamics that achieve significantly lower error on task relevant states, and as a result outperforms standard latent dynamics model learning Hafner et al. (2018) and self-supervised model-free reinforcement learning Nair et al. (2018) across a range of vision based control tasks.

2 RELATED WORK

One area of past work significantly related to our work is *self-supervised reinforcement learning*, where an agent leverages data it collected autonomously to learn meaningful behaviors. Another related area is *model-based reinforcement learning*, where the agent learns a model of the dynamics of an environment, and uses it to complete a task. Lastly, like our work which studies the relationship between model error and task performance, several prior works have also explored studying *model errors* and learning better models for specific tasks. We discuss the related work in each of these areas in depth in Appendix A.

3 GOAL-AWARE PREDICTION

3.1 PRELIMINARIES

We formalize our problem setting as a goal-conditioned Markov decision process (MDP) defined by the tuple $(\mathcal{S}, \mathcal{A}, p, \mathcal{G}, \lambda)$ where $s \in \mathcal{S}$ is the state space, $a \in \mathcal{A}$ is the action space, $p(s_{t+1}|s_t, a_t)$ governs the environment dynamics, $p(s_0)$ corresponds to the initial state distribution, $\mathcal{G} \subset \mathcal{S}$ represents the *unknown* set of goal states which is a subset of possible states, and λ is the discount

factor. Note that this is simply a special case of a Markov decision process, where we do not have access to extrinsic reward (i.e. it is self-supervised), and where we separate the state and goal for notational clarity. We will assume that the agent has collected an unlabeled dataset $\mathcal{D}$ that consists of N trajectories $[\tau_1, ..., \tau_N]$, and each trajectory τ consists of a sequence of state action pairs $[(s_0, a_0), (s_1, a_1), ..., (s_T)]$. We will denote the estimated distance between two states as $C(s_t, s_g) = ||s_t - s_g||_2^2$, which may not accurately reflect the true distance, e.g. when states correspond to images. At test time, the agent is initialized at a start state $s_0 \sim p(s_0)$ with a goal state s_g sampled at random from $\mathcal{G}$, and must minimize cost $\mathcal{C}(s_t, s_g)$. We assume that for any states s_t, s_g we can measure $\mathcal{C}(s_t, s_g)$ as the distance between the states, for example in image space $\mathcal{C}$ would be pixel distance. Success is measured as reaching within some true distance of s_g. In the model-based RL setting we consider here, the agent aims to solve the RL problem by learning a model of the dynamics $p_\theta(s_{t+1}|s_t, a_t)$ from experience, and using that model to plan a sequence of actions or optimize a policy.

3.2 Understanding the Effect of Model Error on Task Performance

A key challenge in model-based RL is that dynamics prediction error does not directly correspond to task performance. Specifically, for good task performance, certain model errors may be more costly than others, and if errors are simply distributed uniformly over dynamics predictions, these critical areas may be exploited when selecting actions downstream. Intuitively, when optimizing actions for a given task, we would like our model to be accurate at predicting actions that are important for completing the task, while the model likely does not need to be as accurate on trajectories that are completely unrelated to the task. In this section, we formalize this intuition.

Suppose the model is used to select from N action sequences $a^i_{1:T}$, each with expected final cost $c^*_i = E_{p(s_{t+1}|s_t,a_t),a^i_{1:T}}[\mathcal{C}(s_T, s_g)]$. Without loss of generality, let $c^*_1 < c^*_2 ... < c^*_N$, i.e. the order of action sequences is sorted by their cost under the true model, which is unknown to the agent. Denote $\hat{c}_i$ as the predicted final cost of action sequence $a^i_{1:T}$ under the learned model, i.e. $\hat{c}_i = E_{p_\theta(s_{t+1}|s_t,a_t),a^i_{1:T}}[\mathcal{C}(\hat{s_T}, s_g)]$. Moreover, we consider a policy that simply selects the lowest cost action sequence under the model: $\hat{a} = \min_{a^i_{1:T}} \hat{c}_i$. We define optimal behavior as selecting the lowest cost action sequence $a^1_{1:T}$. Under this set-up, we now analyze the result of model errors.

Theorem 3.1. *Assume that the model error is such that*

$$|c^*_1 - \hat{c}_1| < \epsilon \tag{1}$$

Then, the policy's behavior is optimal if the errors of other action sequences are bounded more loosely as follows:

$$|c^*_i - \hat{c}_i| < (c^*_i - c^*_1) - \epsilon \quad \forall i > 1 \tag{2}$$

Furthermore, if Equations 1 or 2 ***do not*** *hold, then there is a nonzero probability of the policy selecting a sub-optimal action.*

See Appendix B.1 for proof. Theorem 3.1 suggests that, for good task performance, model error must be low for good trajectories, and we can afford higher model error for trajectories with higher cost. That is, **the greater the trajectory cost, the more model error we can afford**. Specifically, we see that the allowable error bound on cost of an action sequence from a learned model scales linearly with how far from optimal that action sequence is, in order to maintain the optimal policy for the downstream task. Note, that while Theorem 3.1 relates cost prediction error (not explicitly dynamics prediction error) to planning performance, we can expect dynamics prediction error to relate to the resulting cost prediction error. We also verify this empirically in the next section.

3.3 Redistributing Model Errors with Goal Aware Prediction

Our analysis above suggests that distributing errors uniformly across action sequences will not lead to good task performance. Yet, standard model learning objectives will encourage just that. In this section, we aim to change our model learning approach in an aim to redistribute errors more appropriately. Ideally, we would like to encourage the model to have more accurate predictions on the trajectories which are relevant to the task. However, actually identifying how relevant a trajectory is to a specific goal s_g can be challenging.

