2024 33rd IEEE International Conference on
Robot and Human Interactive Communication (RO-MAN)
August 26-30, 2024. Pasadena, California, USA.

2024 33rd IEEE International Conference on Robot and Human Interactive Communication (ROMAN) | 979-8-3503-7502-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/RO-MAN60168.2024.10731194

D3D: Conditional Diffusion Model for Decision-Making Under Random Frame Dropping

Bo Xia[1], Yifu Luo[1], Yongzhe Chang[1], Bo Yuan[2], Zhiheng Li[1] and Xueqian Wang[1]

***Abstract*— The occurrence of frame drops due to issues such as corrupted communications or malfunctioning sensors presents a significant challenge to an agent's decision-making, especially in remote control scenarios. Classical reinforcement learning (RL) usually assumes a continuous data stream without frame drops and relies heavily on online interactions, which is time-consuming, resource-intensive, and often impractical in certain scenarios. Consequently, the performance of RL may deteriorate significantly in face of non-negligible frame drops. To tackle this challenge caused by frame dropping, We propose Conditional Diffusion Model for Decision-Making under Random Frame Dropping (D3D), an offline algorithm that can effectively enhance performance robustness in frame dropping scenarios. D3D addresses this issue through a two-phase approach: 1) During the policy generation phase, D3D adopts a return-conditional diffusion model for decision making rather than the temporal difference learning, whose policy is derived using offline datasets of return-labeled trajectories without information loss. 2) When frame dropping occurs during evaluation, D3D seamlessly substitutes the missing state with its corresponding prediction in the horizon made by the diffusion model. Extensive experiments are conducted on MuJoCo and Adroit tasks to validate D3D's robustness and efficiency. The results demonstrate that D3D consistently outperforms state-of-the-art RL algorithms, especially excelling on tasks featuring severe drop rates.**

I. INTRODUCTION

Deep reinforcement learning (RL) has witnessed significant advancements in the field of artificial intelligence. Policies trained using RL have surpassed human performance in various domains, such as games [1], large language models [2], and control systems [3]. The foundation of these RL algorithms lies in the framework of the Markov Decision Process (MDP), which means that the agent's current action is solely dependent on the current state and is independent of any other information or context. However, as shown in Fig. 1, it's quite common to encounter the loss or delay of observations due to network malfunctions or sensor failures [4][5], especially in remote control. Unfortunately, this phenomenon may disrupt the Markov property, resulting in decreased agent performance [6]. How to maintain the agent's original performance despite frame drops and ensure its robustness against different drop rates remains a challenging problem.

[1] Bo Xia, Yifu Luo, Yongzhe Chang, Zhiheng Li and Xueqian Wang are with Tsinghua Shenzhen International Graduate School, Shenzhen, 518071, China xiab21@mails.tsinghua.edu.cn, 13810337901@163.com, cyz1988havefun@gmail.com, zhhli@tsinghua.edu.cn

[1] Xueqian Wang is with Tsinghua Shenzhen International Graduate School, Shenzhen, 518071, China wang.xq@sz.tsinghua.edu.cn

[2]Bo Yuan is with the Research Institute of Tsinghua University in Shenzhen, Shenzhen 518057, China boyuan@ieee.org

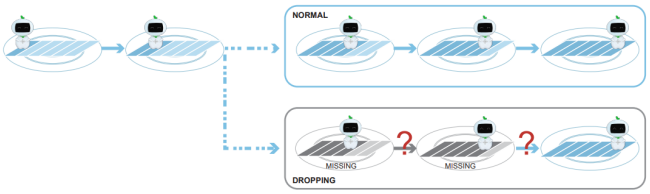

Fig. 1. An illustration of random frame dropping: Agents typically make decisions based on observed states, but frame drops disrupt state observation, hampering the decision-making.

This challenge has been a long-standing concern within the RL community. Some researchers focused on constant or random delay scenarios, attempting to address the issue by transforming the delayed MDP into a delay-free MDP through state augmentation techniques [7][8][9] or by predicting delayed states using delayed information, such as received states and corresponding action sequences [10][11][12]. However, these methods introduce additional computational complexity or cumulative errors. They also require extensive online interactions with the delayed environment, consuming substantial time and resources, which can be unfeasible in real-world tasks. Another approach, known as Defog [6], tackles a more severe issue: complete state dropout. It employs a decision-transformer style autoregressive model to generate policies based on offline datasets. While Defog maintains its performance in scenarios with random frame drops, it neglects the well-trained model in scenarios without frame drops and requires the tuning of a significant number of parameters.

