Extended Abstract

Motivation Generative modeling in machine learning is a hot topic which only in the past couple years has seen application in offline reinforcement learning. Even more recent is the development of the conditional flow matching technique, which offers order of magnitude computational speed up compared to the well known diffusion technique. This paper aims to use these techniques to build a planning agent that anticipates how much reward will be gained by visiting a sequence of states. By explicitly modeling these rewards, this technique build upon prior flow matching planners that require post-training tuning to achieve optimal returns. This works develops, engages, and evaluates this technique called generative planning with reward forecasting.

Method Flow matching lays at the heart of methodology. This technique is used to generate a sequence of states, starting with the agents current state. An inverse dynamics model then reviews the first two states of this generated plan and answers the question: "What action is necessary to reach the next state?"

Flow matching is used to learn a mapping from random noise to examples seen in the offline dataset. This then allows for the agent to sample random noise at test time, and iteratively massage that noise into a plan. Guided flow matching build upon this by adding a conditioning vector informing how a noisy sample relates to the desired plan. The reward forecasting technique conditions on state in order to provide smooth plans grounded in the agent's current state.

Implementation Generative planning with reward forecasting is implemented using a 2D U-Net. This architecture includes residual connections connecting repeated convolutional blocks, which themselves have residual connections. Most importantly, these blocks contain dedicated multi-layer perceptrons for interpretting where in the noise-to-plan interpolation process the current data is, and how the conditioner impacts the flow matching process. The flow matching planner is run iteratively in a process known as simulation to produce a plan from noise. Once the plan is produced a simple multilayer perceptron is used to model the inverse dynamics.

Results The reward forecaster was tested against other flow matching planners utilizing alternative conditioning schemes but only predicting sequences of states. Tests were conducted in the gymnasium hopper environment to measure agent return within two different offline datasets. While the reward forecaster produces visually convincing plans, the generated state sequence is out of distribution from the environments natural dynamics. Because of this, the inverse dynamics model cannot resolve an appropriate action for the agent, and poor performance ensues for all tested models.

Discussion This poor performance does not match literature. Even the baseline does not match the original author's reported values. This is likely due to differences in the flow matching architecture used in this work and prior literature.

Conclusion While it has not been proven whether reward forecasting truly improves upon conditioned state-sequence planners, future opportunity for work have been identified. Impacts on generative architecture need to be measured in the contect of offline reinforcement learning. More impactful would be a method for constraining the output actions of a generative planning agent.

Generative Planning: Conditioning Vs Reward Forecasting

Jon Frydman
Department of Computer Science
Stanford University
jonfryd@stanford.edu

Abstract

This paper introduces generative planning with reward forecasting. This technique utilizes a generative modeling framework to perform offline reinforcement learning (RL). The resulting RL agent is capable of stitching together learned state sequences and corresponding rewards when generating a plan. The technique is tested against state-sequence planners implicitly conditioned by reward or otherwise. Tests show all agents perform poorly in the hopper environment, regardless of dataset. This likely stems from model architecture deficiencies which lead to out-of-distribution generated state sequences.

1 Introduction

Recent advancements in offline reinforcement learning (RL) methodology include applications of generative modeling techniques towards model-based planning. While classical planning leverages a dynamics model to auto-regressively synthesize a sequence of states, generative planning organizes a plan holistically. Conditional generative planning produces plans which meet arbitrary user-specified criteria. These plans can be conditioned to maximize return. This approach avoids common place issues in classical planning such as error compounding, where inaccuracies in the dynamics model compound over time, and actor-critic coupling, where actors maximize the value of a Q-function, which in-turn assesses the performance of those actions. Conditional generative planning instead offers an error-distributed, uncoupled method for producing RL agents capable of maximizing returns and achieving state-of-the-art (SOTA) performance.

Generative modeling often occurs with a trade off: expensive computation. Diffusion based planning methods, often require hundreds of denoising steps in order to produce a synthetic trajectory. Guided flow matching based planning methods offer an alternative to diffusion, achieving the same performance with a 10x speeds up, due to more efficient probability path modeling (Zheng et al., 2023). Flow matching in RL has not been explored much however. Questions remain on why conditioning on suboptimal returns leads to SOTA performance. Furthermore the simulation-free training aspects of flow matching leads to different considerations needed when plan with conditioning signals.

How can we employ flow matching based planners such that they perform inline with their guidance objectives? This work aims to solve this question by explicitly embedding the return maximization objective into the planning process. This idea is based in the general notion of extending model-based RL from dynamics only to dynamics + reward prediction. A generative planning model with full built-in world modeling capabilities contains the information needed to stitch states and reward into a cohesive, target plan. By partially framing a plan's accumulated reward as a goal-conditioned in-painting problem, a generative planner can determine what states must be visited to achieve a desired return. An input-output consistent planner based on guided flow matching allows for a computationally practical method for offline RL that retains the high-expressive capabilities of generative planners.