To that end, we propose goal-aware prediction (GAP) as a technique to re-distribute model error by learning a model that, in addition to the current state and action, s_t and a_t, is conditioned on

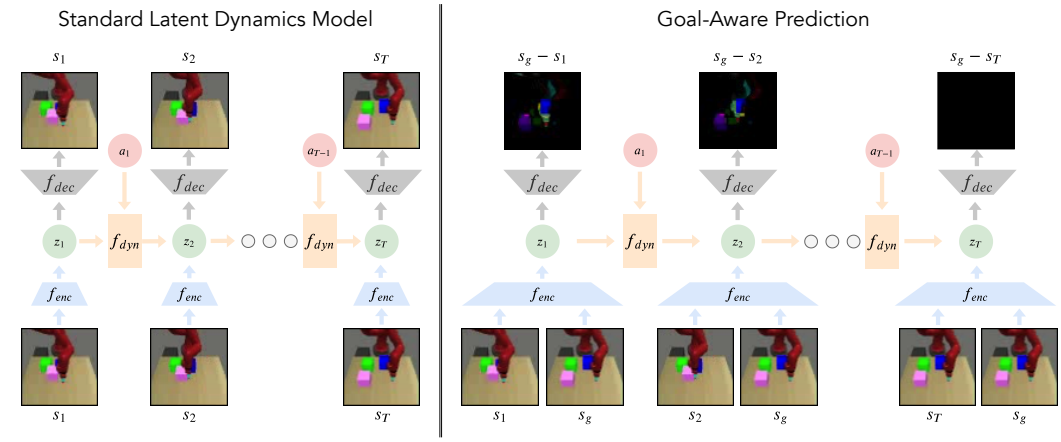

Figure 1: **Goal-Aware Prediction:** Compared to a standard latent dynamics model **(left)**, our proposed method, goal-aware prediction (GAP), **(right)** encodes both the current state s_t and goal s_g into a single latent space z_t. Samples from the distribution of z_t are then used to reconstruct the residual between the current state and goal $s_g - s_t$. Simultaneously, we learn the forward dynamics in the latent space z, specifically, learning to predict z_{t+1} from z_t and a_t. Using this approach, we obtain 2 favorable properties: (1) the latent space only needs to capture components of the scene relevant to the goal, and (2) the prediction task becomes easier (the residual approaches 0) for states closer to the goal.

the goal state s_g, and instead of reconstructing the future state s_{t+1}, reconstructs the difference between the future state and the goal state, that is: $p_\theta((s_g - s_{t+1})|s_t, s_g, a_t)$. Critically, to train GAP effectively, we need action sequences that are relevant to the corresponding goal. To accomplish this, we can choose to set the goal state for a given action sequence as the final state of that trajectory, i.e. using hindsight relabeling Andrychowicz et al. (2017). Specifically, given a trajectory $[(s_1, a_1), (s_2, a_2), ..., (s_T)]$, the goal is assigned to be the last state in the trajectory $s_g = s_T$, and for all states $\{s_t | 1 \leq t \leq T-1\}$, $p_\theta(s_t, s_g, a_t)$ is trained to reconstruct the delta to the goal $s_g - s_{t+1}$.

Our proposed GAP method has two clear benefits over standard dynamics models. First, assuming that the agent is not in a highly dynamic scene with significant background distractor motion, by modeling the delta between s_g and s_t, p_θ only needs to model components of the state which are relevant to the current goal. This is particularly important in high dimensional settings where there may be large components of the state which are irrelevant to the task, and need not be modeled. Second, states s_t that are temporally close to the goal state s_g will have a smaller delta $s_g - s_t$, approaching zero along the trajectory until $s_t = s_g$. As a result, states closer to the goal will be easier to predict, biasing the model towards low error near states relevant to the goal. In light of our analysis of model error in the previous sections, we hypothesize that this model will lead to better downstream task performance compared to a standard model that distributes errors uniformly across trajectories.

We implement GAP with a latent dynamics model, as shown in Figure 1. Given a dataset of trajectories $[\tau_1, ..., \tau_N]$, we sample sequences of states $[(s_1, a_1), ..., (s_T)]$ where we re-label goal for the trajectory as $s_g = s_T$. The GAP model consists of three components, (1) an encoder $f_{enc}(z_t|s_t, s_g; \theta_{enc})$ that encodes the state s_t and goal s_g into a latent space z_t, (2) a decoder $f_{dec}(s_g - s_t|z_t; \theta_{dec})$ that decodes samples from the latent distribution into $s_g - s_t$, and (3) a forward dynamics model in the latent space $f_{dyn}(z_{t+1}|z_t, a_t; \theta_{dyn})$ which learns to predict the future latent distribution over z_{t+1} from z_t and action a_t. In our experiments we work in the setting where states are images, so $f_{enc}(z_t|s_t, s_g)$ and $f_{dec}(s_g - s_t|z_t)$ are convolutional neural networks, and $f_{dyn}(z_{t+1}|z_t, a_t)$ is a fully-connected network. The full set of parameters $\theta = \{\theta_{enc}, \theta_{dec}, \theta_{dyn}\}$ are jointly optimized. Exact architecture and training details for all modules can be found in the Appendix. We also go over the details of implementing GAP in Appendix D including data collection, model training, and planning.

4 Experiments

In our experiments, we investigate three primary questions: **(1)** Does using our proposed technique for goal-aware prediction (GAP) re-distribute model error such that predictions are more accurate on good trajectories? **(2)** Does re-distributing model errors using GAP result in better performance in downstream tasks? **(3)** Does GAP scale to the complexity of real world images? (Appendix C.2)

We test in a simulated manipulation environment, comparing our approach **GAP-Residual (Ours)** with **Residual Model** which predicts residual to the current state, variants of a standard latent dy-

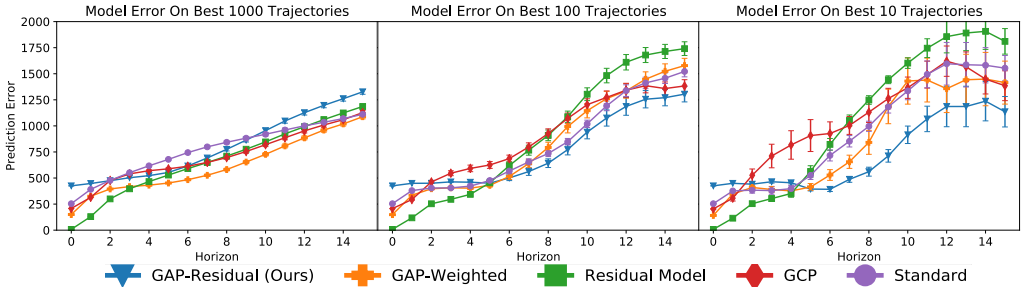

Figure 2: **Distribution of Model Errors:** We examine the distribution of model prediction errors of our proposed method (GAP) compared to baselines over 1000 random action sequences, evaluated on the "Task 1" domain. All y-axis are from [0, 2000] corresponding to model mean squared error (with standard error bars), and the x-axis corresponds to number of time steps predicted. Naturally, we observe that model error increases as the prediction horizon increases, for all approaches. However, although all approaches have a similar error over all 1000 action sequences (left), our proposed GAP method has significantly lower error on the *best* 10 trajectories (right). This suggests that changing the model objective through predicting the goal-state residual leads to more accurate predictions in areas that matter in downstream tasks.