To mitigate the issue of frame drops, we propose Conditional Diffusion Model for Decision-Making under Random Frame Dropping (D3D). This offline algorithm is purposefully designed to be robust against frame dropping with the help of return-labeled offline datasets. The conditional diffusion model in D3D serves as both the decision and prediction module. Specifically, during policy generation, D3D employs classifier-free guidance with low-temperature sampling to obtain return-maximizing trajectories. Additionally, the inverse dynamics model is trained concurrently using a supervised approach, where it takes two adjacent states as inputs and generates corresponding actions as outputs. Both models are based on the original offline datasets without frame drops. In the evaluation phase, D3D seamlessly replaces the dropped state with its corresponding prediction within the horizon of its preceding state when the frame drop occurs. Subsequently, it utilizes the same supervision model to generate new actions.

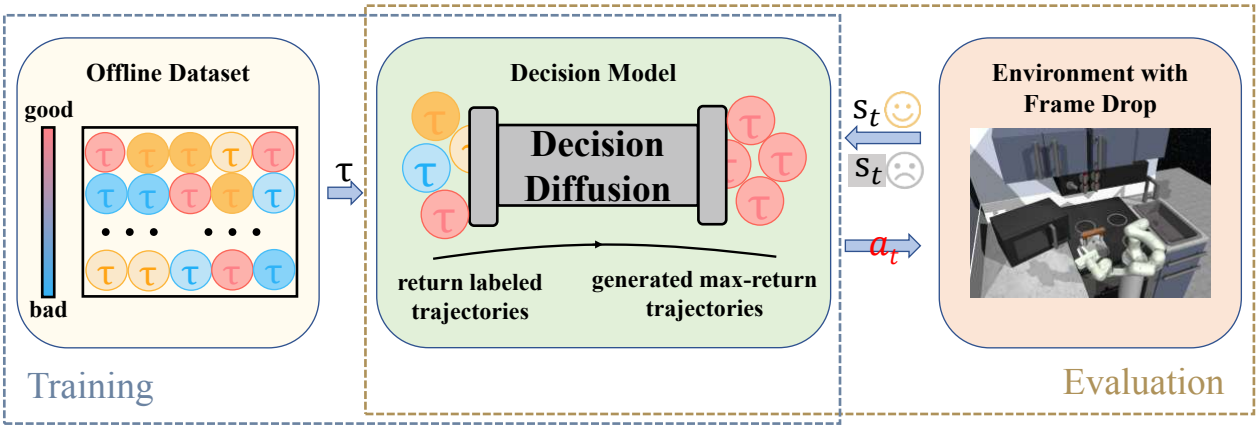

Fig. 2. Flowchart of D3D: This process comprises two stages - training and evaluation. In training, D3D utilizes a conditional diffusion model to generate return-maximizing trajectories and simultaneously trains an inverse dynamics model, both based on return-labeled offline datasets without frame drops. During the evaluation phase, when states are dropped (which are shaded in the figure), D3D seamlessly replaces them with predictions from the diffusion model and derives actions using the inverse dynamics model.

The major contributions of our work can be summarized as the following three aspects: 1) Introduction of a new two-stage algorithm, D3D, to enhance agent robustness under random frame drops; 2) Seamless transfer of well-trained models from Decision Diffusion [13] to D3D with the only difference in the evaluation phase; 3) Extensive experiments across various tasks, including MuJoCo and Adroit environments, to demonstrate D3D's superiority over state-of-the-art RL algorithms in terms of robustness and efficiency.

II. RELATED WORK

A. Control under delay and frame drop

The loss or delay of states caused by network malfunctions or sensor failures significantly impacts an agent's decision-making process, drawing the attention of the RL community for many years.

Research on state or action delays can be broadly categorized into two main approaches. On the one hand, attempts have been made to transform the delayed MDP into a delay-free MDP by reconstructing the state space and reward function. For instance, [7], introduced the concept of Random Delay Markov Decision Process (RDMDP) and demonstrated the theoretical feasibility of solving this problem using state augmentation techniques. In a practical implementation, [8], applied vanilla DQN algorithms to address this issue in simple environments. Moreover, [9] enhanced sample efficiency by relabeling past actions with the current policy to generate actual on-policy sub-trajectories based on the new MDP. On the other hand, the delayed state is predicted by incorporating delay-related information, which includes both received states and action sequences. Specifically, [10], employed a random forest model for the prediction of delayed states, while [11] proposed a perspective that the dynamics of the delayed MDP comprise two distinct components: one attributed to delays and the other originating from the underlying delay-free MDP. Furthermore, [12] developed Q-learning based algorithms that integrate a transition forward model, serving as a predictive tool for delayed states.

