DECISION TRANSFORMER UNDER RANDOM FRAME DROPPING

Anonymous authors
Paper under double-blind review

ABSTRACT

Controlling agents remotely with deep reinforcement learning (DRL) in the real world is yet to come. One crucial stepping stone is to devise RL algorithms that are robust in the face of dropped information from corrupted communication or malfunctioning sensors. Typical RL methods usually require considerable online interaction data that are costly and unsafe to collect in the real world. Furthermore, when they are applied to the frame dropping scenarios, they perform unsatisfactorily even with moderate drop rates. To devise a robust and deployable algorithm, we propose Decision Transformer under Random Frame Dropping (DeFog), an offline RL algorithm that enables agents to act robustly in frame dropping scenarios without online interaction. DeFog first randomly masks out data in the offline datasets and explicitly adds the timespan of frame dropping as inputs. After that, a finetuning stage on the same offline dataset with a higher mask rate would further boost the performance. Empirical results show that DeFog outperforms strong baselines under severe frame drop rates like 90%, while maintaining similar returns under non-frame-dropping conditions in the regular MuJoCo control benchmarks and the Atari environments.

1 INTRODUCTION

Imagine you are guiding a drone through a forest. When you look at the images transmitted from the drone, the display delays heavily or even disappears intermittently due to bad communication. An experienced drone pilot would stabilize the drone with the last received frame until the communication recovers to normal.

In this paper, we aim to empower deep reinforcement learning (RL) algorithms with such abilities to control remote agents. In many real world control tasks, the decision makers are separate from the action executor (Saha & Dasgupta, 2018), which introduce the risk of packet loss and delay during network communication. Furthermore, sensors (e.g., camera, IMUs) are sometimes prone to temporary malfunctioning, or limited by hardward restrictions, thus causing the observation to be unavailable at certain timesteps (Dulac-Arnold et al., 2021). These examples lead to the core challenge of devising the desired algorithm: controlling the agents against frame dropping, i.e., a temporary loss of observations as well as other information.

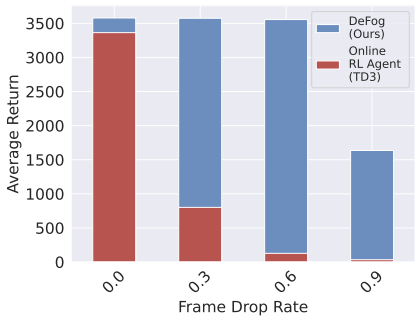

Figure 1: RL performance in the Hopper-v3 environment under different frame drop rates.

Figure 1 illustrates how a regular RL algorithm performs under high frame drop rates. We find that RL agents trained in zero-drop-rate environments encounter a harsh performance degradation when frames drop. This indicates that the frame dropping problem is a severe problem that needs to be resolved if we want to apply RL into remote control tasks.

This problem gradually attracts more attention recently: Nath et al. (2021) adapt vanilla DQN algorithm to a randomly delayed markov decision process; Bouteiller et al. (2020) propose a method that

modifies the classic Soft Actor-Critic algorithm (Haarnoja et al., 2018) to handle observation and action delay scenarios. In contrast to the delayed frames in previous works, we try to solve a more challenging problem where the frames are permanently lost. Moreover, previous methods usually learn in an online frame dropping environment, which can be unsafe and costly.

In this paper, we propose Decision Transformer under Random Frame Dropping (DeFog), an offline reinforcement learner that are robust to frame drops. Specifically, we use a Decision Transformer backbone (Chen et al., 2021) to learn from randomly masked offline datasets and add ~~the drop-span~~ an embedding indicating the timespan of frame dropping to the inputs. Moreover, in continuous control tasks, a finetuning stage with the parameters of the Decision Transformer fixed would further increase the robustness of DeFog.

We evaluate our method on continuous and discrete control tasks in MuJoCo and Atari game environments. In these environments, observations will be dropped randomly before they are sent to the agent. Empirical results show that DeFog outperforms various baselines under the frame drop setting, while maintaining performance that are comparable to other offline RL methods in regular non-frame-dropping environments.

The key contribution of this is that we propose Decision Transformer under Random Frame Dropping (DeFog), a method that is robust to frame drops without any online interaction. The offline nature makes it promising and practical in future real-world remote control tasks. When tested, DeFog achieves superior results in both continuous and discrete control tasks against dropped frames.

2 Related Works

2.1 Control under Frame Dropping and Delay

The observation and control loss or delay is an essential problem in remote control tasks (Balemi & Brunner, 1992; Funda & Paul, 1991). In recent cloud-edge computing systems such as intelligent connected vehicles (Li et al., 2018) and UAV swarm (Bekkouche et al., 2018; Hwangbo et al., 2017), this problem is drawing more attention.

When reinforcement learning is applied to such remote control tasks, a robust RL algorithm is desired. Katsikopoulos & Engelbrecht (2003) first propose the formulation of the Random Delayed Markov Decision Process. Along with the new formulation, a method is proposed to augment the observation space with the past actions. However, previous methods (Walsh et al., 2009; Schuitema et al., 2010) usually stack the delayed observations together, which might require a constant delay duration as a hard threshold.

Hester & Stone (2013) propose to solve this problem by predicting delayed states with a random forest model. Bouteiller et al. (2020) propose to tackle random observation and action delays in a model-free manner by relabelling the past actions with the current policy to mitigate the off-policy problem. Later, Nath et al. (2021) propose to learn a deep Q network (DQN) agent with state augmentation. However, these methods that rely on state augmentation usually assume a maximum delay time. Imai et al. (2021) train a vision-guided quadrupedal robot in the wild against random observation delay by leveraging delay randomization. This work shares the same intuition of the train-time frame masking part, while we are using a decision transformer backbone with additional frame drop interval embedding and a performance-improving finetuning technique.

2.2 Transformers in Reinforcement Learning

Researchers recently formulate the decision making procedure in offline reinforcement learning as a sequence modeling problem using transformer models (Chen et al., 2021; Janner et al., 2021). In contrast to the policy gradient and temporal difference methods, these works advocate the paradigm of treating reinforcement learning as a supervised learning problem (Schmidhuber, 2019), directly predicting actions from the observation sequence and the task specification. The Decision Transformer model (Chen et al., 2021) takes the encoded reward-to-go, state, and action sequence as input to predict the action for the next step, while the Trajectory Transformer (Janner et al., 2021) first discretizes each dimension of the input sequence, maps them to tokens, then predicts the following action's tokens with a beam search algorithm.

The concurrent occurrence of these works attracted much attention in the RL community for further improving upon the transformers. Zheng et al. (2022) increases the model capacity and enables online finetuning of the Decision Transformer by changing the deterministic policy to a stochastic one and adding an entropy term to encourage exploration. Tang & Ha (2021) train transformer-based agents that are robust to permutation of the input order. Apart from these works, various attempts have been made to improve transformers in multi-agent RL, meta RL, multi-task RL, and many other fields (Meng et al., 2021; Xu et al., 2022; Lee et al., 2022; Reid et al., 2022).

3 METHOD

In this section, we first describe the problem setup and then introduce Decision Transformer under Random Frame Dropping (DeFog), a flexible and powerful method to tackle sporadic dropping of the observation and the reward signals.

3.1 PROBLEM STATEMENT

In the environment with random frame dropping, the original state transitions of the underlying Markov Decision Process are broken; hence, the observed states and rewards follow a new transition process. We name the new decision process Random Dropping Markov Decision Process (RDMDP). Formally, we define the RDMDP as follows:

Definition 1 (Random Dropping Markov Decision Process (RDMDP))
An RDMDP could be described as $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mu, \mathcal{P}, \mathcal{D}, \mathcal{O}_{\mathcal{S}}, \mathcal{O}_{\mathcal{R}} \rangle$, where $\mathcal{S}, \mathcal{A}$ are the state and action space, $\mathcal{P}(s_{t+1} \mid s_t, a_t)$ is the state transition possibility, $\mathcal{R}(s_t, a_t)$ is the reward function, $\mu(s_0)$ is the initial state distribution. $\mathcal{D}$ is the Bernoulli Distribution of frame dropping, $\mathcal{O}_{\mathcal{S}}$ is the function that generates the observation, and $\mathcal{O}_{\mathcal{R}}$ the function that generates the cumulative rewards. In the drop frame setting, we assume the cumulative reward $R_t = \sum_{\tau=0}^{t} r_t$ is observed instead of the immediate reward r_t.