	task 1 success	task 2 success	task 3 success	task 4 success
GAP-residual (ours)	**0.48 (0.04)**	**0.38 (0.03)**	0.48 (0.04)	**0.20 (0.03)**
GAP-weighted	0.41 (0.04)	0.21 (0.03)	0.25 (0.03)	0.12 (0.02)
residual model	0.43 (0.04)	0.01 (0.0)	**0.59 (0.03)**	0.15 (0.02)
GCP	0.33 (0.03)	0.0 (0.0)	0.38 (0.03)	0.06 (0.02)
standard	0.35 (0.03)	0.13 (0.02)	0.33 (0.03)	0.16 (0.03)
action reconstruction	0.02 (0.0)	0.0 (0.0)	0 (0.0)	**0.21 (0.03)**
RIG	0.42 (0.04)	0.0 (0.0)	0.13 (0.02)	0.14 (0.02)

Table 1: **Success rate on tabletop manipulation** Given multiple blocks/door on a table and raw RGB goal image and observations, the agent must push either 1 (Task 1) or 2 (Task 2) blocks to target positions, or close (Task 3) or open (Task 4) the door. Our proposed GAP method achieves a higher success rate than standard model-based approaches, as well as self-supervised model-free techniques on 3 out of 4 tasks.

namics model **Standard, GCP, GAP-Weighted**, as well as model-free RL **RIG** and latent dynamics learned with action reconstruction **Action Reconstruction**. Further details about experimental domain and comparisons can be found in Appendix C.1.

Experiment 1: Does GAP Favorably Redistribute Model Error? In our first set of experiments, we study how GAP affects the distribution of model errors, and if it leads to lower model error on task relevant trajectories. We sample 1000 random action sequences of length 15 in the Task 1 domain. We compute the true next states $s^1_{1:H}, ..., s^{1000}_{1:H}$ and costs $c^1, ..., c^{1000}$ for each action sequence by feeding it through the true simulation environment. We then get the predicted next states from our learned models, including GAP as well the comparisons outlined above. We then examine the model error of each approach, and how it changes when looking at all trajectories, versus the lowest cost trajectories.

We present our analysis in Figure 2. We specifically look at the model error on all 1000 action sequences, the top 100 action sequences, and the top 10 action sequences. First, we observe that model error increases with the prediction horizon, which is expected due to compounding model error. More interestingly, however, we observe that while our proposed GAP-Residual approach has the highest error averaged across all 1000 action sequences, it has by far the lowest error on the top 10. This suggests that the goal conditioned prediction of the goal-state residual indeed encourages low model error in the relevant parts of the state space. Furthermore, we see that the conditioning on and reconstructing the difference to the *actual goal* is in fact critical, as the Residual Model which instead is conditioned on and predicts the residual to the first frame actually gets worse error on the lowest cost trajectories.

Experiment 2: Does GAP Lead to Better Downstream Task Performance? To study downstream task performance, we test on the tabletop manipulation tasks described in Appendix C.1, specifically 1 block, 2 block pushing, door closing, and door opening (with distractor objects). We perform planning over 30 timesteps with the learned models as described in Section 3, and report the final success rate of each task over 200 trials in Table 1. We see that in the easier Task 1, GAP-Residual performance the best, while GAP-Weighted, Residual Model, and RIG all achieve reasonable performance. However, as we move to the more difficult task of manipulating 2 objects (Task 2) precise modeling of the relevant objects becomes especially important; in this case, we see that GAP-Residual far exceeds the performance of the other approaches, with most methods failing completely.

REFERENCES

Pulkit Agrawal, Ashvin Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to poke by poking: Experiential learning of intuitive physics. *ArXiv*, abs/1606.07419, 2016.

Brandon Amos, Laurent Dinh, Serkan Cabi, Thomas Rothörl, Sergio Gomez Colmenarejo, Alistair Muldal, Tom Erez, Yuval Tassa, Nando de Freitas, and Misha Denil. Learning awareness models. *CoRR*, abs/1804.06318, 2018a. URL http://arxiv.org/abs/1804.06318.

Brandon Amos, Ivan Dario Jimenez Rodriguez, Jacob Sacks, Byron Boots, and J. Zico Kolter. Differentiable mpc for end-to-end planning and control. *ArXiv*, abs/1810.13400, 2018b.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *CoRR*, abs/1707.01495, 2017.

Mohammad Babaeizadeh, Chelsea Finn, Dumitru Erhan, Roy H. Campbell, and Sergey Levine. Stochastic variational video prediction. *CoRR*, abs/1710.11252, 2017.

Ershad Banijamali, Rui Shu, Mohammad Ghavamzadeh, Hung Hai Bui, and Ali Ghodsi. Robust locally-linear controllable embedding. *ArXiv*, abs/1710.05373, 2017.

Arunkumar Byravan, Jost Tobias Springenberg, Abbas Abdolmaleki, Roland Hafner, Michael Neunert, Thomas Lampe, Noah Siegel, Nicolas Manfred Otto Heess, and Martin A. Riedmiller. Imagined value gradients: Model-based policy optimization with transferable latent dynamics models. *ArXiv*, abs/1910.04142, 2019.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *CoRR*, abs/1805.12114, 2018. URL http://arxiv.org/abs/1805.12114.

Felipe Codevilla, Matthias Müller, Alexey Dosovitskiy, Antonio López, and Vladlen Koltun. End-to-end driving via conditional imitation learning. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1–9, 2017.

Sudeep Dasari, Frederik Ebert, Stephen Tian, Suraj Nair, Bernadette Bucher, Karl Schmeckpeper, Surender Singh, Sergey Levine, and Chelsea Finn. Robonet: Large-scale multi-robot learning. *ArXiv*, abs/1910.11215, 2019.

Marc Peter Deisenroth and Carl Edward Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *In Proceedings of the International Conference on Machine Learning*, 2011.

Pierluca D'Oro, Alberto Maria Metelli, Andrea Tirinzoni, Matteo Papini, and Marcello Restelli. Gradient-aware model-based policy search. *ArXiv*, abs/1909.04115, 2019.

Alexey Dosovitskiy and Vladlen Koltun. Learning to act by predicting the future. *ArXiv*, abs/1611.01779, 2016.

Frederik Ebert, Chelsea Finn, Alex X. Lee, and Sergey Levine. Self-supervised visual planning with temporal skip connections. *CoRR*, abs/1710.05268, 2017.