The exploration of frame dropping was initiated by [6], and this problem is considered more challenging than delay-related issues, particularly due to the complete absence of certain states. [6] is fundamentally based on the decision transformer architecture [14], incorporating a parameter designed to represent the frequency of frame drops.

The previously mentioned delay-related research primarily involves extensive online interactions with the environments, which are viewed as time-consuming and resource-intensive. In contrast, Defog modifies the original framework, resulting in the abandonment of well-trained models from scenarios without frame drops, and necessitates significant parameter tuning even for tasks that have been previously completed. With access to the return-labeled offline datasets, our work addresses the challenge of frame drops. D3D is an offline algorithm and overcomes the challenges associated with the deadly triad in temporal difference learning, capitalizing on the strengths of the diffusion model. Furthermore, it can seamlessly utilize well-trained models from drop-free scenarios, and preserve their superior performance.

B. Development of diffusion models and their application in decision-making

The reverse diffusion process has been introduced as a highly flexible and tractable generative data model [15]. Its effectiveness in data sampling has gained widespread recognition due to its iterative denoising process [16]. Recent developments have dedicated in the connection between diffusion models and score-based generative models [17][18]. This innovative approach models data distribution by leveraging gradients and optimizing the score matching objective. Building on this significant advancement, recent studies have explored various classifiers to enable conditional sampling based on additional information, such as text, exemplified by implicit classifier guidance [19].

Diffusion models have also found applications as observation-to-action models for imitating human behaviors [20] and representing policies in offline RL. Furthermore, more research has focused on directly addressing the challenge of sequential decision-making using diffusion models. [13] formulates a policy as a return-conditional diffusion model, streamlining many complexities inherent in traditional offline RL and showing promise in trajectory generation and offline sequential decision-making. However, research has been lacking regarding frame drops from the perspective of the diffusion model. Our approach, D3D, offers a robust solution to this problem and demonstrates superior performance compared to traditional offline RL methods.

III. METHOD

In this section, we will commence by describing the problem setting related to the frame dropping, followed by a detailed introduction to Conditional Diffusion Model for Decision-Making under Random Frame Dropping (D3D) from two aspects: policy generation and evaluation, as shown in Fig. 2.

A. Problem statement

Frame drops in sequential decision-making disrupt the Markov property inherent in the MDP, giving rise to a new decision process known as the Random Dropping Markov Process (RDMDP) as referenced in [6]. The RDMDP can be correspondingly defined as a tuple $(S, A, r, \rho, \gamma, \mu)$, where S and A denote the state and action spaces, respectively; $r(s_t, a_t)$ is the instant reward function; $\rho(s_0)$ represents the initial state distribution; γ is the discounted factor; μ is the probability of the frame drop.

At each time step t, the drop flag d_t follows a Bernoulli distribution: $d_t \sim Bern(\mu)$. Correspondingly, the state and the reward received by the agent can be defined as:

$$\hat{s}_t = \delta(d_t) s_t, \tag{1}$$

$$\hat{r}_t = \delta(d_t) \sum_{k=t_f}^{t} r_t, \tag{2}$$

where $\delta(\cdot)$ represents the Dirac delta function and t_f denotes the last time the state is observed. The reward $\hat{r}_t$ can be further explained as follows: when the state is dropped, the agent receives nothing, and the instant rewards accumulate in the remote system simultaneously; when the state returns to normal, the agent receives cumulative rewards first, followed by instant rewards.

B. Policy generation

We employ a return-conditional diffusion model for decision making to generate policies from return-labeled offline datasets without frame drops. This model, which eliminates temporal differences' instability and distribution shifts, consists of two sub-modules: one for generating return-maximizing trajectories and the other for generating actions with inverse dynamics. The loss function includes both of these modules.

1) Generating return-maximizing trajectories: The sequential decision-making is formulated as the standard problem of conditional generating modeling:

$$max_\theta \mathbb{E}_{\tau \sim D}[\log p_\theta(\boldsymbol{x_0}(\tau) | \boldsymbol{R}(\tau))], \tag{3}$$

where τ is the trajectory sampled from the offline dataset D, and $\boldsymbol{R}(\tau)$ is the return of the whole trajectory. We aim to derive a conditional data distribution p_θ from Equation 3 to generate more trajectories $\boldsymbol{x_0}(\tau)$ guided by the return $\boldsymbol{R}(\tau)$. Moreover, the generative model is constructed as:

$$\begin{aligned} q(\boldsymbol{x_{k+1}}(\tau) | \boldsymbol{x_k}(\tau)), \\ p_\theta(\boldsymbol{x_k}_{\ 1}(\tau) | \boldsymbol{x_k}(\tau), \boldsymbol{R}(\tau)), \end{aligned} \tag{4}$$

where q is the forward noising process and p_θ represents the reverse denoising process.