Stanford CS224R 2025 Final Report

2 Related Work

2.1 Generative Offline Reinforcement Learning

Due to the success of generative modeling in machine learning, many researchers have rushed to apply these techniques to offline reinforcement learning. Catagories of such applications include policy extraction, dynamics prediction, and planning techniques. Generative planning with reward forecasting falls in the last category. The technique differs from generative policy extraction as the actor is not coupled to a Q-function (Park et al., 2025)(Wang et al., 2023). Additionally dynamics and reward modeling are learned implicitly in the planning model, as opposed to explicitly through the use of a generative world model Ding et al. (2024).

2.2 Generative Planning

Generative planning lies at the heart of this work. Traditional dynamics modeling planners fall within state-space planning framework described by Sutton and Barto (2018), whereas generative planners fall within the plan-space optimization framework. The transformer-based Decision Transformer (Chen et al., 2021) is unlike the technique proposed in this paper, because this generative model produces plans auto-regressively. Instead this proposed technique is more similar to diffusion-based methods like Diffuser (Janner et al., 2022) and Decision Diffuser (Ajay et al., 2023). The use of diffusion results in a array of subtle differences from a flow matching based planner. For example, flow matching allows for simulation-free training and the flow matching loss differs from the diffusion reconstruction loss. Furthermore, Diffuser plans by synthesizing trajectories which include both states and actions. Decision Diffuser only plans for states, similarly to the proposed. Continuity of plans and current state for these two methods are achieved differently from each other and the reward forecaster. In Diffuser, continuity arises from use of a 1D U-Net and pinning of the current state into the plan. In Decision Diffuser the 1D U-Net is used as well as a fixed context window of multiple states in the plan. The reward forecaster achieves continuity with the same first state pinning as Diffuser but also conditions a 2D U-Net with the current state to ensure continuity.

Of generative planning framworks, the reward forecaster most closely resembles the guided flow matching planner of Zheng et al. (2023), outside the use of a 1D U-Net. The guided flow methodology, adapts classifier-free guidance (Ho and Salimans, 2022) for flow matching. This work uses the same technique when implementing and testing a variety of classifiers for comparative testing against the reward forecaster.

This work also contrasts from other SOTA models like π_0 Black et al. (2024) which integrates a flow matching planner on top of a pre-trained vision-language model. This difference mostly stems from the use-case. π_0 aims to be a generalist multi-task RL agent, whereas the reward forecaster aims to specialize for an individual task. Furthermore, π_0 predicts action sequences compared to the reward forecaster's state and accumulated return sequences.

3 Method

The generative planner with reward forecasting comprises two main components: a planner $\hat{\tau}_\theta$ and an inverse dynamics model $\hat{a}_\phi$. Together these blur the lines between model-based RL and policy gradient, effectively forming a model-based policy technique. Details for both training and sampling/ acting can be found in algorithm 1 and algorithm 2 respectively.

3.1 Guided Flow Matching

All generative planners tested in this work are based in conditional flow matching (Lipman et al., 2024), which maps an arbitrary probability distribution to the distribution of a given dataset. Here, a multivariate normal distribution is mapped to the offline RL dataset through a learned diffeomorphism, embedded in probability-time space. A step forward in flow time indicates a partial traversal in this probability mapping according to the learned diffeomorphism, also known as the flow. As a point of clarity, the flow time t_{flow} is distinct from the episode time t, which is considered as a component of the probability space in the flow matching framework. Flow time ranges from zero to unity. At each end, $x_0 \sim \mathcal{N}(0, I)$ represents a sample from noise and $x_1 = \tau^{[i]}$ a sample trajectory from the dataset.

Because conditional flow matching operates by interpolating individual points between the source and target probability distributions, an interpolation scheme must be employed. This project uses simple linear interpolation to define the interpolated point $x_{t_{\text{flow}}}$ and the flow at that point $\frac{dx}{dt_{\text{flow}}}$.

$$x_{t_{\text{flow}}} = (1 - t_{\text{flow}})x_0 + t_{\text{flow}}\tau^{[i]}$$

$$\frac{dx}{dt_{\text{flow}}} = \tau^{[i]} - x_0$$

The aim of conditional flow matching is to minimize the difference between the predicted flow $\hat{\tau}_\theta(x_{t_{\text{flow}}}, t_{\text{flow}})$ and the prescribed flow $\frac{dx}{dt_{\text{flow}}}$.