Frederik Ebert, Chelsea Finn, Sudeep Dasari, Annie Xie, Alex X. Lee, and Sergey Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *CoRR*, abs/1812.00568, 2018.

Benjamin Eysenbach, Ruslan Salakhutdinov, and Sergey Levine. Search on the replay buffer: Bridging planning and reinforcement learning. *CoRR*, abs/1906.05253, 2019.

Kuan Fang, Yuke Zhu, Animesh Garg, Silvio Savarese, and Li Fei-Fei. Dynamics learning with cascaded variational inference for multi-step manipulation. *ArXiv*, abs/1910.13395, 2019.

Amir-Massoud Farahmand, Andre Barreto, and Daniel Nikovski. Value-Aware Loss Function for Model-based Reinforcement Learning. In Aarti Singh and Jerry Zhu (eds.), *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pp. 1486–1494, Fort Lauderdale, FL, USA, 20–22 Apr 2017. PMLR. URL http://proceedings.mlr.press/v54/farahmand17a.html.

Chelsea Finn and Sergey Levine. Deep visual foresight for planning robot motion. *CoRR*, abs/1610.00696, 2016.

C. Daniel Freeman, Luke Metz, and David Ha. Learning to predict without looking ahead: World models without forward prediction. *ArXiv*, abs/1910.13038, 2019.

Carles Gelada, Saurabh Kumar, Jacob Buckman, Ofir Nachum, and Marc G. Bellemare. Deepmdp: Learning continuous latent space models for representation learning. *CoRR*, abs/1906.02736, 2019. URL http://arxiv.org/abs/1906.02736.

Karol Gregor, Danilo Jimenez Rezende, Frédéric Besse, Yan Wu, Hamza Merzic, and Aaron van den Oord. Shaping belief states with generative environment models for rl. In *NeurIPS*, 2019.

David Ha and Jürgen Schmidhuber. World models. *ArXiv*, abs/1803.10122, 2018.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *ArXiv*, abs/1801.01290, 2018.

Danijar Hafner, Timothy P. Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. *CoRR*, abs/1811.04551, 2018.

Danijar Hafner, Timothy P. Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *ArXiv*, abs/1912.01603, 2019.

Kristian Hartikainen, Xinyang Geng, Tuomas Haarnoja, and Sergey Levine. Dynamical distance learning for unsupervised and semi-supervised skill discovery. *ArXiv*, abs/1907.08225, 2019.

Aaron Havens, Yi Ouyang, Prabhat Nagarajan, and Yasuhiro Fujita. Learning latent state spaces for planning through reward prediction, 2020. URL https://openreview.net/forum?id=ByxJjlHKwr.

Brian Ichter and Marco Pavone. Robot motion planning in learned latent spaces. *CoRR*, abs/1807.10366, 2018.

Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. In *NeurIPS*, 2019.

Leslie Pack Kaelbling. Learning to achieve goals. In *IJCAI*, pp. 1094–1098, 1993.

Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H. Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, Ryan Sepassi, George Tucker, and Henryk Michalewski. Model-based reinforcement learning for atari. *ArXiv*, abs/1903.00374, 2019.

Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arxiv:Preprint*, 2018.

Thanard Kurutach, Aviv Tamar, Ge Yang, Stuart J. Russell, and Pieter Abbeel. Learning plannable representations with causal infogan. *CoRR*, abs/1807.09341, 2018.

Alex X. Lee, Richard Zhang, Frederik Ebert, Pieter Abbeel, Chelsea Finn, and Sergey Levine. Stochastic adversarial video prediction. *CoRR*, abs/1804.01523, 2018.

Alex X. Lee, Anusha Nagabandi, Pieter Abbeel, and Sergey Levine. Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model. *ArXiv*, abs/1907.00953, 2019.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *CoRR*, abs/1504.00702, 2015.

Kara Liu, Thanard Kurutach, Pieter Abbeel, and Aviv Tamar. Hallucinative topological memory for zero-shot visual planning, 2020. URL https://openreview.net/forum?id=BkgF4kSFPB.

Rowan McAllister and Carl Edward Rasmussen. Improving pilco with bayesian neural network dynamics models. 2016.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen. King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

Anusha Nagabandi, Kurt Konoglie, Sergey Levine, and Vikash Kumar. Deep dynamics models for learning dexterous manipulation. *ArXiv*, abs/1909.11652, 2019.

Ashvin Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. Visual reinforcement learning with imagined goals. *CoRR*, abs/1807.04742, 2018.

Ashvin Nair, Shikhar Bahl, Alexander Khazatsky, Vitchyr Pong, Glen Berseth, and Sergey Levine. Contextual imagined goals for self-supervised robotic learning. *ArXiv*, abs/1910.11670, 2019.

Suraj Nair and Chelsea Finn. Hierarchical foresight: Self-supervised learning of long-horizon tasks via visual subgoal generation. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=H1gzR2VKDH.

OpenAI. Openai five. https://blog.openai.com/openai-five/.

OpenAI, Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Józefowicz, Bob McGrew, Jakub W. Pachocki, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning dexterous in-hand manipulation. *CoRR*, abs/1808.00177, 2018. URL http://arxiv.org/abs/1808.00177.

Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 488–489, 2017.

Chris Paxton, Yotam Barnoy, Kapil D. Katyal, Raman Arora, and Gregory D. Hager. Visual robot task planning. *CoRR*, abs/1804.00062, 2018.

Lerrel Pinto and Abhinav Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. *CoRR*, abs/1509.06825, 2015.

Sébastien Racanière, Theophane Weber, David P. Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adrià Puigdomènech Badia, Oriol Vinyals, Nicolas Manfred Otto Heess, Yujia Li, Razvan Pascanu, Peter W. Battaglia, Demis Hassabis, David Silver, and Daan Wierstra. Imagination-augmented agents for deep reinforcement learning. *ArXiv*, abs/1707.06203, 2017.

R Rubinstein and D Kroese. *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation and Machine Learning*. 01 2004.

Oleh Rybkin, Karl Pertsch, Frederik Ebert, Dinesh Jayaraman, Chelsea Finn, and Sergey Levine. Goal-conditioned video prediction, 2020. URL `https://openreview.net/forum?id=B1g79grKPr`.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International Conference on Machine Learning*, 2015.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy P. Lillicrap, and David Silver. Mastering atari, go, chess and shogi by planning with a learned model. *ArXiv*, abs/1911.08265, 2019.

David Silver, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016. URL `http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html`.

Aravind Srinivas, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Universal planning networks. *ArXiv*, abs/1804.00645, 2018.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL `http://incompleteideas.net/book/the-book-2nd.html`.