In our work, we only diffuse over states in the same way as [18], with the consideration of the continuity of states and the dispersion and unsmoothness of actions. Therefore, the diffused object is defined as:

$$\boldsymbol{x_k}(\tau) = (s_t, s_{t+1}, \cdots, s_{t+H-1})_k, \tag{5}$$

where $k \in [0, K]$ is the timestep in the forward process, t means the t-th state in the trajectory τ and $\boldsymbol{x_k}(\tau)$ is considered as the different level of noise state sequences of length H. During the forward process, $\boldsymbol{x_0}(\tau)$ represents the real states from the trajectory, while $\boldsymbol{x_1}(\tau), \cdots, \boldsymbol{x_K}_{\ 1}(\tau)$ are latent variables, and $\boldsymbol{x_K}(\tau)$ follows a Gaussian distribution $N(\mathbf{0}, \boldsymbol{I})$ within carefully selected parameters and a sufficiently large K. The diffusion model is optimized using $\epsilon_\theta(\boldsymbol{x}_k, k)$ rather than $\log p_\theta$ due to their equivalence and simplicity. The optimization process aims to estimate the noise $\epsilon \sim N(\mathbf{0}, \boldsymbol{I})$ added to the real states $\boldsymbol{x}_0$ to produce noise $\boldsymbol{x}_k$. Besides, the classifier-free guidance predicts the noise by learning both a conditional $\epsilon_\theta(\boldsymbol{x_k}(\tau), \boldsymbol{R}(\tau), k)$ and an unconditional $\epsilon_\theta(\boldsymbol{x_k}(\tau), \varnothing, k)$ with minimal modifications to the original training setup.

The term $\boldsymbol{R}(\tau)$ in Equation 4 imposes a constraint condition on the reverse denoising process, compelling the model to generate behaviors conforming to the constraint, even in the presence of sub-optimal trajectories in the datasets. Futhermore, we utilize classifier-free guidance with low-temperature sampling to address the limitations of dynamic programming. This approach involves the sampling of $\boldsymbol{x_0}(\tau)$ starting from Gaussian noise $\boldsymbol{x_K}(\tau)$ and iteratively refining it into $\boldsymbol{x_k}_{\ 1}(\tau)$ from K to 0 using perturbed noise:

$$\epsilon_\theta(\boldsymbol{x_k}(\tau), \varnothing, k) + \omega(\epsilon_\theta(\boldsymbol{x_k}(\tau), \boldsymbol{R}(\tau), k) - \epsilon_\theta(\boldsymbol{x_k}(\tau), \varnothing, k)), \tag{6}$$

where the scalar ω is purposefully designed to optimize and select the most favorable trajectories based on the condition $\boldsymbol{R}(\tau)$. Low-temperature sampling implies that, during the denoising process, the mean of Gaussian sampling, μ_k, calculated at each step is directly used to obtain while the variance Σ_k is incorporated only after applying α weighting ($\alpha \in [0, 1)$).

Furthermore, returns in the offline datasets should be normalized to ensure that $\boldsymbol{R}(\tau)$ lies within the range $[0, 1)$,

signifying that sampling a high-return trajectory is equivalent to conditioning on $\boldsymbol{R}(\tau) = 1$.

2) Action: An action a_t has the capability to transition the state s_t to the subsequent state s_{t+1}, suggesting that the action a_t can be inferred from the adjacent state s_t and s_{t+1}. Starting from the current state s_t, we can employ the diffusion model to sample and predict the subsequent state s_{t+1} that maximizes returns. As a result, the action is generated by the inverse dynamics model:

$$a_t = f_\phi(s_t, s_{t+1}), \tag{7}$$

where f_ϕ is the inverse dynamics with parameters ϕ.

3) Loss function: We design the loss function as follows:

$$\begin{aligned} L =& \mathbb{E}_{\tau,k,\beta}[||\epsilon - \epsilon_\theta(\boldsymbol{x}_k(\tau), (1-\beta)\boldsymbol{R}(\tau) + \beta\varnothing, k)||^2] + \\ & \mathbb{E}_{(s,a,s')\in D}[||a - f_\phi(s, s^{'})||^2], \end{aligned} \tag{8}$$

where $\tau \in D$ represents the trajectory, $k \sim U[1, \cdots, K]$ is the timestep in the diffusion model, $\beta \sim$ Bern(p) means that the conditional information would be neglected with probability p and $\epsilon \sim N(\mathbf{0}, \boldsymbol{I})$ denotes the sampled noise. Therefore, the reverse denoising process and the inverse dynamic model are trained simultaneously.