At each timestep t, a drop frame indicator $d_t \in \{0, 1\}$ is drawn from the distribution $\mathcal{D}$, with $d = 1$ indicating that the frame is dropped and $d = 0$ the opposite. The observed state and the cumulative reward $\hat{s}_t$ and $\hat{R}_t$ of the timestep are updated by

$$\hat{s}_t = \mathcal{O}_{\mathcal{S}}(s_t, \hat{s}_{t-1}, d) = d \cdot \hat{s}_{t-1} + (1 - d) \cdot s_t \tag{1}$$

$$\hat{R}_t = \mathcal{O}_{\mathcal{R}}(R_t, \hat{R}_{t-1}, d) = d \cdot \hat{R}_{t-1} + (1 - d) \cdot R_t \tag{2}$$

The observation and the cumulative reward repeats the last observed one $\hat{s}_{t-1}$ and $\hat{R}_{t-1}$ respectively if the current frame is dropped. If the current frame arrives normally, the observation and cumulative reward is updated. Note that the intermediate rewards obtained during the dropped frame are also added to the cumulative reward since the operation only happens on the remote side. Following the definition in the Decision Transformer, when a target return R_{target} is set for the environment, we define the real and observed reward-to-go as $g_t = R_{target} - R_t$, and $\hat{g}_t = R_{target} - \hat{R}_t$ respectively.

The goal of DeFog is to extract useful information from the offline dataset so that the agent can act smoothly and safely in a frame-dropping environment.

3.2 DECISION TRANSFORMER UNDER RANDOM FRAME DROPPING

We choose decision transformer as our backbone because of its expressiveness and flexibility as an offline RL learner. Then, we attack the random frame dropping problem from three perspectives: 1) We modify the offline dataset by randomly masking out observations and reward-to-gos during training and dynamically update the mask ratio; 2) we provide a drop-span embedding that captures the information of dropped frame duration; 3) we find the agent can act robustly against higher dropping rate when finetuning the drop-span encoder and action predictor after the model is fully converged. A full illustration of our method is shown in Figure 2.

3.2.1 DECISION TRANSFORMER BACKBONE

The Decision Transformer (DT) takes in past K timesteps of reward-to-goes $g_{t-K,t}$, observations $s_{t-K,t}$, and actions $a_{t-K,t}$ as embedded tokens and predicts the next tokens in the same way as the

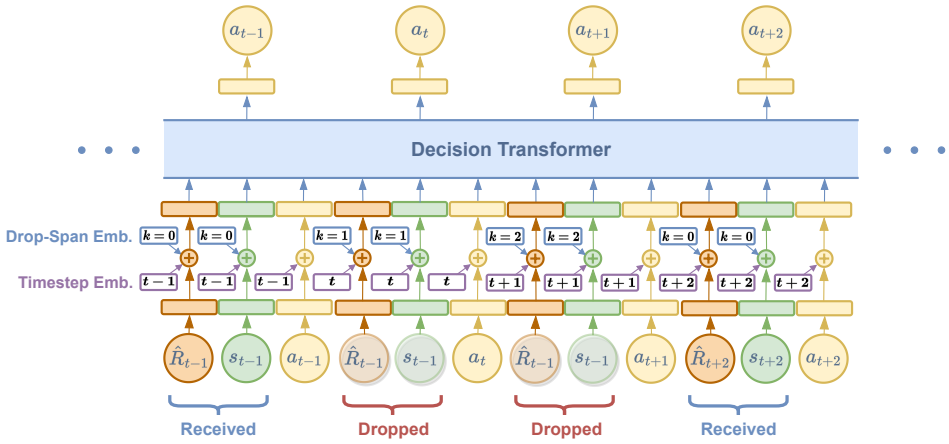

Figure 2: **Decision Transformer under Random Frame Dropping (DeFog).** Reward-to-goes (RTGs) and states are repeated from the previous steps if the current frame is dropped. Timestep and drop-span embeddings, indicating the timestep and number of consecutive frame drops, are added onto the encoded RTGs and states before sent to the Decision Transformer backbone. Since actions are not dropped, only the timestep embeddings are added to the encoded actions. The DT backbone outputs the predicted action embeddings, which is passed through a decoder to obtain the predicted actions.

Generative Pre-Training model (Radford et al., 2018). Let ϕ_g, ϕ_s and ϕ_a denote the reward-to-go, state, and action encoders respectively, the input tokens are obtained by first mapping the inputs to a d-dimensional embedding space, then add a timestep embedding $\omega(t)$ to the tokens.

Let u_{g_t}, u_{s_t} and u_{a_t} denote the input tokens corresponding to the observation, the action and the reward-to-go of the t-th timestep respectively, and v_{g_t}, v_{s_t} and v_{a_t} be their counterparts on the output token side. DT could be formalized as:

$$u_{s_t} = \phi_s(s_t) + \omega(t), \quad u_{g_t} = \phi_g(g_t) + \omega(t), \quad u_{a_t} = \phi_a(a_t) + \omega(t) \tag{3}$$

$$v_{g_{t-K}}, v_{s_{t-K}}, v_{a_{t-K}}, \ldots, g_{s_t}, s_{g_t}, v_{a_t} = DT(u_{g_{t-K}}, u_{s_{t-K}}, u_{a_{t-K}}, \ldots, u_{g_t}, u_{s_t}, u_{a_t}) \tag{4}$$

Online decision transformer (ODT) by Zheng et al. (2022) enables online finetuning of the decision transformer models. We adopt the ODT model architecture because it has larger model capacity. Following this work, we also omit the timestep embedding $\omega(t)$ in the gym environments.

During training time, instead of directly predicting the action a_t, we follow the setting of the ODT to predict a gaussian distribution for action from the output token of state token inputs.

$$\pi_\theta\left(a_t \mid v_{s_t}\right) = \mathcal{N}\left(\mu_\theta\left(v_{s_t}\right), \Sigma_\theta\left(v_{s_t}\right)\right) \tag{5}$$

The covariance matrix Σ_θ is assumed to be diagonal, and the training target is to minimize the negative log-likelihood for the model to produce the real action in the dataset:

$$J(\theta) = \frac{1}{K}\mathbb{E}_{(\mathbf{a},\mathbf{s},\mathbf{g})\sim\mathcal{T}}\left[-\sum_{k=1}^{K}\log \pi_\theta\left(a_k \mid v_{s_k}\right)\right] \tag{6}$$

3.2.2 Train-Time Frame Dropping

To prepare the model for frame dropping, we mask out observations and reward-to-gos mannually from the dataset. During the training stage, we specify some empirical dropping distribution $\hat{\mathcal{D}}$ and sample "drop-mask" periodically from it. A drop-mask is a binary vector the same size as the dataset, It serves as the drop distribution $\mathcal{D}$ of an RDMDP. If a frame in the dataset is marked as "dropped" by the current drop-mask, the observation and the reward-to-go of that frame are then overwritten by the most recent non-dropped frame. We name the timespan between the current frame and the last non-dropped frame of it the drop-span of that frame.

One key consideration for the training scheme is the distribution $\hat{\mathcal{D}}$ of the drop-mask. A natural solution is to assume each frame to have the same possibility p_d to be dropped, and the occurrence of frame dropping is independent. Under this assumption, the random variable sequence of whether each frame is dropped becomes a Bernoulli process. Additionally, we guarantee that the first frame for each trajectory is noy dropped. After a certain training interval, the drop-mask is resampled from $\hat{\mathcal{D}}$ so that those dropped frames of the dataset could be used. We can also change $\hat{\mathcal{D}}$ as the training proceeds, for example, to linearly increase the p_d throughout training. However, we empirically find that usually, a constant p_d is sufficient.

3.2.3 Drop-Span Embedding

In a frame dropping scenario, the agent must utilize the information of the past action sequence since the last observation. It is also required for the agent to deal with the missing observation and reward-to-go tokens. Instead of dropping the corresponding tokens in the input sequence, we repeat the last observation or reward-to-go token and explicitly add a unique drop-span embedding to those tokens apart from the original timestep embedding $\omega(t)$.

Let k_t denote the drop-span since the last observation of time t, the drop-span encoder ψ maps integer k_t to a d-dimensional token the same shape as the other observed input tokens $u_{\hat{g}_t}, u_{\hat{s}_t}, u_{a_t}$. The model input with the drop-span embedding becomes:

$$u_{\hat{s}_t} = \phi_s(\hat{s}_t) + \psi(k_t) + \omega(t), \quad u_{\hat{g}_t} = \phi_g(\hat{g}_t) + \psi(k_t) + \omega(t), \quad u_{a_t} = \phi_a(a_t) + \omega(t) \tag{7}$$

Since the actions are decided and executed by the agent itself, they do not face the problem of frame dropping. The drop-span embedding is an analogy to the function of the timestep embedding in the Decision Transformer, but it bears the semantic meaning of how many frames have passed. Compared to the other indirect methods, the explicit use of drop-span embedding achieves better results. Detailed comparison could be found in Section 4.5.

3.2.4 Freeze Trunk Finetuning

The combination of the train-time frame dropping and the drop-span embedding is adequate to make our model rather robust to dropped frames. However, we found that in the continuous control tasks, a fine-tuning procedure could further improve the performance in more challenging scenarios.

Inspired by recent progress on prompt finetuning in both the natural language processing (Liu et al., 2021) and the computer vision (Jia et al., 2022) areas, we propose to finetune the drop-span encoder ψ and the action predictor π_θ after the entire model converges. We term this procedure "freeze trunk finetuning" since the majority of the model parameters are frozen during the finetuning.