Brijen Thananjeyan, Ashwin Balakrishna, Ugo Rosolia, Felix Li, Rowan McAllister, Joseph E. Gonzalez, Sergey Levine, Francesco Borrelli, and Ken Goldberg. Safety augmented value estimation from demonstrations (saved): Safe deep model-based rl for sparse cost robotic tasks. 2019.

Rishi Veerapaneni, John D. Co-Reyes, Michael Chang, Michael Janner, Chelsea Finn, Jiajun Wu, Joshua B. Tenenbaum, and Sergey Levine. Entity abstraction in visual model-based reinforcement learning. *ArXiv*, abs/1910.12827, 2019.

Ruben Villegas, Arkanath Pathak, Harini Kannan, Dumitru Erhan, Quoc Le, and Honglak Lee. High fidelity video prediction with large stochastic recurrent neural networks. 11 2019.

Angelina Wang, Thanard Kurutach, Kara Liu, Pieter Abbeel, and Aviv Tamar. Learning robotic manipulation through visual planning and acting. *CoRR*, abs/1905.04411, 2019.

Tingwu Wang and Jimmy Ba. Exploring model-based planning with policy networks. *ArXiv*, abs/1906.08649, 2019.

Manuel Watter, Jost Tobias Springenberg, Joschka Boedecker, and Martin A. Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. *CoRR*, abs/1506.07365, 2015.

Annie Xie, Frederik Ebert, Sergey Levine, and Chelsea Finn. Improvisation through physical understanding: Using novel objects as tools with visual foresight. *CoRR*, abs/1904.05538, 2019.

Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan R Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. *ArXiv*, abs/1910.10897, 2019a.

Tianhe Yu, Gleb Shevchuk, Dorsa Sadigh, and Chelsea Finn. Unsupervised visuomotor control through distributional planning networks. *CoRR*, abs/1902.05542, 2019b.

Andy Zeng, Shuran Song, Stefan Welker, Johnny Lee, Alberto Rodriguez, and Thomas A. Funkhouser. Learning synergies between pushing and grasping with self-supervised deep reinforcement learning. *CoRR*, abs/1803.09956, 2018.

Marvin Zhang, Sharad Vikram, Laura Smith, Pieter Abbeel, Matthew J. Johnson, and Sergey Levine. Solar: Deep structured latent representations for model-based reinforcement learning. *ArXiv*, abs/1808.09105, 2018.

A Related Work

Recent years have seen impressive results from reinforcement learning Sutton & Barto (2018) applied to challenging problems such as video games Mnih et al. (2015); OpenAI, Go Silver et al. (2016), and robotics Levine et al. (2015); OpenAI et al. (2018); Kalashnikov et al. (2018). However, the dependence on large quantities of labeled data can limit the applicability of these methods in the real world. One approach is to leverage self-supervision, where an agent only uses data that it can collect autonomously.

Self-Supervised Reinforcement Learning: Self-supervised reinforcement learning explores how RL can leverage data which the agent can collect autonomously to learn meaningful behaviors, without dependence on task specific reward labels, with promising results on tasks such as robotic grasping and object re-positioning Pinto & Gupta (2015); Ebert et al. (2018); Zeng et al. (2018). One approach to self-supervised RL has been combining goal-conditioned policy learning (Kaelbling, 1993; Schaul et al., 2015; Codevilla et al., 2017) with goal re-labeling Andrychowicz et al. (2017) or sampling goals Nair et al. (2018; 2019). While there are numerous ways to leverage self-supervised data, ranging from learning distance metrics Yu et al. (2019b); Hartikainen et al. (2019), generative models over future states Kurutach et al. (2018); Fang et al. (2019); Eysenbach et al. (2019); Liu et al. (2020); Nair & Finn (2020), and representations Veerapaneni et al. (2019), one of the most heavily utilized techniques is learning the dynamics of the environment Watter et al. (2015); Finn & Levine (2016); Agrawal et al. (2016).

Model-Based Reinforcement Learning: Learning a model of the dynamics of the environment and using it to complete tasks has been a well studied approach to solving reinforcement learning problems, either through planning with the model Deisenroth & Rasmussen (2011); Watter et al. (2015); McAllister & Rasmussen (2016); Banijamali et al. (2017); Chua et al. (2018); Amos et al. (2018a); Hafner et al. (2018); Nagabandi et al. (2019) or optimizing a policy in the model Racanière et al. (2017); Ha & Schmidhuber (2018); Kaiser et al. (2019); Lee et al. (2019); Janner et al. (2019); Wang & Ba (2019); Hafner et al. (2019); Gregor et al. (2019); Byravan et al. (2019). Numerous works have explored how these methods might leverage deep neural networks to extend to high dimensional problem settings, such as images. One technique has been to learn large video prediction models Finn & Levine (2016); Babaeizadeh et al. (2017); Ebert et al. (2017; 2018); Paxton et al. (2018); Lee et al. (2018); Villegas et al. (2019); Xie et al. (2019), however model under-fitting remains an issue for these approaches Dasari et al. (2019). Similarly, many works have explored learning low dimensional latent representations of high dimensional states Watter et al. (2015); Dosovitskiy & Koltun (2016); Zhang et al. (2018); Hafner et al. (2018); Kurutach et al. (2018); Ichter & Pavone (2018); Wang et al. (2019); Lee et al. (2019); Gelada et al. (2019) and learning the dynamics in the latent space. Unlike these works, we aim to make the problem easier by encouraging the network to predict only task-relevant quantities, while also changing the objective, and hence the distribution of prediction errors, in a task-driven way. This allows the prediction problem to be more directly connected to the downstream use-case of task-driven planning.

Addressing Model Errors: Other works have also studied the problem of model error and exploitation. Approaches such as ensembles Chua et al. (2018); Thananjeyan et al. (2019) have been leveraged to measure uncertainty in model predictions. Similarly, Janner et al. (2019) explore only leveraging the learned model over finite horizons where it has accurate predictions and Levine et al. (2015) leverage local models. Exploration techniques like Pathak et al. (2017) can also be leveraged to iteratively collect more data where the model is uncertain.

Most similar to our proposed approach are techniques which explicitly change the models objective to optimize for performance on downstream tasks. Schrittwieser et al. (2019); Havens et al. (2020) explore only predicting future reward to learn a latent space in which they learn dynamics, Freeman et al. (2019) learn a model with the objective of having a policy achieve high reward from training in it, and Amos et al. (2018b); Srinivas et al. (2018) embed a model/planner inside a neural network. Similarly, Farahmand et al. (2017); D'Oro et al. (2019) explore how model training can be re-weighted using value functions or policy gradients to emphasize task specific performance. Unlike these works, which depend heavily on task-specific supervision, our proposed approach can be learned on purely self-supervised data, and generalize to unseen tasks.