C. Evaluation

In situations involving frame drops, the agent may face challenges in making decisions when encountering state drops. Nevertheless, at each time step, D3D generates new data denoted as $\boldsymbol{x}_0(\tau)$ with the primary objective of maximizing trajectory returns. Within $\boldsymbol{x}_0(\tau)$, there are H states, enabling the estimation of the dropped state.

To be more specific, two methods are employed for estimating the state of the dropped state:

- Iterative estimation: The estimation of each dropped state is obtained from the $\boldsymbol{x}_0(\tau)$ generated by the preceding state, without consideration of whether the previous state is real or estimated. In other words, if s_t is dropped at time t, we extract s_t from the$\boldsymbol{x}_0(\tau)$ generated by s_{t-1} and replace it, regardless of whether s_{t-1} is a real or estimated state.
- Direct estimation: For the dropped state, we directly select the corresponding estimate from $\boldsymbol{x}_0(\tau)$ generated based on the last observed real state. If the number of frame skips exceeds the horizon of $\boldsymbol{x}_0(\tau)$, a new $\boldsymbol{x}_0(\tau)$ is generated starting from the last state in $\boldsymbol{x}_0(\tau)$, and its corresponding estimate is extracted from this new $\boldsymbol{x}_0(\tau)$. In essence, when the most recently observed true state is s_t, the estimation of s_{t+d} is directly obtained from $\boldsymbol{x}_0(\tau)$ derived from s_t. If $d > H-1$, we generate a new $\boldsymbol{x}_0(\tau)$ using s_{t+H-1} from the old $\boldsymbol{x}_0(\tau)$ and then extract the estimate from the new $\boldsymbol{x}_0(\tau)$.

The comparison of two methods, termed "D3D_iterative" and "D3D_direct", and analysis are shown in Experiment IV-B, and the pseudocode is provided in Algorithm 1.

Algorithm 1 Evaluation of D3D

Input: Noise model ϵ_θ, inverse dynamics f_ϕ, guidance scale ω, history length C, return $\boldsymbol{R}(\tau)$, variance scale α

Initialize $h \leftarrow$ Queue $(length = C)$, $t \leftarrow 0$
state s_0 = env.reset()
$\boldsymbol{x_0}(\tau)$ = PREDICT $(s_0, \boldsymbol{R}(\tau), h, \alpha, K)$
while not done **do**
  Extract (s_t, s_{t+1}) from $\boldsymbol{x_0}(\tau)$
  Execute $a_t = f_\phi(s_t, s_{t+1})$
  state s_{t+1}, reward r, done = env.step(a_t)
  if s_{t+1} is None **then**
    Estimate s_{t+1} from $\boldsymbol{x_0}(\tau)$, using iterative estimation and direct estimation in III-C
  end if
  h.insert(s_{t+1})
  $\boldsymbol{x_0}(\tau)$ = PREDICT$(s_{t+1}, \boldsymbol{R}(\tau), h, \alpha, K)$
  $t \leftarrow t+1$
end while

function PREDICT(state s, return $\boldsymbol{R}(\tau)$, h, variance scale α, K)
  Initialize $\boldsymbol{x_K}(\tau) \sim N(\mathbf{0}, \alpha\mathrm{I})$
  for $k = K$ to 1 **do**
    $\boldsymbol{x_k}(\tau)$ [: length (h)] $\leftarrow h$
    $\hat{\epsilon} \leftarrow \epsilon_\theta(\boldsymbol{x_k}(\tau), k)$ + $\omega(\epsilon_\theta(\boldsymbol{x_k}(\tau), \boldsymbol{R}(\tau), k)$ - $\epsilon_\theta(\boldsymbol{x_k}(\tau), k))$
    $(\mu_{k-1}, \Sigma_{k-1}) \leftarrow Denoise(\boldsymbol{x_k}(\tau), \hat{\epsilon})$
    $\boldsymbol{x_{k\ 1}}(\tau) \sim N(\mu_{k-1}, \alpha\Sigma_{k-1})$
  end for
  return $\boldsymbol{x_0}(\tau)$
end function

IV. EXPERIMENT

In this section, we assess the effectiveness and robustness of D3D across a range of decision-making tasks. This part is structured as follows: (1) We commence by introducing the experimental setup, such as the description of different tasks, the sources and composition of datasets, and the hyperparameter used in the model; (2) We conduct comparative experiments involving both offline and online state-of-the-art algorithms, with a focus on various tasks featuring different drop rates. Subsequently, we delve into an in-depth analysis of the results.