Guided conditional flow matching (Zheng et al., 2023) builds upon conditional flow matching by incorporating classifier-free guidance (Ho and Salimans, 2022) into the learning and sampling processes. Guidance augments the base flow predictor such that a random sample x_0 accompanied with a label y maps to some datapoint $\tau^{[i]}$ associated with y. For the model $\hat{\tau}_\theta$ to incorporate guidance, it must be capable of both unguided flow prediction, with null label $\emptyset$, and guided prediction, with label y. This is accomplished by assigning a hyperparameter $p_{\text{unconditioned}}$ which states the probability of training with or without the label. Given a coin toss where $\xi \sim \text{Bernoulli}(p_{\text{unconditioned}})$, the guided conditional flow matching objective is:

$$\mathcal{L}_{GCFM} = \begin{cases} \left\| \hat{\tau}_\theta(x_{t_{\text{flow}}}, t_{\text{flow}}, \emptyset) - \frac{dx}{dt_{\text{flow}}} \right\|^2 & \text{if } \xi \\ \left\| \hat{\tau}_\theta(x_{t_{\text{flow}}}, t_{\text{flow}}, y) - \frac{dx}{dt_{\text{flow}}} \right\|^2 & \text{otherwise} \end{cases}$$

This loss can be seen incorporated in to the training algorithm on line 16 of algorithm 1. There, the horizon limited partial trajectories are the datapoints. Targets and flow points are formed by randomly sampling time from a uniform distribution. The grouping of $\{x_t, t, y, \frac{dx_t}{dt}\}$ form an entry in a batch $\mathcal{B}$, and the total loss is the mean loss over $\mathcal{B}$.

Algorithm 1 Training the Generative Planner with Reward Forecasting

1: **Given** H, $p_{\text{unconditioned}}$, α, $\mathcal{D}$,
2: Initialize $\hat{\tau}_\theta$ with (H, w_{guidance}) and initialize $\hat{a}_\phi$
3: **while** not done **do**
4: **for** $i \in [0, |\mathcal{D}|)$ **do** ▷ Iterate over episodes
5: **for** $j \in \left\lceil \frac{|\tau^{[i]}|}{H} \right\rceil$ **do** ▷ Construct horizon limited partial trajectories
6: $x_0 \sim \mathcal{N}(0, I)$
7: $t \sim \mathcal{U}[0, 1]$
8: $\xi \sim \text{Bernoulli}[p_{\text{unconditioned}}]$
9: $AR = \sum_{t'=0}^{k} \gamma^{t'} r_{jH+t'}^{[i]}$ ▷ $k \in [0, H-1]$
10: $\tau_j^{[i]} \leftarrow \{o_{jH:(j+1)H}^{[i]}, AR_{jH:(j+1)H}\}$ ▷ Partial trajectory with accumulated reward
11: $x_t = (1-t)x_0 + t\tau_j^{[i]}$
12: $\frac{dx_t}{dt} = \tau_j^{[i]} - x_0$
13: $y = o_{jH}^{[i]}$ **if** $\xi = 0$ **else** $\emptyset$
14: $\mathcal{B}_j^{[i]} \leftarrow \{x_t, t, y, \frac{dx_t}{dt}\}$ ▷ The flow matching arguments from partial trajectory j
15: **end for**
16: $\theta \leftarrow \theta - \alpha \mathbb{E}_{(x_t, t, y, \frac{dx_t}{dt}) \sim \mathcal{B}^{[i]}} \left[\| \hat{\tau}_\theta(x_t, t, y) - \frac{dx_t}{dt} \|^2 \right]$
17: $\phi \leftarrow \phi - \alpha \mathbb{E}_{(o,a,o') \sim \tau^{[i]}} \left[\| \hat{a}_\phi(o, o') - a \|^2 \right]$
18: **end for**
19: **end while**

Sampling with a guided flow matching model is equivalent to traversing the learned flow mapping from starting noise and a given label. Analogously this can also be seen as integrating through time on the guided flow, $\tilde{\tau}_\theta$.

$$\tilde{\tau}_\theta(x_{t_{\text{flow}}}, t_{\text{flow}}, y) = (1 - \omega_{\text{guidance}})\hat{\tau}(x_{t_{\text{flow}}}, t_{\text{flow}}, y) - \omega_{\text{guidance}}\hat{\tau}(x_{t_{\text{flow}}}, t_{\text{flow}}, \emptyset)$$

A guidance weight of $\omega_{\text{guidance}} = 2$ was used throughout this project.