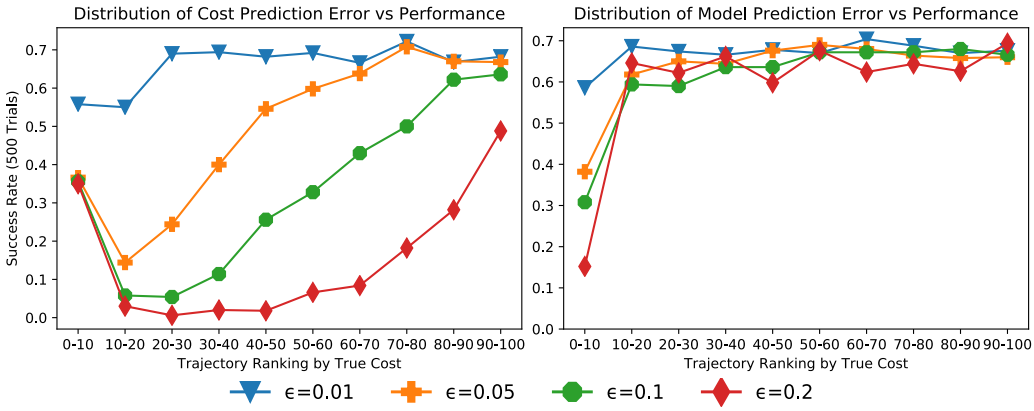

Figure 3: **Distribution of model errors vs. performance**: We validate how the distribution of model errors affects performance on a simple 2D navigation domain, by adding noise to cost predictions **(left)** or model predictions **(right)**. We add varying amounts of noise with magnitude up to ϵ to the predictions of the 10 lowest true cost trajectories **(0-10)** to the 10 highest true cost trajectories **(90-100)**. We observe that adding noise to low true cost trajectories dramatically reduces performance, while adding noise to the high true cost trajectories has no nearly no impact on performance.

B Further Analysis

B.1 Theorem 3.1 Proof

Proof. Consider any action sequence $a^i_{1:T}$. It follows from the bound in Equation 2 that in the worst case, $\hat{c_i} > c^*_i - (c^*_i - c^*_1) + \epsilon$. Similarly it follows from the bound in Equation 1 that in the worst case $\hat{c_1} < c^*_1 + \epsilon$. Substituting, we clearly see that

$$\hat{c_i} > c^*_i - (c^*_i - c^*_1) + \epsilon = c^*_1 + \epsilon > \hat{c_1} \tag{3}$$

Hence $\hat{c_i} > \hat{c_1}$ for any $i > 1$, and thus $a^1_{1:T}$ will correctly remain the lowest cost trajectory, and the policy will still select $a^1_{1:T} = \min_{a^i_{1:T}} \hat{c}_i$.

Furthermore, if Equations 1 or 2 do not hold, then there is a non-zero probability that $\hat{c_i} < c^*_i - (c^*_i - c^*_1) + \epsilon = c^*_1 + \epsilon$ or a non-zero probability that $\hat{c_1} > c^*_1 + \epsilon$, and thus a non-zero probability that $\hat{c_i} < \hat{c_1}$, meaning that the policy would not select $a^1_{1:T}$ and would be sub-optimal. □

B.2 Verifying Theorem 3.1 Experimentally

We now verify Theorem 3.1 through a controlled study of how prediction error affects task performance. To do so, we will use the true model of an environment and true cost of an action sequence for planning, but will artificially add noise to the cost/model predictions to generate model error.

Consider a 2 dimensional navigation task, where the agent is initialized at $s_0 = [0.5, 0.5]$ and is randomly assigned a goal $s_g \in [0, 1]^2$. Assume we have access to the underlying model of the environment, and cost defined as $\mathcal{C}(s_t, s_g) = ||s_t - s_g||_2$. We can run the policy described in Section 3.2, specifically sampling $N = 100$ action sequences, and selecting the one with lowest predicted cost, where we consider 2 cases: (1) predicted cost is using the true model, but with noise ε added to the true cost $\hat{c_i} = c^*_i + \varepsilon$ of some subset of action sequences, and (2) predicted cost is true cost, but with noise ε added to the model predictions $s_{t+1} = \bar{s}_{t+1} + \varepsilon$ where $\bar{s}_{t+1} \sim p(s_{t+1}|s_t, a_t)$ of some subset of action sequences. The first case relates directly to Theorem 3.1, while the second case relates to what we can control when training a self-supervised dynamics model. When selectively adding noise, we will use uniform noise $\varepsilon \sim \mathcal{U}(-\epsilon, \epsilon)$.

We specifically study the difference in task performance when adding noise ε to model predictions for the first 10% of trajectories with lowest true cost, the second 10% lowest true cost trajectories, etc., up to the 10% of trajectories with highest true cost. Here "true cost" refers to the cost of the action sequence under the true model and cost function without noise. For each noise augmented model we measure the task performance, specifically the success rate (reaching within 0.1 of the goal), over 500 random trials.

We see in Figure 3 that for multiple values of noise ϵ, when adding noise to the better (lower true cost) trajectories we see a significant drop in task performance, while when adding noise to the worse

(higher true cost) trajectories task performance remains relatively unchanged (except for the case with very large ϵ). In particular, we notice that when adding noise to cost predictions, performance scales almost linearly as we add noise to worse trajectories. Note there is one exception to this trend: if we add noise only to the top 10% of trajectories, performance is not optimal, but reasonable because the best few trajectories will occasionally be assigned a lower cost under the noise model.

In the case of model error, we see a much steeper increase in performance, where adding model error to the best 10 trajectories significantly hurts performance, while adding to the others does not. This is because, in this environment, noise added to model predictions generally makes the cost of those predictions worse; so if no noise is added to the best trajectories, the best action sequence is still likely to be selected. The exact relationship between model prediction error and cost prediction error depends on the domain and task. But, we can see that in both cases in Figure 3, the conclusion from Theorem 3.1 holds true: accuracy on good action sequences matters much more than accuracy on bad action sequences.