A. Experiment setup

Experiments are conducted across six continuous control tasks including ant, hopper, and walker in the MuJoCo environment [21], as well as door, pen, relocate in tasks using the Adroit Hand [22], as depicted in Fig. 3. In MuJoCo, the objective of the tasks involves controlling the agent to achieve forward locomotion by exerting torques on the joints that connect its body parts. Additionally, in Adroit, the tasks revolve around specific goals, namely, opening a door (door), aligning a pen with a target orientation (pen), and relocating a ball to a target position (relocate). All tasks are simulated

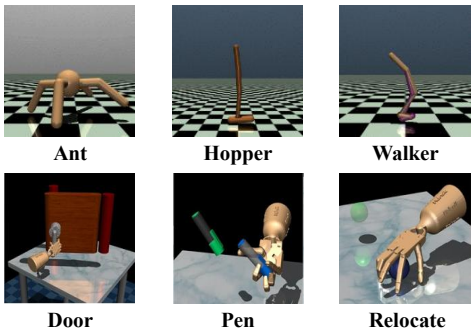

Fig. 3. Simulation environments.

TABLE I

ACTION AND STATE DIMENSIONS IN DIFFERENT TASKS

Environment	State dimension	Action dimension
ant	111	8
hopper	11	3
walker	17	6
pen	45	24
door	39	28
relocate	39	30

using the MuJoCo physics engine. A detailed overview of action and state dimensions across these environments is shown in Table I. It's worth noting that Adroit tasks are generally considered more challenging, as indicated by their more complex dynamics.

The offline datasets we used are from D4RL [23], an open-source benchmark for offline reinforcement learning. Given the increased complexity of frame drops problem within the context of offline reinforcement learning, our dataset selection is conducted meticulously. Specifically, in the MuJoCo environment, we choose Expert, Medium-Expert, and Medium datasets, where the ratio of the expert trajectories accounts for 100%, 50%, and 33%, respectively. In the Adroit environment, we choose Expert and Cloned datasets, where the ratio of the expert trajectories accounts for 100% and 50%.

Additionally, the diffusion model in D3D utilizes a temporal U-Net architecture, which is a neural network composed of iterated convolutional residual blocks. The architecture and hyperparameter used in D3D are provided in Table II.

B. Comparison experiments

To validate the effectiveness and robustness of D3D, we conduct separate comparisons with state-of-the-art RL algorithms with frame drop rates from 0 to 0.8. In cases where D3D is compared to offline reinforcement algorithms, we apply masking to the offline datasets to align with the frame drop rates before the start of training. In comparison with online reinforcement learning, training is performed directly within an environment featuring frame drops. We train each experiment with three different training seeds. Each curve in the figures represents the average return from 10 runs conducted during the testing phase, with the shaded region indicating the 95% confidence interval.

TABLE II

HYPERPARAMETERS USED IN DIFFERENT TASKS

	Hyperparameter	Value
U-Net Architecture Parameters	Number of blocks	6
	Number of convolutions	2
Dimension of Vectors	Timestep	128
	Condition embeddings	128
Inverse Dynamics Architecture Parameters	Number of layers	3
	Number of hidden units	1024
	Activation function	$Relu$
Training Parameters for ϵ_θ and f_θ	Learning rate	$2e^{-4}$
	Batch size	32
	Train steps	$2e^6$
Planning Horizon in Different Tasks	MuJoCo	100
	pen	20
	door	40
	relocate	40
Max-Episode Length	MuJoCo	1000
	pen	100
	door	200
	relocate	200
Other Parameters	Diffusion steps K	200
	Guidance scale s	1.2
	Variance scale α	0.5
	Context length C	20

The compared algorithms are as follows:

- **Decision Transformer under Random Frame Dropping (DeFog) [6].** Building upon the concept and framework of Decision Transformer, Defog introduces an additional variable to account for the number of frame drops in order to tackle the issue. Due to the absence of open-source code, we manually implement the algorithm and conduct comparative analysis.
- **Decision Transformer (DT) [14].** DT, categorized as an offline RL algorithm, leverages an autoregressive model and formulates its policy optimization by considering the entire trajectory through a Transformer architecture.
- **Conservative Q-learning (CQL) [24].** CQL is the state-of-the-art in offline RL which learns a conservative Q-function to overcome the problems caused by distribution shift.
- **Reinforcement learning with random delays (RLRD) [9].** RLRD is an online RL algorithm that uses state augmentation techniques to tackle random delay issue, with a predefined maximum delay limit. In our comparative experiments, when the frame drop rate is zero, we configure RLRD with no delays for a fair evaluation. For other frame drop rates, we set the state delay to its maximum value and substitute the dropped state with the most recently received state.
- **Twin-delayed DDPG (TD3) [25].** TD3 is a commonly used online RL algorithm for continuous control tasks. We initially train the policy in environments without frame drops and then freeze the network parameters for direct application in scenarios with different frame loss rates.