A first-order integration step through this guided flow can be expressed as follows:

$$x_{t_{\text{flow}}+dt} = x_t + dt\tilde{\tau}_\theta(x_{t_{\text{flow}}}, t_{\text{flow}}, y)$$

Algorithm 2 shows this procedure using second-order midpoint integration steps. The process of traversing the entire flow often through repeated forward stepping is referred to as simulation. For notation's sake the simulation operation is denoted as $\tilde{\mathcal{T}}_\theta^n$, where n indicates the number of integration steps used. Because flow time and random sampling are internal to simulation, $\tilde{\mathcal{T}}_\theta^n$ operates only on the reward forecasting inputs.

Algorithm 2 Acting with the Generative Planner with Reward Forecasting

Given $\hat{\tau}_\theta, w_{\text{guidance}}, n_{\text{steps}}, o_0, AR_{\text{desired}}$
$dt \leftarrow \frac{1}{n_{\text{steps}}}$
$x_0 \sim \mathcal{N}(0, I)$
$z \leftarrow x_0$
$z_{0,\dim o} \leftarrow o_0$ ▷ In-paint current state
$z_{0,\dim AR} \leftarrow 0$ ▷ In-paint return-so-far
$z_{H-1,\dim AR} \leftarrow AR_{\text{desired}}$ ▷ In-paint desired accumulated return
for $i \in [1, n_{\text{steps}}]$ **do** ▷ Solve ODE with constrained midpoint method
    $m \leftarrow z + \frac{dt}{2}\left((1 - w_{\text{guidance}})\hat{\tau}_\theta(z, t, o_0) - w_{\text{guidance}}\hat{\tau}_\theta(z, t, \emptyset)\right)$
    $z \leftarrow z + \frac{dt}{2}\left((1 - w_{\text{guidance}})\hat{\tau}_\theta(m, t + \frac{dt}{2}, o_0) - w_{\text{guidance}}\hat{\tau}_\theta(m, t + \frac{dt}{2}, \emptyset)\right)$
    $z_{0,\dim o} \leftarrow o_0$
    $z_{0,\dim AR} \leftarrow 0$
    $z_{H-1,\dim AR} \leftarrow AR_{\text{desired}}$
end for
plan $\leftarrow (z_{0:H-1}, \dim o)$
return $\hat{a}_\phi(\text{plan}_0, \text{plan}_1)$

3.2 Reward Forecasting

The planning model achieves reward forecasting by predicting both state sequences and expected future rewards. Rather than explicitly predicting rewards associated with each transition, the planner predicts the reward plus the achieved return prior to reaching that state. This accumulated reward AR closely resembles the finite horizon discounted return, but is limited from the starting point as well:

$$AR_{t:t+k} = \sum_{t'=t}^{t+k} \gamma^{t'-t}\hat{r}_{t'}$$

Accumulated reward therefore measures progress along a state-sequence plan and forms a well-posed goal-conditioning objective.

Goal-conditioning is achieved through in-painting, where the starting conditions and end return of a fixed horizon window are pinned at either end of a state + AR sequence. The guided conditional flow model then iteratively fills in the gaps between with synthetic transitions, noted by their hat symbols.

$$\tilde{\mathcal{T}}_\theta^n(o_t, AR_{t:t+H}) \rightarrow \begin{bmatrix} \mathbf{o}_t & \hat{o}_{t+1} & \hat{o}_{t+2} & \dots & \hat{o}_{t+H-1} & \hat{o}_{t+H} \\ \mathbf{0} & \hat{AR}_{t:t+1} & \hat{AR}_{t:t+2} & \dots & \hat{AR}_{t:t+H-1} & \mathbf{AR}_{t:t+H} \end{bmatrix}$$

This setup allows for the reward forecaster to stitch together generated transitions that will achieve the desired horizon limited accumulated reward $\mathbf{AR}_{t:t+H}$. During testing, the max horizon limited return is used as the desired accumulated reward in order to maximize return, in-line with the typical RL objective.

Baseline planning models without forecasting only generate states given the starting observation. As such these reference models plan are not tied to return a specific desired return.

3.3 State Conditioning and More

State conditioning is a vital aspect in the generative planning process. Without it, generated synthetic trajectories do not naturally include the current state of the agent. Even with the current state iteratively assigned to the start of the plan (line 11 in algorithm 1), a natural discontinuity occurs, where the agent expects to be in a very different state. In order to ensure plans are grounded in reality, the current state is also used as the guidance signal during the simulation process. This conditioning allows for smooth plans starting with the agent's current state.

As shown in Zheng et al. (2023), a flow matching planner, conditioned on state and return can achieve SOTA performance, however the specifics of that conditioning are unclear. This work uses two alternative formulations state + return as relative baselines. In the former, the state and return are concatenated to form a single conditioning signal. In the latter, the states and return are treated as separate signals, with both being assigned the null conditioner during training depending on coin toss ξ. The difference between these two stems from the assignment of a multi-layer perceptron (MLP) for interpreting each conditioning signal in the planner's internal architecture. With two signals, two MLPs are used in the latter.