The training procedure during this stage is the same as that of the entire model, as we also draw drop-masks from $\hat{\mathcal{D}}$ during this stage to give the drop-span embeddings enough supervision. While the training steps of this stage could be just one-fifth of the main stage, the drop rate p_d can usually be higher. Empirical results prove its ability to improve the model's performance in higher dropping rates among multiple environments. The whole training pipeline of our method could be found at Appendix A.

4 Experimental Results

In this section, we describe our experiment setup and analyze the results. We compare our method with state-of-the-art delay-aware reinforcement learning methods, as well as online and offline reinforcement learning methods in multiple frame dropping setttings. We first evaluate whether DeFog is able to maintain its performance as the drop rate p_d increases. We then explore which key factors and design choices helped DeFog to achieve its performance. Finally, we provide insights of why DeFog can accomplish control tasks under severe frame dropping conditions.

4.1 Experiment Setup

To comprehensively evaluate the performance of DeFog and baselines, we conduct experiments on three continuous control environments with proprioceptive state inputs in the gym MuJoCo environ-

ment (Todorov et al., 2012), as well as three discrete control environments with high-dimensional image inputs in the Atari games.

In each of the three MuJoCo environments, we collect offline datasets of three different levels: expert, medium, and medium-replay. While in the three Atari environments, we follow the Decision Transformer to train on an average sampled dataset from a DQN agent's replay buffer. We train on 3 seeds and average their results in test time. We leave the detailed description of the settings to Appendix B.2.

During evaluation, we test the agents in an environment that has frame drop rates ranging from 0% to 90%. Results are shown by plotting the average return under 10 trials against test-time drop rate for different agents. The performance curve of our method is compared against three baselines:

- **Reinforcement learning with random delays (RLRD; Bouteiller et al., 2020).** In order to train delay-robust agents, RLRD adds randomly sampled delay as well as an action buffer to its observation space. Since RLRD has a maximum delay constraint, we limit the delay value in the augmented observation to its maximum even if frames are still dropped. RLRD performs max-entropy learning of the action, it's not suited for discrete action tasks like Atari, so its performance is only included in the gym MuJoCo environment.
- **Twin-delayed DDPG (TD3; Fujimoto et al., 2018).** We train an online expert RL agent under regular non-frame-dropping settings using TD3 for continuous control tasks. We note that TD3 also has the privilege to access the environment.
- **Decision transformer (DT; Chen et al., 2021).** We train the vanilla DT using exactly the same offline dataset without the proposed components in Section 3.2.
- **Batch-constrained deep Q-learning (BCQ; Fujimoto et al., 2019).** BCQ is an offline RL method that aims to reduce the extrapolation error in offline RL by encouraging the policy to visit states and actions similar to the datasets.
- **TD3 + Behavioral cloning (TD3+BC; Fujimoto & Gu, 2021).** TD3+BC is built on top of TD3 to work offline by adding a behavior cloning term to the maximizing objective. Despite the simplicity, it is able to match the state-of-the-art performance.
- **Conservative Q-learning (CQL; Kumar et al., 2020).** CQL is a state-of-the-art model-free method which tries to overcome the over-estimation problem in offline RL by learning a conservative Q function that lower bounds the real one.

We leverage the implementation of Takuma Seno (2021) to train an offline agent in BCQ, TD3+BC, and CQL. We note that the online methods such as RLRD and TD3 are trained directly in the environment. Hence, they do not have any differences in terms of the dataset types. Their performance curves are plotted repeatedly for comparison. We also include a DeFog without finetuning version to evaluate the effectiveness of freeze trunk finetuning. Since TD3 cannot handle frame-dropping scenarios, we only plot TD3 in one row of the figures for better illustration. For a fair comparison, we assume the delay for RLRD is created by resending the dropped observations, which again has a probability p_d of being lost. For baselines of the discrete control tasks, we train the offline RL agents until they reach the performance of DeFog under non-frame-dropping conditions. By doing so, we can make a fairer comparison as we evaluate the performance of different methods under frame dropping settings.

4.2 Evaluation in the Continuous Control Tasks

We first evaluate our performance on three MuJoCo Continuous control environments, namely Hopper-v3, HalfCheetah-v3, and Walker2d-v3, each on the three levels. The results on each dataset are given in Figure 3.

As shown in Figure 3, we find that DeFog is able to maintain performance under severe drop rates. For example, the performance of the finetuned version on Walker2d dataset barely decreases even when the drop rate is as high as 80%. Meanwhile, the performance of the vanilla DT and TD3 agents come close to zero once the drop rate exceeds 67%.

By looking at the starting point of the performance curves, we see that DeFog can achieve the same performance as the vanilla DT agent in non-frame-dropping scenarios. Despite a high train-time

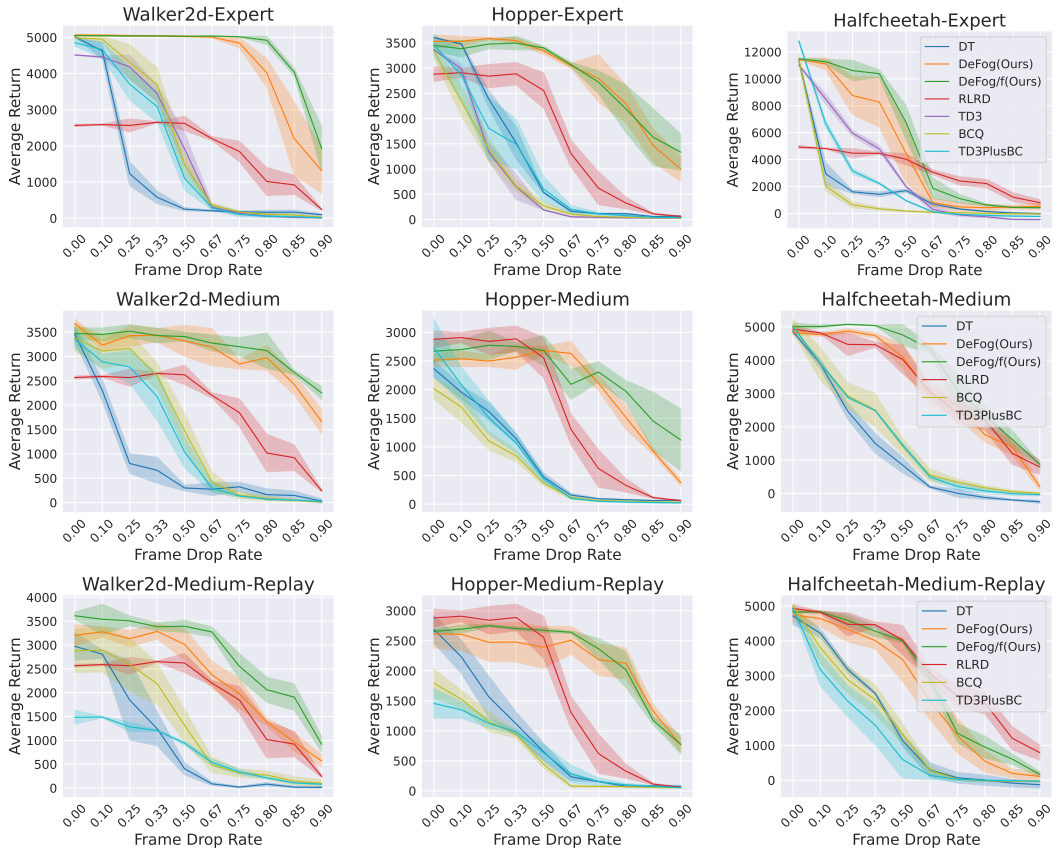

Figure 3: Performance on continuous control tasks. There are three baselines: a) TD3: the online TD3 agent. b) DT: the offline Decision Transformer agent trained on the same dataset as DeFog. c): RLRD: the online RLRD agent that is optimized to deal with random frame delay. The full-fledged version of our method is indicated with DeFog/f (Ours), while the result of a non-finetuned version is indicated with DeFog (Ours) for comparison.

drop rate of 80% applied to DeFog none of them are negatively affected when tested with a drop rate of 0%. This hints that DeFog is data efficient and could learn with limited access to the full dataset. As a comparison, the online RLRD failed to achieve the same non-frame-dropping performance as other online baselines.

In the HalfCheetah with expert dataset setting, our method significantly outperforms the RLRD baseline with drop rates lower than 50%; however, in more extreme cases the RLRD takes over. RLRD, with its advantage of interacting with the environment, was able to keep the performance better possibly due to HalfCheetah being relatively easy out of these three environments. In the medium and medium-replay datasets, DeFog is limited to data with less expertise, thus obtaining a reduced average return, while RLRD with full access could only reach of maximum level of the same return.

We can also find that freeze trunk finetuning effectively improves DeFog's performance in various settings. In all 9 settings, the finetuned agent obtains better or at least the same results with the non-finetuned ones. The finetuning is especially helpful in high drop rate scenarios as shown in the Hopper-medium and Walker2d-expert settings. We highlight that the finetuning also only needs the offline dataset rather than online interactions.