In Figures 4 and 5, each subplot is titled according to the task and the offline dataset used. It should be noted that

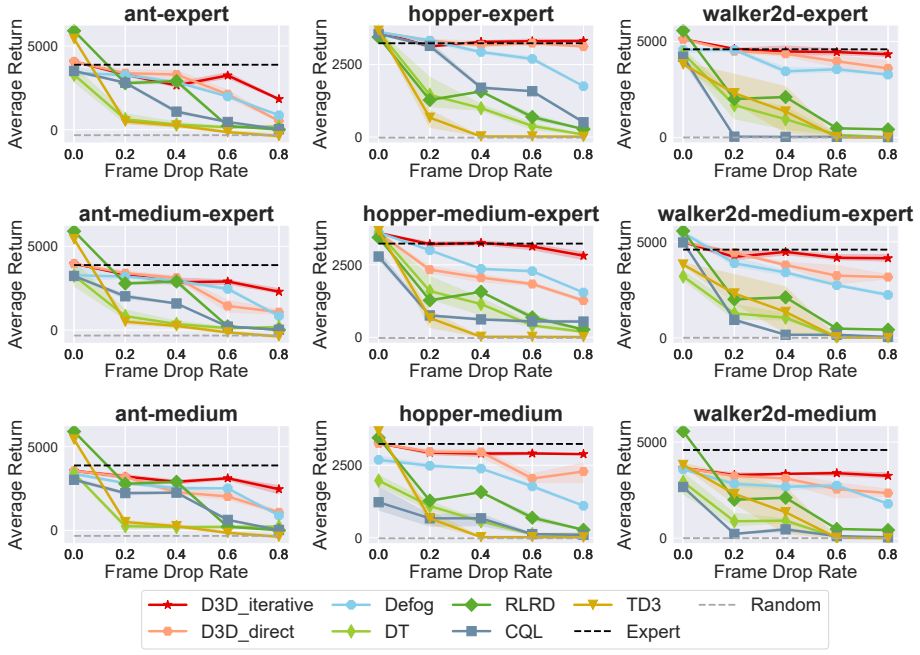

Fig. 4. Comparison experiments in the MuJoCo environments with frame drop rate set to 0, 0.2, 0.4, 0.8.

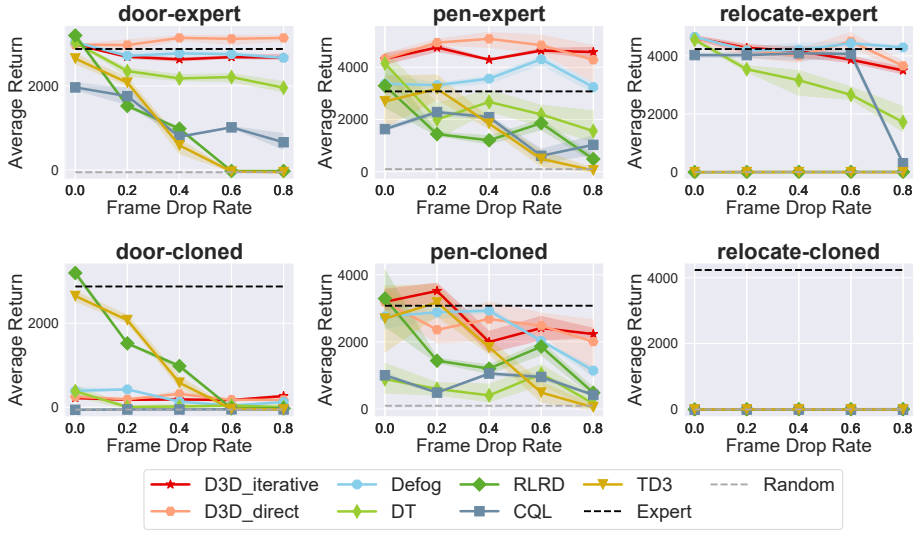

Fig. 5. Comparison experiments in the Adroit environments with frame drop rate set to 0, 0.2, 0.4, 0.8.

RLRD and TD3 are online algorithms, so their performance does not change with the dataset in the same task.

From these figures, such observations and conclusions are summarized as follows:

1) When the frame drop rate is 0, the environment operates normally. RLRD and TD3, owing to their real-time interaction with the environment, can better reflect the impact of the current policy. However, they cannot solve the relocate task. Offline algorithms, despite variations in performance due to training dataset differences, generally match or surpass the best performance of the training datasets. For instance, with the expert dataset, all tasks typically achieve expert-level performance; with the medium-expert dataset, tasks like ant, hopper, and walker still perform close to expert levels. Except when using the cloned dataset, where all algorithms perform near a random policy. In summary, performance at a frame drop rate of 0 reflects the state-of-the-art performance in a normal environment.