A third form was also initially tested. Here two separate conditioners were used with only the return conditioning was affected by ξ. The resulting plans were highly chaotic in nature however, and this was not pursued further. As a opportunity for future research, it may be worthwhile to explore multi-conditioner guidance where ξ is drawn from a multinomial distribution instead of a Bernoulli distribution.

3.4 Inverse Dynamics

An inverse dynamics model, $\hat{a}_\phi$, determines which action leads from one state to a subsequent state. As such it fits perfectly into the generative planning framework, using the first two states of the planner's output as input. The inverse dynamics model is instantiated as a simple two-layer, feed-forward MLP with no output activation function. It is trained to minimize the mean square error in predicted action and actual in each episode as shown in line 17 of algorithm 1.

3.5 Model Architecture

The generative planner is comprised of a U-Net operating 2D convolutions over both episode time and observation + reward dimensions. This architecture resembles the classical image-resolving 2D U-Net (Ronneberger et al., 2015), but also includes embeddings for both guidance conditioners and t_{flow}. Residual-Convolutional-Conditioning (RCC) and Residual-DeConvolutional-Conditioning (RDCC) blocks repeatedly process the random noise sample, the conditioning signal, and time signal into a synthetic plan trajectory. Figure 1 depicts the flow of information throughout these network blocks. Refer to figure 2 for a look inside the RCC block. Conditioning is accomplished by runnining the signal through dedicated MLPs, separate for each block, and multiplying the MLP output to the prior convolutional output. In cases where no conditioning is used ($y = \emptyset$), this multiplication is skipped. The time signal also has a dedicated MLP within each block, additively contributing the convolutional result. The RDCC block is composed of the same process as the RCC block, but instead using deconvolution operations.

4 Experimental Setup

To test the reward forecasting generative planner, comparative models were setup with the alternative state+return conditioning signals mentioned in section 3.3. Baselines with no-conditioning and just state conditioning were trained and evaluated as well. All return conditioning signals matched the maximum horizon limited return for a particular dataset. Models were tested in both the Hopper and Walker2D gymnasium environments (Towers et al., 2024). The Hopper environment was tested with two offline datasets: medium-v0 and the medium-expert-v0. The Walker2D environment was trained using only the corresponding medium-v0 dataset (Younis et al., 2024). These datasets are composed of post-training trajectories from a reference policy.

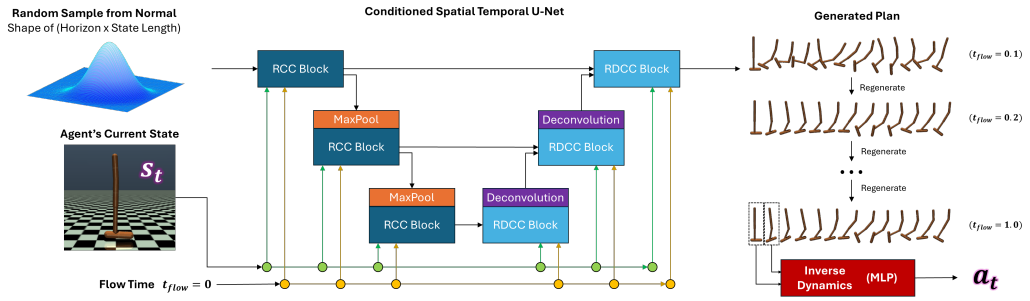

Figure 1: High level process of generating an action using the U-Net architecture used for all generative planners in this project.

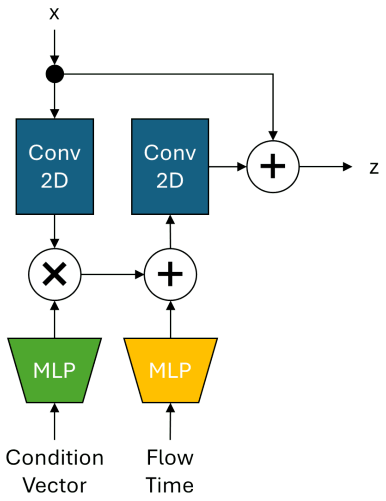

Figure 2: How conditioning and time signals are processed within an individual Residual-Convolutional-Conditioning block.

To judge the efficacy of the generated planners, the mean episode return over 10 trial episodes are reported as a ratio to the maximum return in the dataset. The mean performance of the dataset's policy is also reported as a reference baseline.