C Additional Results

C.1 Experimental Domains and Comparisons

Experimental Domains: Our primary experimental domain is a simulated tabletop manipulation task built off of the Meta-World Yu et al. (2019a) suite of environments. Specifically, it consists of a simulated Sawyer robot, and 3 blocks on a tabletop. In the *self-supervised* data collection phase, the agent executes a random policy for 2,000 episodes, collecting 100,000 frames worth of data. Then, after learning a model, the agent is tested on 2 previously unseen tasks, where the task is specified by a goal image. **Task 1** consists of pushing the green, pink, or blue block to a goal position, while the more challenging **Task 2** requires the robot to push 2 blocks to each of their respective goal positions (see Figure 4), where task success is defined as being within 0.1 of the goal positions. **Task 3/4** involve closing/opening the door respectively with distractor objects on the table, where success is defined as being within $\pi/6$ of the goal position. The agent receives $64\times64\times3$ RGB camera observations of the tabletop. We also study our methods model error on real robot data from the RoboNet Dasari et al. (2019) dataset, particularly on the subset of data from the Sawyer robot in Section C.2.

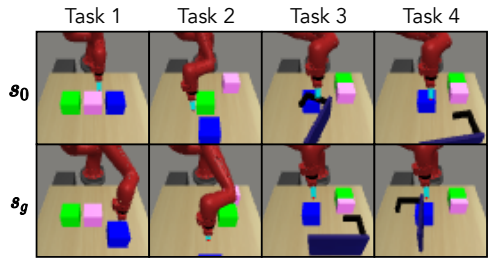

Figure 4: **Tasks:** Sample initial & goal states for tasks.

Comparisons: We compare to several model variants in our experiments. **GAP-Residual (Ours)** is our approach of learning dynamics in a latent space conditioned on the current state and goal, and reconstructing the residual between the current state and goal state, as described in Section 3.3. **GAP-Weighted** refers to a standard latent dynamics model which is also conditioned on the goal and re-weights the reconstruction loss by $1 - \bar{C}(s_t, s_g)$ where $\bar{C}$ is $\mathcal{C}$ normalized to be from [0,1]. This explicitly encourages more accurate predictions on states closer to the goal. **Residual Model** is similar to our approach (GAP-Residual), except instead of conditioning on the goal and predicting the residual to the goal, it is conditioned on the current state, and predicts the residual to the current state. This is reflective of prior works (e.g. Nagabandi et al. (2019)) that predict residuals for model-based RL. **Goal-Conditioned Prediction (GCP)** amounts to a standard dynamics model, but is also conditioned on the goal, similar to prior work on goal-conditioned video prediction Rybkin et al. (2020). **Standard** refers to a standard latent dynamics model, representative of approaches such as PlaNet Hafner et al. (2018), but without reward prediction since we are in the self-supervised setting.

When studying task performance, we also compare to two alternative self-supervised reinforcement learning approaches. First, we compare to **Action Reconstruction**, which is a latent dynamics model where the latent space is learned via a behavior cloning loss (instead of image reconstruction), as is done in Pathak et al. (2017) and reinforcement learning with imagined goals (**RIG**) Nair et al. (2018), where we train a VAE on the same pre-collected dataset as the other models, then train a model-free policy in the latent space of the VAE to reach goals sampled from the VAE. Further implementation details can be found in Appendix D.

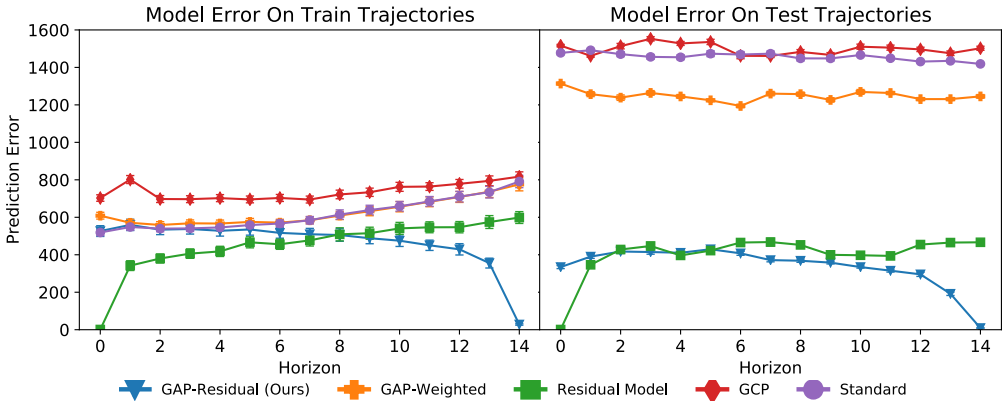

Figure 5: **Distribution of Model Errors (RoboNet):** We examine the distribution of model prediction errors of our proposed method (GAP) compared to baselines over 100 random trajectories from the RoboNet Dasari et al. (2019) train dataset **(left)** and test dataset **(right)**. All y-axis are from [0, 1600] corresponding to model's mean squared error (with standard error bars), and the x-axis corresponds to number of time steps predicted. The goal in all cases is the relabeled state at timestep 15, and thus all action sequences are goal reaching. Interestingly, we observe in both training and test datasets that our GAP-Residual approach leads to much lower model error toward the end of the planning horizon, as we approach the goal. This again suggests that GAP-Residual successfully re-distributes model error to be low in states which matter for the task. Additionally, we see in the test data that both our GAP-Residual and Residual Model achieve much lower test error, suggesting that trying to model all components of the scene makes the model more prone to overfitting.

C.2 Experiment 3: Does GAP scale to real, cluttered visual scenes?

Lastly, we study whether our proposed GAP method extends to real, cluttered visual scenes. To do so we evaluate on a subset of the RoboNet Dasari et al. (2019) dataset, specifically data which contains a front-facing view of a Sawyer robot. We train all models on 100,000 frames of data, and train to convergence for 750,000 iterations. We sample sequences of length 15, and re-label the final state in the trajectory as the goal.

We evaluate the distribution of model errors of the different approaches across the planning horizon in Figure 5. First, we observe that our proposed GAP-Residual approach does indeed redistribute model errors to be low in states near the goal, as seen from the dip in prediction error approaching zero as the horizon approaches 15. Intuitively, this is expected since in GAP-Residual the prediction task becomes easier as states approach the goal. Second, we see that for both GAP-Residual and the Residual Model, test error stays roughly the same, while for the other approaches test error is significantly higher. This suggests that modeling the delta (effectively only the relevant parts of the scene), makes the model less prone to overfitting compared to approaches that must model the entire scene. Hence, GAP is able to re-distribute model error, even on challenging visual scenes.

D Method Implementation Details

In this section we go over implementation details for our method as well as our comparisons.