4.3 Evaluation in the Discrete Control Tasks

In this section, we evaluate our performance on three Atari discrete control environments (Bellemare et al., 2013): Qbert, Breakout, and Seaquest. Following the practice of the Decision Transformer,

we use 1% of a fully trained DQN agent's replay buffer to train the agent. The results are shown in Figure 4. We find that DeFog outperforms the Decision Transformer. Similarly, we attribute the performance gain to the proposed data masking and drop-span embeddings. We also find that in some environments, the performance of DeFog outperforms the Decision Transformer even under non-frame-dropping conditions. We believe that these environments, using masked out datasets not only helps the agent to be more robust to frame dropping, but also makes the task more challenging in the sense that the agent needs to understand the environment dynamics to give better action predictions. This requirement not only helps the agent in frame dropping scenarios, but also lets it make better decisions in general, helping performance when frame drop rate is 0.

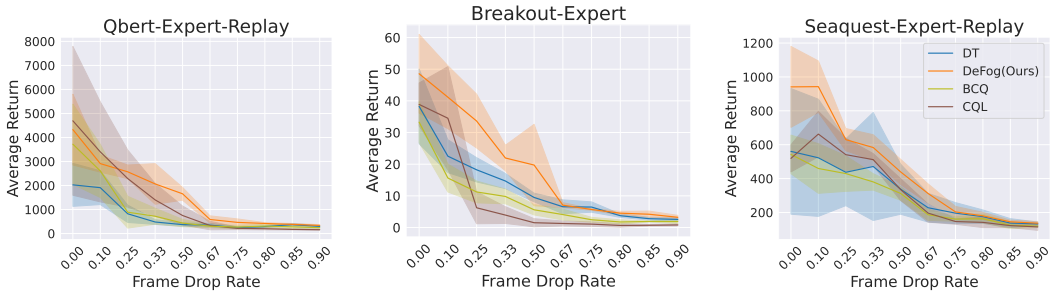

Figure 4: The performance in the discrete Atari game environments. DT is the offline Decision Transformer agent trained on the same dataset as DeFog. Our method is indicated with DeFog (Ours).

4.4 Visualized Results

To gain a better understanding of DeFog, we visualize the results in the MuJoCo environment.

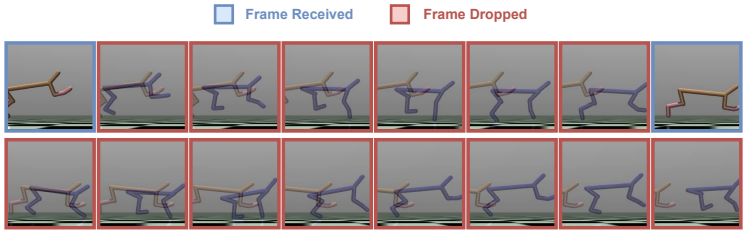

Figure 5: Visualization results of DeFog HalfCheetah agent under 90% frame drop rate. In each frame, the orange cheetah is the observed state $\hat{s}_t$, while the purple cheetah is the actual state s_t.

We first visualize an agent that is trained with DeFog in a frame-dropping environment. We show the frames of a recorded trajectory in Figure 5, under a frame dropping rate of 90%. As shown, the HalfCheetah agent (blue) is able to act correctly even if the observation (semi-transparent yellow) is stuck at 8 steps ago. Once a new observation comes in, the agent immediately adjusts itself to the newest state and continues to perform a series of correct actions.

Building on top of the previous setting, we aim to exploit the capability of DeFog under extreme conditions by increasing the frame drop rate to 100%. In this way, the agent is only able to look at the initial observation and needs to make the rest of the decisions blindly. As shown in Figure 6, the HalfCheetah agent continues to run smoothly for more than 24 frames. This may be due to the Transformer architecture's ability to infer from contextual action history, predicting a sequence of actions even without seeing the next observations. We conjecture that such a phenomenon is analogous to how humans perform a skill such as swing a tennis racket without using the observations.

4.5 Ablation Study

We conduct ablation study on the drop-span embeddings and freeze trunk finetuning parts of DeFog.

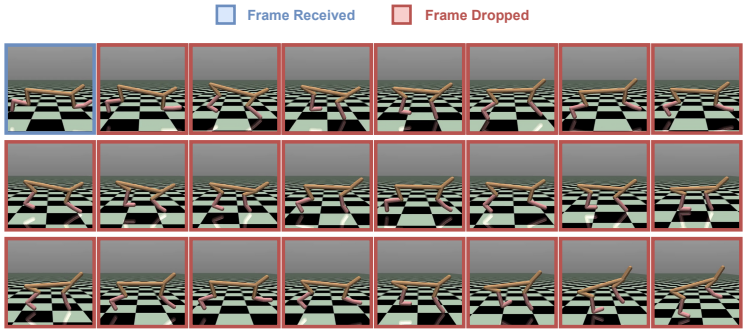

Figure 6: Visualization results of DeFog HalfCheetah agent under 100% frame drop rate. Only the very first observation is received. The scenarios challenges the limit of the DeFog model.

For the drop-span embeddings, we implement an alternative method that implicitly embeds the drop span. Concretely, at each time step t, if the current frame is dropped, we change the corresponding timestep embedding $\omega(t)$ to the last received one $t - k_t$. Hence, for a implict embedding method, we have $u_{s_t} = \phi_s(s_t) + \omega(t - k_t),\ u_{g_t} = \phi_g(g_t) + \omega(t - k_t),\ u_{a_t} = \phi_a(a_t) + \omega(t)$. In this way, the agent can similarly have the information of the drop-span. As shown in Figure 7, we see that the proposed drop-span embedding outperforms the implicit embedding, showing the effectiveness and necessity of explicitly providing the drop-span information to DeFog.

Now we ablate the freeze trunk finetuning method by comparing the the same model with and without the finetuning stage. All comparisons are performed on three MuJoCO medium-replay datasets. As shown in Figure 7, the finetuning largely outperforms the original model on the continuous tasks. We believe that the performance gain in complex continuous control tasks is because the lightweight yet crucial modules (action predictor, drop-span encoder) are easier to be trained for accurate actions. We leave more results in the ablation section in Appendix C.

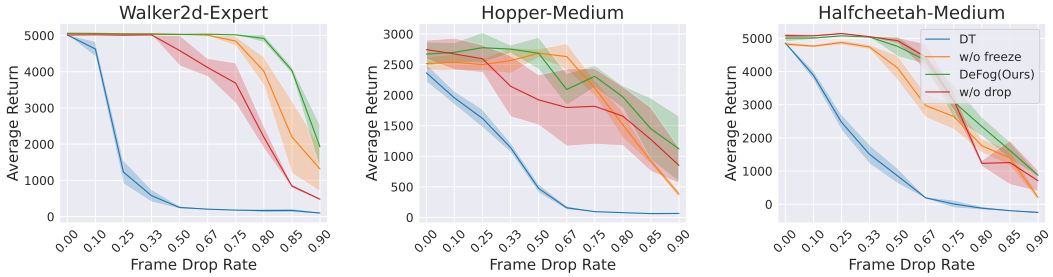

Figure 7: Ablation study of the drop-span embedding and the finetuning stage. "w/o freeze" means without freeze trunk finetuning, and "w/o drop" stands for without drop-span embedding. We find that the proposed methods are effective in the continuous control tasks.

5 Conclusion

In this paper, we propose Decision Transformer under Random Frame Dropping (DeFog), an algorithm based on the Decision Transformer that is robust against a crucial challenge in real-world remote control tasks — frame dropping. DeFog simulates the frame dropping phenomenon with by randomly masking out the observations in offline datasets and embeds frame dropping timespan information explicitly to the model. Furthermore, when finetuned on the drop-span encoder and the action predictor, it becomes more robust to high dropping rates in continuous tasks. Empirical results show that DeFog outperforms strong baselines on continuous and discrete control benchmarks under severe frame dropping settings, where frame drop rates may be as high as 90%. Along with work, we find that a promising future direction is to further handle corrupted observations such as blurred images or inaccurate velocities and to deploy on a real robot.

REFERENCES

Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. An optimistic perspective on offline reinforcement learning. In *International Conference on Machine Learning*, pp. 104–114. PMLR, 2020.

S Balemi and UA Brunner. Supervision of discrete event systems with communication delays. In *1992 American Control Conference*, pp. 2794–2798. IEEE, 1992.

Oussama Bekkouche, Tarik Taleb, and Miloud Bagaa. Uavs traffic control based on multi-access edge computing. In *2018 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6. IEEE, 2018.

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47: 253–279, 2013.

Yann Bouteiller, Simon Ramstedt, Giovanni Beltrame, Christopher Pal, and Jonathan Binas. Reinforcement learning with random delays. In *International conference on learning representations*, 2020.

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.

Gabriel Dulac-Arnold, Nir Levine, Daniel J Mankowitz, Jerry Li, Cosmin Paduraru, Sven Gowal, and Todd Hester. Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Machine Learning*, 110(9):2419–2468, 2021.

Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning, 2020.

Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pp. 20132–20145, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/a8166da05c5a094f7dc03724b41886e5-Abstract.html.

Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pp. 1587–1596. PMLR, 2018.

Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 2052–2062. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/fujimoto19a.html.

J Funda and RP Paul. Efficient control of a robotic system for time-delayed environments. In *Fifth International Conference on Advanced Robotics' Robots in Unstructured Environments*, pp. 219–224. IEEE, 1991.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1861–1870. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/haarnoja18b.html.

Todd Hester and Peter Stone. Texplore: real-time sample-efficient reinforcement learning for robots. *Machine learning*, 90(3):385–429, 2013.

Jemin Hwangbo, Inkyu Sa, Roland Siegwart, and Marco Hutter. Control of a quadrotor with reinforcement learning. *IEEE Robotics and Automation Letters*, 2(4):2096–2103, 2017.

Chieko Sarah Imai, Minghao Zhang, Yuchen Zhang, Marcin Kierebinski, Ruihan Yang, Yuzhe Qin, and Xiaolong Wang. Vision-guided quadrupedal locomotion in the wild with multi-modal delay randomization. *arXiv preprint arXiv:2109.14549*, 2021.

Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one big sequence modeling problem. In *Advances in Neural Information Processing Systems*, 2021.

Menglin Jia, Luming Tang, Bor-Chun Chen, Claire Cardie, Serge Belongie, Bharath Hariharan, and Ser-Nam Lim. Visual prompt tuning. *arXiv preprint arXiv:2203.12119*, 2022.

Konstantinos V Katsikopoulos and Sascha E Engelbrecht. Markov decision processes with delays and asynchronous cost collection. *IEEE transactions on automatic control*, 48(4):568–574, 2003.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1179–1191, 2020.

Kuang-Huei Lee, Ofir Nachum, Mengjiao Yang, Lisa Lee, Daniel Freeman, Winnie Xu, Sergio Guadarrama, Ian Fischer, Eric Jang, Henryk Michalewski, et al. Multi-game decision transformers. *arXiv preprint arXiv:2205.15241*, 2022.

Yongfu Li, Chuancong Tang, Srinivas Peeta, and Yibing Wang. Nonlinear consensus-based connected vehicle platoon control incorporating car-following interactions and heterogeneous time delays. *IEEE Transactions on Intelligent Transportation Systems*, 20(6):2209–2219, 2018.

Xiao Liu, Kaixuan Ji, Yicheng Fu, Zhengxiao Du, Zhilin Yang, and Jie Tang. P-tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks. *arXiv preprint arXiv:2110.07602*, 2021.

Linghui Meng, Muning Wen, Yaodong Yang, Chenyang Le, Xiyun Li, Weinan Zhang, Ying Wen, Haifeng Zhang, Jun Wang, and Bo Xu. Offline pre-trained multi-agent decision transformer: One big sequence model conquers all starcraftii tasks. *arXiv preprint arXiv:2112.02845*, 2021.

Somjit Nath, Mayank Baranwal, and Harshad Khadilkar. Revisiting state augmentation methods for reinforcement learning with stochastic delays. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pp. 1346–1355, 2021.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

Machel Reid, Yutaro Yamada, and Shixiang Shane Gu. Can wikipedia help offline reinforcement learning? *arXiv preprint arXiv:2201.12122*, 2022.

Olimpiya Saha and Prithviraj Dasgupta. A comprehensive survey of recent trends in cloud robotics architectures and applications. *Robotics*, 7(3), 2018. ISSN 2218-6581. doi: 10.3390/robotics7030047. URL `https://www.mdpi.com/2218-6581/7/3/47`.

Juergen Schmidhuber. Reinforcement learning upside down: Don't predict rewards–just map them to actions. *arXiv preprint arXiv:1912.02875*, 2019.

Erik Schuitema, Lucian Buşoniu, Robert Babuška, and Pieter Jonker. Control delay in reinforcement learning for real-time dynamic systems: A memoryless approach. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3226–3231. IEEE, 2010.

Michita Imai Takuma Seno. d3rlpy: An offline deep reinforcement library. In *NeurIPS 2021 Offline Reinforcement Learning Workshop*, December 2021.

Yujin Tang and David Ha. The sensory neuron as a transformer: Permutation-invariant neural networks for reinforcement learning. *Advances in Neural Information Processing Systems*, 34: 22574–22587, 2021.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012. doi: 10.1109/IROS.2012.6386109.

Thomas J Walsh, Ali Nouri, Lihong Li, and Michael L Littman. Learning and planning in environments with delayed feedback. *Autonomous Agents and Multi-Agent Systems*, 18(1):83–105, 2009.

Mengdi Xu, Yikang Shen, Shun Zhang, Yuchen Lu, Ding Zhao, Joshua Tenenbaum, and Chuang Gan. Prompting decision transformer for few-shot policy generalization. In *International Conference on Machine Learning*, pp. 24631–24645. PMLR, 2022.

Qinqing Zheng, Amy Zhang, and Aditya Grover. Online decision transformer. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 27042–27059. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/zheng22c.html.

A ALGORITHM DETAILS

The overall algorithm of DeFogcould be summarized as in Alortithm 1, For the hyperparameters please refer to Appendix B.2

Algorithm 1 Decision Transformer under Random Frame Dropping (DeFog)

Require: update_interval {interval to update the drop-mask}
Require: n_updates, n_transitions {number of updates, number of transitions in the dataset}
Require: $\mathbf{sample}(\mathcal{S},\ n)$ {sample n elements uniformly from $\mathcal{S}$}
Require: $\mathbf{cumcount}$(flags) {count cumulative number of 0's before current position (inclusive)}
Require: $\mathbf{train}$(rtg, obs, act, timestep, dropspan), freeze_trunk {train the DT model}
Require: batch_size, act_buffer, obs_buffer, rtg_buffer, context_length
  for freeze_trunk $\in$ {**true**, **false**}, **do**
    $N \leftarrow$ n_updates
    while $N \neq 0$ **do**
      $M \leftarrow$ update_interval
      if freeze_trunk **then**
        $M \leftarrow \lfloor M/5 \rfloor$
      end if
      drop_mask $\leftarrow \mathbf{sample}(\{\textbf{true}, \textbf{false}\},$ n_transitions) {whether each frame is dropped}
      dropspans $\leftarrow \mathbf{cumcount}$(drop_mask) {duration of frames dropped}
      while $M \neq 0$ **do**
        selected_index $\leftarrow \mathbf{sample}(\{0, 1, ..., \text{n_transitions}\},$ batch_size)
        timestep $\leftarrow$ selected_index
        dropspan $\leftarrow$ dropspans[selected_index]
        dropped_index $\leftarrow$ selected_index $-$ dropspan
        rtgs $\leftarrow$ rtg_buffer[dropped_index: dropped_index + context_length]
        observations $\leftarrow$ obs_buffer[dropped_index: dropped_index + context_length]
        actions $\leftarrow$ act_buffer[selected_index: selected_index + context_length]
        $\mathbf{train}$(rtgs, observations, actions, timestep, dropspan, freeze_trunk)
        $M \leftarrow M - 1$
      end while
      $N \leftarrow N - 1$
    end while
  end for

B EXPERIMENT DETAILS

B.1 DATASETS AND SETUP

B.1.1 GYM MUJOCO

We use the D4RL dataset (Fu et al., 2020) that contains data collected by SAC agents. There are three datasets for each environment – expert, medium, and medium-replay. The expert dataset is collected by a fully-trained expert policy, while the medium dataset is collected by an agent about half the performance of the expert. Medium-replay includes the trajectories in a medium agent's buffer, and thus is the most diverse dataset with the lowest average return. Details of the different datasets are provided in Table 1.

B.1.2 ATARI

For atari game environments, we use the DQN Replay Dataset (Agarwal et al., 2020), which is collected from the replay buffer of a DQN agent during traing of these Atari games. Following the practice of the Decision Transformer, we only use a small portion of the dataset: 1% of the whole dataset, which is 500 thousand of the 50 million transitions observed by an online DQN agent. We define three kinds of datasets for each game as well – epert, medium, and expert-replay. The expert and the medium dataset are collocted from the collected data of a DQN agent's during the final

Dataset	No. Trajectories	No. Timesteps	Average Returns	Best Returns
Halfcheetah-Expert	1000	1000 000	10656.43	11252.04
Halfcheetah-Medium	1000	1000 000	4770.33	5309.38
Halfcheetah-Medium Replay	202	202 000	3093.29	4985.14
Hopper-Expert	1027	999 494	3511.36	3759.08
Hopper-Medium	2186	999 906	1422.06	3222.36
Hopper-Medium Replay	2041	402 000	467.3	3192.93
Walker2d-Expert	1000	999 214	4920.51	5011.69
Walker2d-Medium	1190	999 995	2852.09	4226.94
Walker2d-Medium Replay	1093	302 000	682.7	4132

Table 1: D4RL Gym MuJoCo Datasets Sizes and Returns

and medium training stages, while a expert-replay dataset is sampled evenly from the whold replay buffer.