All models were trained over 2.3 million gradient steps, equating to 1800 epochs in the medium-v0 dataset and 950 epochs in the medium-expert-v0 dataset. All reported values reflect the models after training was complete.

5 Results

5.1 Quantitative Evaluation

Overall none of the flow matching agents performed as anticipated. All agents performed well below the reference agent's performance, and none came near literature's SOTA performance, which exceeds the reference policy return. Table 5.1 shows this poor performance for the hopper environment. Note that the standard deviations in performance are comparable to the mean returns themselves, indicating that there is high uncertainty in the reported values. The outset hypothesis, which states reward forecasting improves upon reward conditioning, is therefore neither proven nor disproven, as the relative baseline results are unclear.

How does this poor performance manifest? As shown in the video referenced in figure 4, the state conditioning and reward forecasting planners produce smooth, convincing plans, grounded in the

Method	Dataset	
	medium-v0	medium-expert-v0
Dataset Reference Policy	72.1 ± 22.4	75.1 ± 21.4
No Conditioning	7.8 ± 3.6	2.1 ± 1.1
State Conditioning	4.6 ± 10.2	7.8 ± 3.5
Concatenated State + Reward Conditioning	5.8 ± 5.8	5.2 ± 2.6
Independent State + Reward Conditioning	0.6 ± 0.3	1.7 ± 1.3
Reward Forecasting + State Conditioning	8.5 ± 5.9	-0.7 ± 0.4

Table 1: Percent return achieved in the Hopper environment compared to the maximum episode return of the offline dataset. Values are accompanied by standard deviations over ten evaluation episodes. 20 flow steps were used to generate a plan at each time step.

agent's current state. With quality plans, poor performance is surprising. The issue likely lies in a subtle detail of training algorithm 1.

Despite achieving low average training losses, as shown in figure 3, the inverse dynamics model is not well suited to predict which actions connect the states generated by the planner. The inverse dynamics model is trained on real trajectory information but is then asked to predict synthetic data. If the first two states of a generated plan do not resemble the transitions seen in the dataset, expect a nonsense action be produced by the agent.

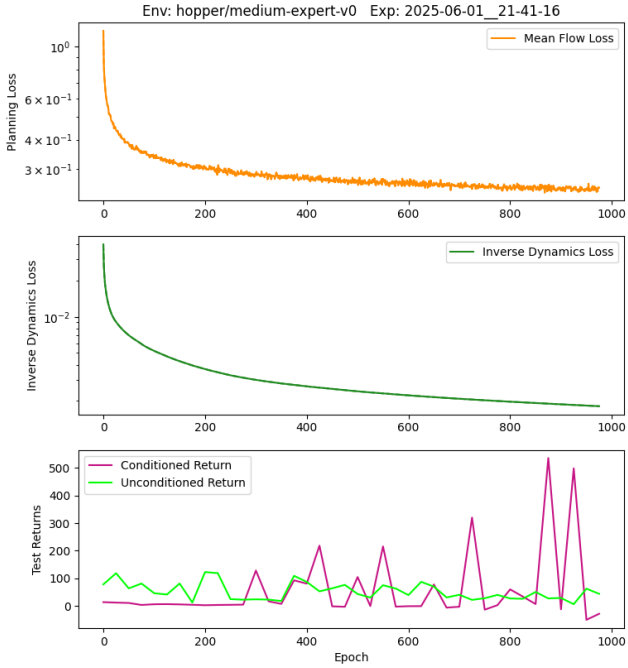

Figure 3: Loss dashboard of the reward forecaster in the medium-expert-v0 hopper environment. The losses shown here are very similar for all other generative planners. Note that returns of 500 equate to 12% performance.

So do all the trained planners produce nonsense actions? Table 5.1 answers this question prescriptively. The euclidean distance between adjacent states are measured and presented. If the generated plans are realistic, they should have the same state transitions as the dataset. The table also shows the euclidean difference in the dataset's actions and each agents actions. While an ideal agent should have some deviation to mark improvement over the reference actor, it worth noting a perfect behavior cloning agent will have zero action deviation.

Method	State Transition L2	State Transition Max	Action Deviation L2	Action Deviation Max
Dataset	1.5	7.7	0	0
No Conditioning	5.7	15.3	11.6	134
State Conditioning	1.9	13.7	1.5	42.7
Concatenated State + Reward Conditioning	2.2	6.5	2.2	81.7
Independent State + Reward Conditioning	2.6	9.7	3.4	117
Reward Forecasting + State Conditioning	1.9	19.9	1.6	98.1

Table 2: Diagnostic metrics showing how far the generated plan for each model deviated from the support of the dataset. Metrics are averaged over 50,000 Monte Carlo samples of transitions in the Hopper medium-expert-v0 dataset.