D.1 Implementing Goal-Aware Prediction

Data Collection and Model Training: In our *self-supervised* setting, data collection simply corresponds to rolling out a random exploration policy in the environment. Specifically, we sample uniformly from the agent's action space, and collect 2,000 episodes, each of length 50, for a total of 100,000 frames of data.

During training, sub-trajectories of length 30 time steps are sampled from the data set, with the last timestep labeled as the goal $s_g = s_{30}$ Depending on the current value of H, loss is computed over H step predictions starting from states $s_{t:(t+H-1)}$. We use a curriculum when training all models, where H starts at 0, and is incremented by 1 every 50,000 training iterations. All models are trained to convergence, for about $300,000$ iterations on the same data set.

Planning with GAP: For all trained models, when given a new goal at test time s_g, we plan using model predictive control (MPC) in the latent space of the model. Specifically, both the current state s_t and s_g are encoded into their respective latent spaces z_t and z_g (Algorithm 1, Line 3). Then using

the model $f_{dyn}(z_{t+1}|z_t, a_t)$, the agent plans a sequence of H actions to minimize cost $||z_g - z_{t+H}||_2^2$ (Algorithm 1, Lines 4-11). Following prior works Finn & Levine (2016); Hafner et al. (2018), we use the cross-entropy method Rubinstein & Kroese (2004) as the planning optimizer. Finally, the best sequence of actions is returned and executed in the environment (Algorithm 1, Line 13).

Algorithm 1 Latent MPC($f_{enc}, f_{dyn}, s_t, s_g$)

1: Let $D = 1000, D^* = 10, H = 15$
2: Receive current state s_t and goal state s_g
3: Encode $z_t \sim f_{enc}(s_t, s_g), z_g \sim f_{enc}(s_g, s_g)$
4: Initialize $N(\mu, \sigma^2) = \mathcal{N}(0, 1)$
5: Let cost function $C(z_i, z_j) = ||z_i - z_j||_2^2$
6: **while** iterations ≤ 3 **do**
7: $\quad a^1_{t:H}, ..., a^D_{t:H} \sim N(\mu, \sigma^2)$
8: $\quad z^1_{t+1:t+H}, ..., z^D_{t+1:t+H} \sim f_{dyn}(z_t, a^1_{t:H}, ..., a^D_{t:H})$
9: $\quad \hat{c}^1, ..., \hat{c}^D = [\sum_{h=1}^{H} \mathcal{C}(z^1_{t+h}, z_g), ..., \sum_{h=1}^{H} \mathcal{C}(z^D_{t+h}, z_g)]$
10: $\quad a_{sorted} = Sort([a^1_{t:H}, ..., a^D_{t:H}])$ by $\hat{c}$
11: $\quad$ Refit μ, σ^2 to $a_{sorted}[1 - D^*]$
12: **end while**
13: Return $\hat{c}_{sorted}[1]$, $a_{sorted}[1]$

While executing the plan, our model re-plans every H timesteps. That is, it starts at state s_t, uses Latent MPC (Algorithm 1) to first plan a sequence of H actions, executes them in the environment resulting in a state s_{t+H}, then re-plans an additional H actions, and executes them resulting in a final state s_T. Success is computed based the difference between s_T and s_g.

D.2 Architecture Details

Block Pushing Domain: All comparisons leverage a nearly identical architecture, and are trained on an Nvidia 2080 RTX. In the block pushing domain input observations are [64,64, 6] in the case of our model (GAP-Residual), as well as GAP-Weighted, Residual Model, Goal-Conditioned Prediction, and [64,64, 3] in the case of Standard.

All use an encoder f_{enc} with convolutional layers (channels, kernel size, stride): [(32, 4, 2), (32, 3,1), (64, 4, 2), (64, 3,1), (128, 4, 2), (128, 3,1), (256, 4, 2), (256, 3,1)] followed by fully connected layers of size [512, 2 $\times L$] where L is the size of the latent space (mean and variance). All layers except the final are followed by ReLU activation.

The decoder f_{dec} takes a sample from the latent space of size L, then is fed through fully connected layers [128, 128, 128], followed by de-convolutional layers (channels, kernel size, stride): [(128, 5, 2), (64, 5, 2), (32, 6, 2), (3, 6,2)]. All layers except the final are followed bu ReLu activation, except the last layer which is a Sigmoid in the case of Standard, Goal-Conditioned Prediction, and GAP-Weighted, and Tanh in the case of GAP-Residual and Residual Model.

For all models the dynamics model f_{dyn} are a fully connected network with layers [128, 128, 128, L], followed by ReLU activation except the final layer.

The action reconstruction baseline utilizes the same f_{enc} and f_{dyn} as above, but f_{dec} is instead a fully connected network of size [128, 128, action size] where action size is 4 (corresponding to delta x,y, z motion and gripper control). All layers except the final are followed by ReLU activation.

Lastly, the RIG Nair et al. (2018) baseline uses a VAE with identical f_{enc} and f_{dec} to the standard approach above, except learns a policy in the latent space. The policy architecture used is the default SAC Haarnoja et al. (2018) from RLkit, namely 2 layer MLPs of size 256.

RoboNet: In the Robonet domain, the architecture is similar to the above, except the encoder and decoder network are larger.

Specifically, f_{env} has convolutional layers: [(32, 4, 2), (32, 3,1), (32, 3,1), (64, 4, 2), (64, 3,1), (64, 3,1), (128, 4, 2), (128, 3,1), (128, 3,1) (256, 4, 2), (256, 3,1), (256, 3,1)] followed by fully connected layers of size [1024, 2 $\times L$]

The decoder network has fully connected layers [1024, 512, 128] followed by de-convolutional layers: [(128, 5, 2), (128, 3,1), (128, 3,1), (64, 5, 2), (64, 3,1), (64, 3,1), (32, 6, 2), (32, 3,1), (32, 3,1) (3, 6, 2), (3, 3,1), (3, 3,1)]

D.3 Training Details

Block Pushing Domain: We collect a dataset of 2,000 trajectories, each 50 timesteps with a random policy. All models are trained on this dataset to convergence for roughly 300,000 iterations. All models are trained with learning rate of 1e-4, and batch size 32.

The RIG baseline is trained using the default SAC example parameters in RLkit, for an additional 3 million steps.

RoboNet: We collect a dataset of 100,000 timesteps pulled from the subset of RoboNet Dasari et al. (2019) consisting of the Sawyer data. All models are trained on this dataset to convergence for 750,000 iterations. All models use learning rate of 1e-5, and batch size 16.