B.2 Hyperparameter Settings

B.2.1 Gym Mujoco

For the gym MuJoCo environment, we use the same model architecture as the Online Decision Transformer. While the Online Decision Transformer uses different training parameters for each environment, we keep most of the training parameters the same among different environments

Hyperparameter	Value
Number of layers	4
Number of attention heads	4
Embedding dimension	512
Training context length K	20
Dropout probability	0.1
Activation function	ReLU
Gradient norm clip	0.25

(a) Architechture Parameters

Hyperparameter	Value
Learning rate	1e-4
Weight decay	1e-3
Batch size	256
Total training steps	1e5
Finetune training steps	2e4
Learning rate warmup steps	1e4
Drop-mask update interval	100

(b) Training Parameters

Table 2: Common Parameters for Gym Mujoco

For each dataset, we specify a target reward with it, and report the combination of train time drop-rate and finetuning drop-rate. The environment and dataset related paramenters are as follows:

Environment	Dataset	Target Reward	Training Drop Rate	Finetuning Drop Rate
HalfCheetah	Expert	12000	0.5	0.5
HalfCheetah	Medium	12000	0.8	0.8
HalfCheetah	Medium Replay	12000	0.8	0.8
Hopper	Expert	4000	0.9	0.9
Hopper	Medium	4000	0.5	0.8
Hopper	Medium Replay	4000	0.8	0.8
Walker2d	Expert	5000	0.9	0.9
Walker2d	Medium	5000	0.8	0.8
Walker2d	Medium Replay	5000	0.8	0.8

Table 3: Dataset Specific Parameters for Gym Mujoco

B.2.2 Atari

For the Atari environments, we use the same model architecture as the Decision Transformer since the Online Decision Transformer doesn't perform experiments on these environments. The hyper-parameters are as follows:

Hyperparameter	Value
Number of layers	6
Number of attention heads	8
Embedding dimension	64
Training context length K	30
Dropout probability	0.1
Activation function	ReLU
Gradient norm clip	1.0

(a) Architechture Parameters

Hyperparameter	Value
Learning rate	6e-4
Weight decay	0.1
Batch size	128
Total training steps	1e5
Finetune training steps	2e4
Learning rate warmup steps	1e4
Drop-mask update interval	1000

(b) Training Parameters

Table 4: Common Parameters for Atari

The environment and dataset related parameters are given in Table 5. Linear increasing frame drop rate means that the drop rate is gradually increased from start to end values.

Environment	Dataset	Target Reward	Training Drop Rate	Finetuning Drop Rate
Qbert	Expert Replay	14000	0.4	0.5
Seaquest	Expert Replay	1150	0–0.8 Linear Increase	0.8
Breakout	Expert Replay	90	0–0.8 Linear Increase	0.8

Table 5: Dataset Specific Parameters for Atari

C Supplementary Results

In this section, we provide more experimental results on the different components and settings of the DeFog model. As we aim to show the change in performance of the agent when the frame drop rate increases, results are given by plotting performance curves of average return against frame drop rate. To save space, for most of the figures we pick the 3 most representative datasets to show. The descriptions and analyses of the results will be based on all datasets.

C.1 Decision Transformer Backbone

Training a non-Decision Transformer Model on Masked-out Datasets DeFog simulates a frame dropping scenario by using a masked-out dataset with frames intentionally hidden away from the agent. This is tightly integrated with our drop-span embedding in the DeFog model because the drop-span information needs to be conveyed and encoded into the hidden representations. To determine the Decision Transformer architecture's contribution to DeFog's strong results in frame dropping scenarios, we perform an experiment using the TD3+BC Fujimoto & Gu (2021) baseline training on a masked-out dataset. We use a mask-out rate of 50%. We note that the DeFog curves shown in the results all use mask-out rates greater than 50%.

The results are shown in Figure 8. The TD3+BC trained with a masked-out dataset is able to achieve slightly better performance than the normal TD3+BC agent under higher frame drop rates in the Walker2d and Hopper environments. However, it collapses in the HalfCheetah environment. We observe similar patterns for the Medium and Medium-Replay datasets – a little bit of improvement in Walker2d and Hopper, and collapsing on HalfCheetah. While the average return is slightly improved by using a masked-out dataset, it is still nowhere close to the performance of DeFog. We believe this shows that while using a masked-out dataset is important and should help in frame dropping settings, it is the drop-span embedding that we add onto the Decision Transformer backbone which is the key contributor to strong performance under severe frame dropping.

Reconstruction of Frames During Training The Decision Transformer architecture is able to output three different types of tokens, corresponding to the action, state, and reward-to-go respectively. While the authors only let the model predict the real actions from the action tokens, it would be nice to have a grasp of the real state when the observation is dropped. With this motivation, we

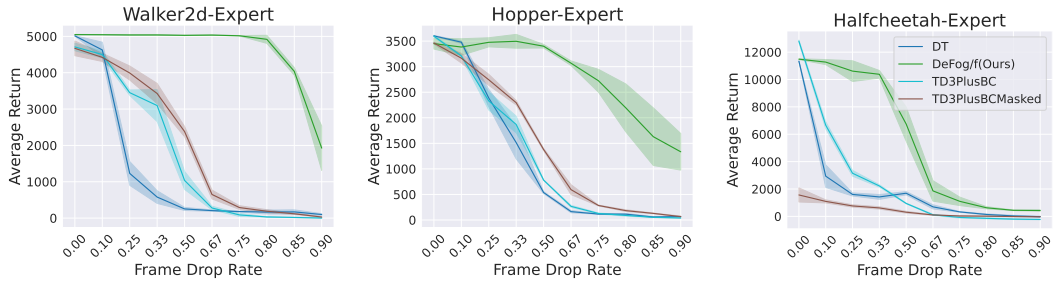

Figure 8: Ablation study on training a non-Decision Transformer based method with a masked-out dataset. "TD3PlusBC" is TD3+BC trained on a perfect uncorrupted dataset, while "TD3PlusBCMasked" denotes training a TD3+BC agent with a masked-out dataset.

conduct experiments to explore whether letting the model predict the current state or reward-to-go is beneficial to its performance. We evaluate the influence of making the model predict the states, the reward-to-goes, and both of them on all nine settings of the MoJuCo tasks, and report the results on three of them in Figure 9.

We find it a bit surprising that the model's performance deteriorates significantly in four out of the nine environments (HalfCheetah Expert, Walker2d Medium-Replay, Walker2d Expert, and HalfCheetah Medium-Replay) when applying state prediction alone. Only predicting the reward-to-go apart from the actions doesn't hinder the performance as much. And to predict both of them doesn't influence the performance in general. We reckon this may be due to the absence of supervision on the reward signal, which is exacerbated when the model is forced to predict both the state and action signal, In the original setting of predicting actions only and the new setting of predicting all three tokens, this kind of unbalance is not that intensified.

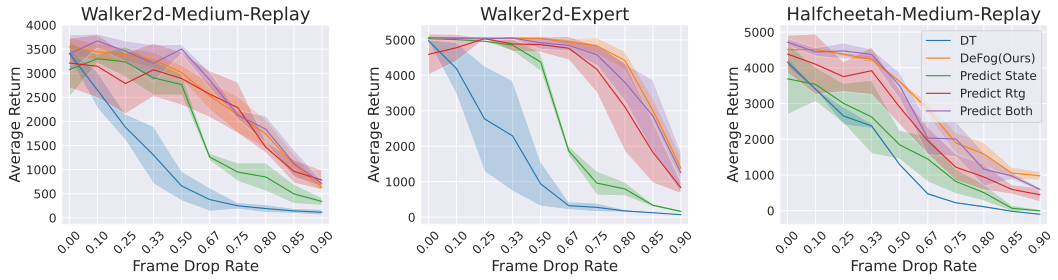

Figure 9: Ablation study on dropping the action together with the observation and reward-to-go.

C.2 Train-Time Frame Dropping

We conduct experiments exploring parts of the train-time frame dropping scheme, specifically the resampling of the drop-mask, the placeholder for dropped frames, the process of frame dropping, and the content of frames dropped.

Frame Dropping Mask DeFog samples a drop-mask that decides which frames in the dataset are simulated as dropped. There is a drop-mask update interval which DeFog periodically samples a new drop-mask. By doing so, DeFog can still take advantage of the full dataset and learn from the oracle policy. To further explore the learning ability of DeFog, we conduct the experiment with datasets in which more than 60% of the frames are dropped. During training, the dropped frames are never seen by Defog.