The diagnostic table shows that all generative planning agents, on average, produce adjacent states with distances further apart then the real transitions. This discrepancy leads to out of distribution data for the inverse dynamics model, which then produces nonsense actions. Inappropriate actions leads to further out of distribution states and then quick episode termination.

5.2 Qualitative Analysis

Not all generative planners achieved the same level of synthesis quality. The state conditioned planner and the reward forecaster both produce convincing plans, while the reward+state conditioned planners output chaotic plans. The video referenced in figure 4 contrasts each agent's plan. A reference trajectory from the dataset is included for comparison.

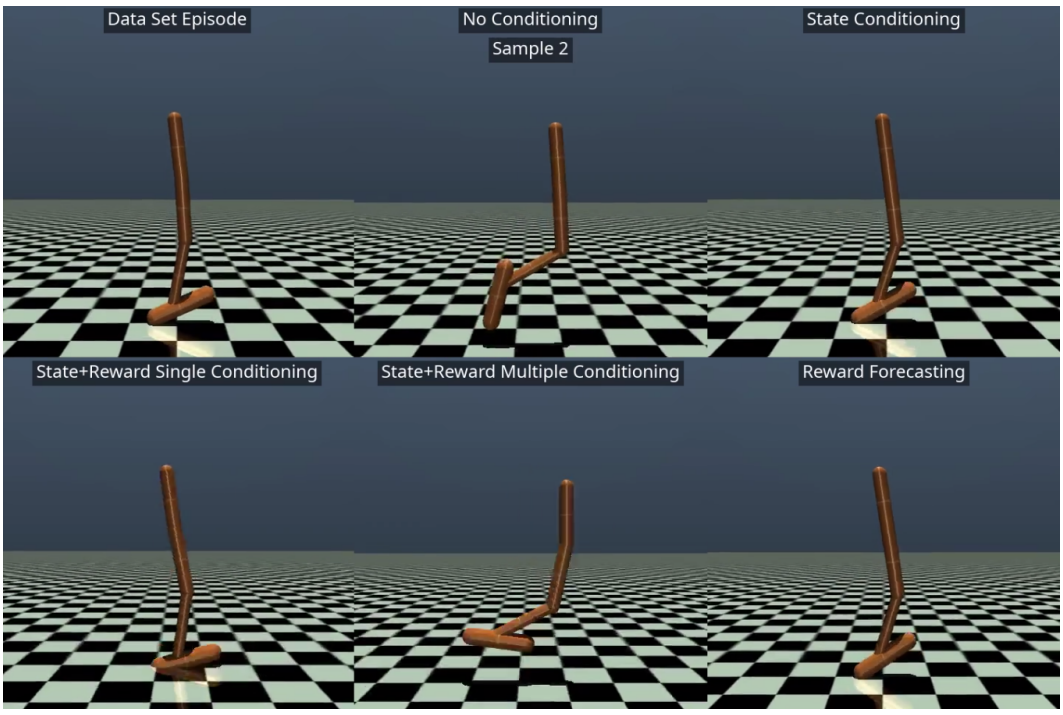

Figure 4: Comparison of initial plans generated by each planning model trained with the hopper medium-expert-v0 dataset. Top left includes a clip of the reference actor for comparison. To see the whole video, visit https://youtu.be/MeWz1-mFVx4

There are a few unrealistic aspect to even the quality plans. Note how, in the video, the state conditioned and reward forecasting agents initiate a plan with a heel strike that leaves the agent hovering. The reference video shows the same action but the agent is in contact with the ground the

whole time. As such the agent's plan is more akin to a hyperactive imagination, that produces subtly unrealistic trajectories.

This lack of realism can also be seen in the walker2d environment. Here the tested agents perform a few highly unrealistic motions like foot slides, which the environment's friction does not allow. For this reason, combined with higher computational expense, the walker2d tests were discontinued to focus on diagnostics with the hopper environment.

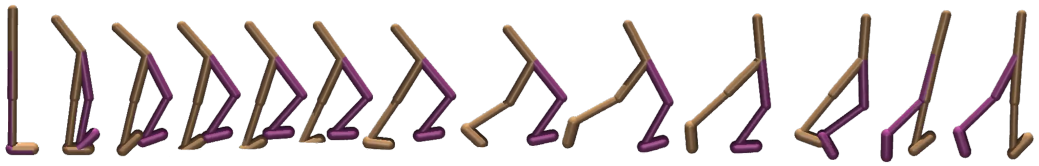

Figure 5: Frame-by-frame of the reward forecaster plan in the walker2d environment. Every tenth frame of the 128 frame plan is shown.