2) As the frame drop rate increases, the degree of information loss in the environment becomes more severe, making decision-making more difficult for the agent and leading to worse performance. Specifically, for RLRD, in tasks other than the relocate task, there is a consistent downward trend, with a few rare cases where it remains unchanged. This indicates that directly adopting delayed strategies when facing frame drops is not suitable. TD3 exhibits similar

trends to RLRD, especially in the ant and hopper tasks, where performance drops significantly even with a rate of only 0.2. This suggests that even a well-trained strategy in a normal environment is not suitable for direct application in an environment with frame drops. DT and CQL, across different tasks and datasets, perform significantly worse compared to Defog, D3D_iterative, and D3D_direct. The latter three algorithms show relatively stable performance with increasing drop rates. However, as the drop rate becomes larger (e.g., 0.6 or 0.8), D3D variants outperform Defog, indicating the effectiveness and robustness of the proposed algorithms.

3) Both variants of D3D consistently outperform other algorithms, especially in tasks like hopper-expert and pen-expert, where agents can achieve performance surpassing that of experts even with a frame drop rate of 0.8. However, in tasks like door-cloned and relocate-cloned, both D3D variants have minimal impact. This can be attributed to the complexity of the environment's dynamics, where other state-of-the-art RL algorithms cannot yield good results even in a normal environment. Additionally, in most cases, D3D_iterative outperforms D3D_direct, except in the door-expert and pen-expert tasks. This suggests that the former provides a more real-time reflection of state changes compared to the latter.

V. CONCLUSIONS

This paper introduces D3D, an offline algorithm designed to address frame dropping challenges in remote control systems. D3D adopts classifier-guidance with low-temperature sampling to generate return-maximizing trajectories without frame drops, while simultaneously training inverse dynamics using the same offline datasets. Furthermore, when states are dropped in the evaluation phase, D3D seamlessly replaces them with predictions from the diffusion model. Experimental evaluations across various tasks demonstrate that D3D achieves comparable or superior performance compared to existing algorithms.

Future research can explore two primary directions. Firstly, applying D3D to real-world tasks susceptible to frame drops, like multi-sensor-equipped robots, holds significant promise. Additionally, investigating how D3D can tackle delays, which are common in real-world tasks, is another essential area for research.

REFERENCES

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

[2] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.

[3] Elia Kaufmann, Leonard Bauersfeld, Antonio Loquercio, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Champion-level drone racing using deep reinforcement learning. *Nature*, 620(7976):982–987, 2023.

[4] S Balemi and UA Brunner. Supervision of discrete event systems with communication delays. In *1992 American control conference*, pages 2794–2798. IEEE, 1992.

[5] 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*, pages 219–224. IEEE, 1991.

[6] Kaizhe Hu, Ray Chen Zheng, Yang Gao, and Huazhe Xu. Decision transformer under random frame dropping. *arXiv preprint arXiv:2303.03391*, 2023.

[7] 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.

[8] 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*, pages 1346–1355, 2021.

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

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

[11] Baiming Chen, Mengdi Xu, Liang Li, and Ding Zhao. Delay-aware model-based reinforcement learning for continuous control. *Neurocomputing*, 450:119–128, 2021.

[12] Esther Derman, Gal Dalal, and Shie Mannor. Acting in delayed environments with non-stationary markov policies. *arXiv preprint arXiv:2101.11992*, 2021.

[13] Anurag Ajay, Yilun Du, Abhi Gupta, Joshua Tenenbaum, Tommi Jaakkola, and Pulkit Agrawal. Is conditional generative modeling all you need for decision-making? *arXiv preprint arXiv:2211.15657*, 2022.

[14] 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.

[15] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pages 2256–2265. PMLR, 2015.

[16] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in neural information processing systems*, 34:8780–8794, 2021.

[17] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in neural information processing systems*, 32, 2019.

[18] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

[19] Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *International Conference on Machine Learning*, pages 8162–8171. PMLR, 2021.

[20] Tim Pearce, Tabish Rashid, Anssi Kanervisto, Dave Bignell, Mingfei Sun, Raluca Georgescu, Sergio Valcarcel Macua, Shan Zheng Tan, Ida Momennejad, Katja Hofmann, et al. Imitating human behaviour with diffusion models. *arXiv preprint arXiv:2301.10677*, 2023.

[21] 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*, pages 5026–5033. IEEE, 2012.

[22] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017.

[23] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

[24] 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.

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