Results in Figure 10 show that the performance of DeFog is similar to using the full dataset and still outperforms other baselines, implying that DeFog is able to learn even when the dropped frames are never recovered. We note that on the medium-replay datasets of Halfcheetah and Hopper environments the performance degrades. We find this degradation due to the lack of training data. As shown

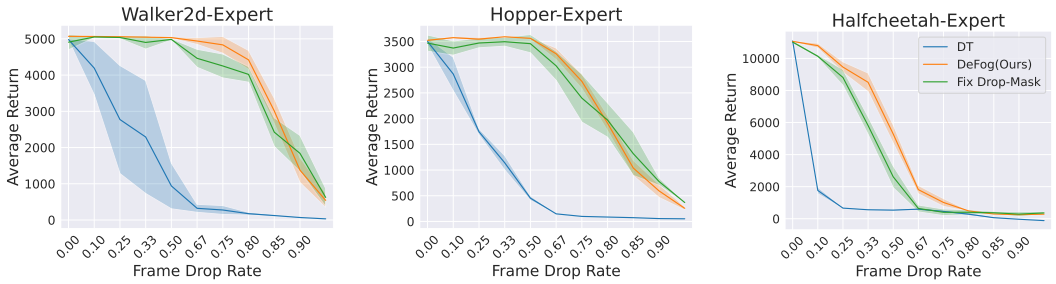

Figure 10: Ablation study on fixing the frame drop-mask for expert datasets. A similar performance to DeFog is achieved. "Fix Drop Mask" is fixing the drop-mask throughout training.

in Table 1, while the expert and medium datasets contain around 1M timesteps of data, the medium replay datasets have only 200k–400k timesteps. Given that we fix our drop-mask, only 40% of the data are available for the model. In the case of Halfcheetah-Medium Replay, the number of steps is only 8% of the full medium dataset.

Placeholder for Dropped Frames During train-time frame dropping, if a frame is marked as dropped, DeFog follows a simple and intuitive approach to replace both the observation and the reward-to-go of that frame with the most recent frame that was not dropped. We explore the following different replacements for the dropped frames:

- Adding noise to the dropped frames. This could be interpreted as stimulating the evolution of the unknown real states. For each noise step, we sample from a Gaussian distribution that is drawn from all the changes between consecutive observations from the dataset. When multiple frames are dropped, the Gaussian noises are added to form a new Gaussian distribution. We use a noise coefficient of both 0.1 and 0.5 to experiment on the scale of the noise.
- Replace the embedding of those replaced tokens to a specific learnable [MASK] token. We trial on 2 settings, one where the dropped observation and dropped reward-to-go share the same [MASK] token, and one where the two tokens are separate.
- Replace the dropped frames with zeros.

The results are presented in Figures 11 and 12.

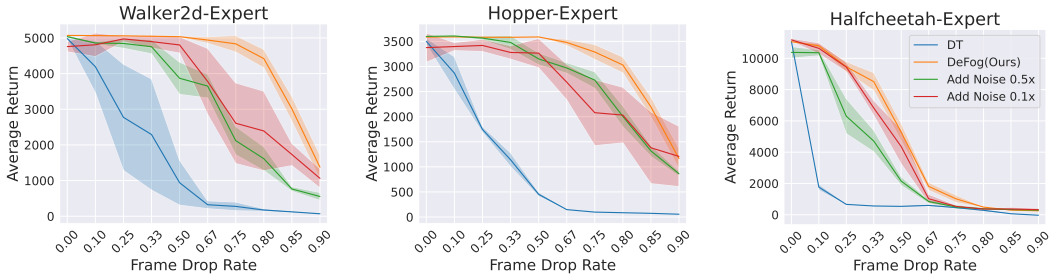

Figure 11: Ablation study on replacing the dropped frame with noise added onto itself. "Add Noise 0.1x" and "Add Noise 0.5x" denote adding noise onto the dropped frame with a coefficient of 0.1 and 0.5, respectively.

The results show that for adding noises, despite the perturbation in observation, the overall performance of our methods has no significant change. Neither of the 0.1 and 0.5 coefficients helped Defog's performance. We find that increasing the noise simply deteriorates performance, possibly due to the noisy observation distracting the agent. In datasets such as Hopper-medium and Walker-expert, the deterioration was more noticeable.

In the case where we replace the dropped frames with specific learnable [MASK] tokens, both settings give performances better than vanilla Decision Transformer but worse than DeFog. We

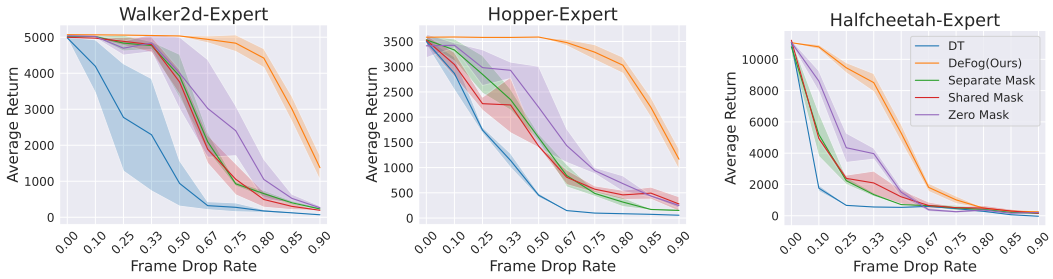

Figure 12: Ablation study on replacing the dropped frame with different kinds of [MASK] tokens. "Separate Mask" denotes that the observation and the reward-to-go do not share the same [MASK] token, while "Shared Mask" indicates that they do. "Zero Mask" is simply all zeros.

do not find this surprising as a single learnable mask cannot carry enough information for all the dropped frames, while the previous frame that DeFog uses would usually be similar to the current dropped frame.

Finally, replacing dropped frames with zeros does not perform much better than the vanilla Decision Transformer. As the zero token basically provides no information, this aligns with our expectation that it should perform worse than using the learnable [MASK] token as replacement for dropped frames.

Frame Dropping Process We use a drop rate p_d representing the probability for any single frame to be dropped, which results in a Bernoulli process for dropping frames where every frame is a single trial. To explore other kinds of drop-mask distributions, we go beyond the Bernoulli Process, and conduct experiments on the frame dropping being a Markov process. The probability of the next frame being dropped is no longer a constant value p_d, but instead follows the matrix $\mathbf{P}$

$$\mathbf{P} = \begin{bmatrix} 1 - p_1 & p_1 \\ 1 - p_2 & p_2 \end{bmatrix}$$

where given that the current frame was not dropped, the probability of the next frame to be dropped is p_1; if the current frame was dropped, then the probability of the next frame being dropped is p_2. The reason for choosing a Markov process is that it resembles the behavior of communication errors where frames are dropped chunk by chunk rather than frame by frame, thus simulating a typical category of realistic frame dropping issues. If $p_1 = p_2$, then the situation is identical to a Bernoulli process.

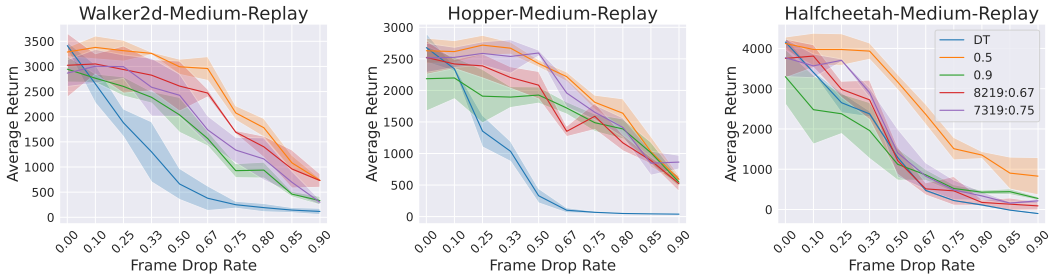

Figure 13: Ablation study on using a Markov Process for frame dropping. "0.5" and "0.9" represent using a Bernoulli Process of $p_d = 0.5$ and $p_d = 0.9$. "8219:0.67" denotes $p_1 = 0.2,\ p_2 = 0.9$, with a stationary distribution of frame dropping probability 0.67. Similarly, "7319:0.75" means $p_1 = 0.3,\ p_2 = 0.9$ with a stationary distribution of frame dropping probability 0.75.

Our experimental results, given in Figure 13, show that when comparing the Bernoulli to the Markov, the p_d in Bernoulli is highly correlated with the p_2 in Markov. We find this result to abide with the fact that in a frame dropping setting, the moments where frames are dropped affect the overall performance more. If we fix p_2 and change p_1, we find that in general the less p_1 is, the better the

performance. This is not surprising as p_1 decreasing would imply that there are more timesteps of consecutive undropped frames where the agent can make better decisions.

Dropping the Action In the training of DeFog, after a drop-mask is sampled, frames are dropped accordingly to the mask. Specifically, the observation and the reward-to-go of a frame are dropped, while the action is not, as actions are determined by the agent. We perform an experiment where actions are masked out alongside the states and reward-to-goes, and results show that the performance is negatively affected. Results are shown in Figure 14.

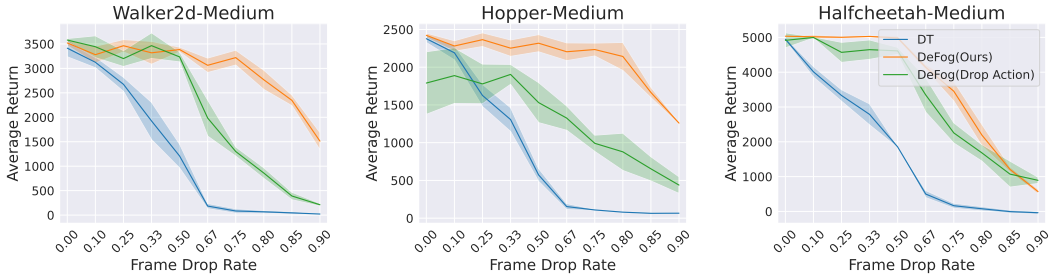

Figure 14: Ablation study on dropping the action together with the observation and reward-to-go.