6 Discussion

All generative planners failed to produce synthetic trajectories similar enough to the dataset for the inverse dynamics models to produce sensible actions. This is hinted at in the flow planning losses achieving only a loss of >0.2, where ideally a loss of < 0.1 would be realized. While many trajectories seem superficially realistic, the dilated distance between generated states indicate poor prediction.

Unmentioned in the preceding text, were the attempts to rectify this problem. One major hurdle was the difference in generative architecture from literature. While all results and experiments were shown with the 2D U-Net, additional experiments were run with a 1D U-Net. This more closely matches the architecture in Diffuser and Decision Diffuser, limiting the chances for mistakes. Despite bridging the gap, poor performance was still realized. Differences in architecture were still present. For example, this project uses a simple MLP to learn a flow time embedding, where as the Diffuser uses a sinusoidal transformation + MLP to condition flow time. These types of difference in architecture likely play a large role in performance difference between this project and literature.

7 Conclusion

Reward forecasting has not yet been proven to achieve improved consistency over return+state conditioning. The generative planners trained in this work did not produce synthetic trajectories compatible with their corresponding inverse dynamics models. All models performed poorly and chaotically, leaving the hypothesis open ended.

This shows the challenge in producing generative RL agents. Future work includes closer adherence to literature's generative model architecture. More promisingly however, would be improvements in the robustness of the inverse dynamics model. Perhaps constraints could be applied such that the inverse dynamics model does not produce out of distribution actions. A holistic approach to robustness allows for more flexibility and deeper interpretation of the model's generated plans.

8 Team Contributions

- **Group Member 1:** Just one team member here! All work and effort was done by myself. Special thanks to Xingze Dai for helping guide my project and let me bounce ideas.

References

Anurag Ajay, Yilun Du, Abhi Gupta, Joshua Tenenbaum, Tommi Jaakkola, and Pulkit Agrawal. 2023. Is Conditional Generative Modeling all you need for Decision-Making? arXiv:2211.15657 [cs.LG] https://arxiv.org/abs/2211.15657

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. 2024. π_0: A Vision-Language-Action Flow Model for General Robot Control. arXiv:2410.24164 [cs.LG] https://arxiv.org/abs/2410.24164

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. 2021. Decision Transformer: Reinforcement Learning via Sequence Modeling. arXiv:2106.01345 [cs.LG] https://arxiv.org/abs/2106.01345

Zihan Ding, Amy Zhang, Yuandong Tian, and Qinqing Zheng. 2024. Diffusion World Model: Future Modeling Beyond Step-by-Step Rollout for Offline Reinforcement Learning. arXiv:2402.03570 [cs.LG] https://arxiv.org/abs/2402.03570

Jonathan Ho and Tim Salimans. 2022. Classifier-Free Diffusion Guidance. arXiv:2207.12598 [cs.LG] https://arxiv.org/abs/2207.12598

Michael Janner, Yilun Du, Joshua B. Tenenbaum, and Sergey Levine. 2022. Planning with Diffusion for Flexible Behavior Synthesis. arXiv:2205.09991 [cs.LG] https://arxiv.org/abs/2205.09991

Yaron Lipman, Marton Havasi, Peter Holderrieth, Neta Shaul, Matt Le, Brian Karrer, Ricky T. Q. Chen, David Lopez-Paz, Heli Ben-Hamu, and Itai Gat. 2024. Flow Matching Guide and Code. arXiv:2412.06264 [cs.LG] https://arxiv.org/abs/2412.06264

Seohong Park, Qiyang Li, and Sergey Levine. 2025. Flow Q-Learning. arXiv:2502.02538 [cs.LG] https://arxiv.org/abs/2502.02538

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. arXiv:1505.04597 [cs.CV] https://arxiv.org/abs/1505.04597

Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA.

Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, et al. 2024. Gymnasium: A Standard Interface for Reinforcement Learning Environments. *arXiv preprint arXiv:2407.17032* (2024).

Zhendong Wang, Jonathan J Hunt, and Mingyuan Zhou. 2023. Diffusion Policies as an Expressive Policy Class for Offline Reinforcement Learning. arXiv:2208.06193 [cs.LG] https://arxiv.org/abs/2208.06193

Omar G. Younis, Rodrigo Perez-Vicente, John U. Balis, Will Dudley, Alex Davey, and Jordan K Terry. 2024. *Minari*. doi:10.5281/zenodo.13767625

Qinqing Zheng, Matt Le, Neta Shaul, Yaron Lipman, Aditya Grover, and Ricky T. Q. Chen. 2023. Guided Flows for Generative Modeling and Decision Making. arXiv:2311.13443 [cs.LG] https://arxiv.org/abs/2311.13443