C.3 Drop-Span Embedding and Freeze Trunk Finetuning

Explicit Drop-Span Encoder and Finetuning Figure 15 are the full ablation results of Figure 7, containing performances of the ablated models on all 9 datasets. "w/o freeze" denotes removing the freeze trunk finetuning, and "w/o drop" means we use the implicit embedding method instead.

The use of explicit drop-span embedding was able to improve performance over implicit embedding by a huge margin in these 4 datasets: Walker2d-Expert, Hopper-Medium, Walker2d-Medium-Replay, and Hopper-Medium-Replay. For the other 5 datasets, using implicit embedding all led to harm in performance as well, though not so significant. In addition, in 5 of the 9 datasets (Walker2d-Expert, Walker2d-Medium, Walker2d-Medium-Replay, Hopper-Medium, Hopper-Medium-Replay), performance was especially improved on high frame drop rates such as 0.8, 0.85, 0.9. The result shows that drop-span information is important and should be explicitly encoded into the model.

Removing Drop-Span and Timestep Embeddings Both the explicit drop-span encoder and the implicit embedding try to convey the drop-span information to the agent. We also conduct experiments on totally removing this piece of information, by using a normal timestep embedding without any other kind of drop-span embedding. The agent no longer receives information on where and how many frames are dropped. We then perform the experiment of removing the timestep embedding. Under this setting, we keep the explicit drop-span encoder. As mentioned above, the explicit drop-span encoder only gives information on the number of past timesteps before which the frame was observed, not the actual timestep information.

The results are given in Figure 16, and performance is degraded upon both of the embeddings' removal. We find drop-span embedding to be the key factor in DeFog. We believe this shows that the actual useful information leveraged by a DeFog agent is **the number of past timesteps before** a frame was observed, not **the timestep** it was observed. The longer the drop-span of the frame, the less it should be considered in the action prediction process, and the action history could be a better reference for decision-making. From the similarity of implicit vs. no embedding scores we also find that useful numbers like the drop-span need to be explicitly given, and performance would be hindered even if the agent can work out the number by simple arithmetic.

Meanwhile, the removal of the timestep embedding does not cause a severe drop in performance. Under non-frame-dropping conditions, the Online Decision Transformer also conducted the experiments of removing timestep embeddings and found that performance was not heavily affected. As suggested by Zheng et al. (2022), this could be due to the timestep information deduced from the reward-to-go value, making the lack of timestep embedding no longer fatal.

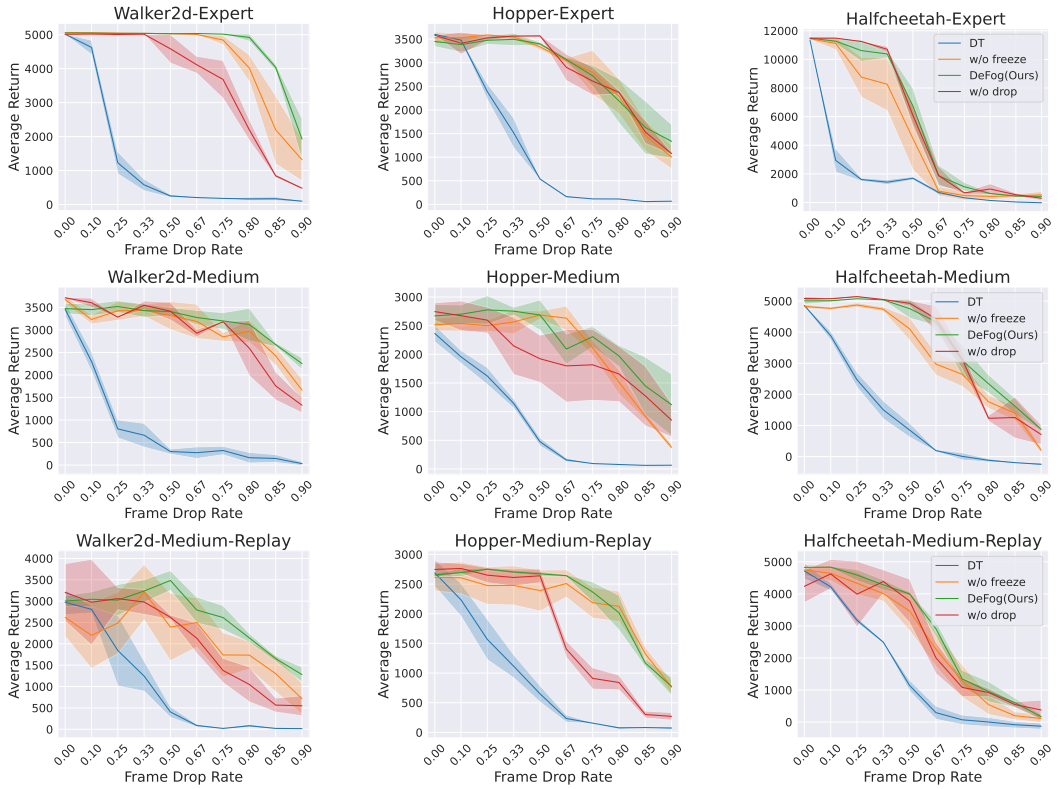

Figure 15: Ablation results on the explicit drop-span encoder and freeze trunk finetuning in all 9 gym MuJoCo environments. The label "w/o freeze" stands for without freeze trunk finetuning, while "w/o drop" denotes using the implicit embedding.

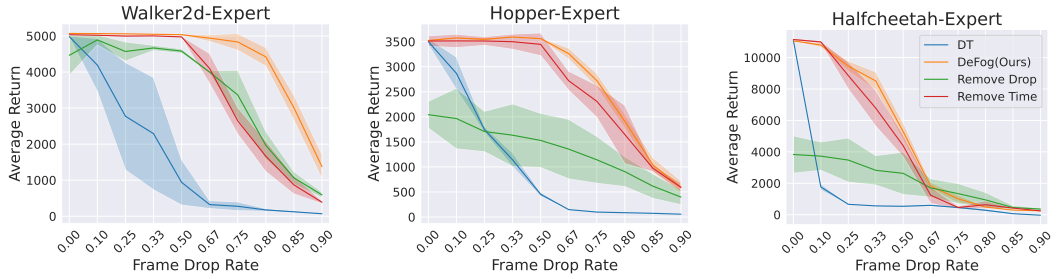

Figure 16: Ablation study on removing the drop-span information. "Remove Drop" denotes removing the drop-span embedding without using implicit embedding, still keeping the timestep embedding; "Remove Time" indicates removing the timestep embedding, still keeping the explicit drop-span embedding.

Finetuning Individual Components DeFog currently finetunes the action predictor and drop-span encoder. For a better understanding of the finetuning stage and the specific parts, we conduct experiments on these components individually. The results are given in Figure 17.

We find that finetuning only the drop-span encoder gives slightly better performance on Walker2d-medium-replay, Walker2d-expert, and HalfCheetah-medium-replay datasets. Meanwhile, on the other datasets, for example the Hopper medium-replay, finetuning both the drop-span encoder and the action predictor led to better results. In general, neither way of finetuning significantly outperforms its counterparts. We believe this is understandable as the action predictor and the drop-span encoder are both key components of the DeFog model. The ablation results on the drop-span encoder show that how well the drop-span information can be leveraged by the model heavily affects

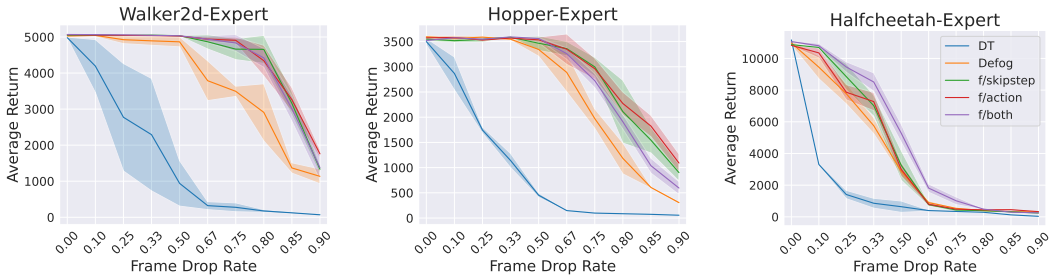

Figure 17: Ablation study on individually finetuning the components of DeFog. "DeFog" denotes not finetuning anything. "f/skipstep", "f/action", "f/both" stand for finetuning the drop-span encoder, the action predictor, and both, respectively.

the performance. It is also shown that the quality of the action predictor also directly determines the ability of the agent.