PrivORL: Differentially Private Synthetic Dataset for Offline Reinforcement Learning

Chen Gong*, Zheng Liu*, Kecen Li, and Tianhao Wang
University of Virginia

***Abstract*—Recently, offline reinforcement learning (RL) has become a popular RL paradigm. In offline RL, data providers share pre-collected datasets—either as individual transitions or sequences of transitions forming trajectories—to enable the training of RL models (also called agents) without direct interaction with the environments. Offline RL saves interactions with environments compared to traditional RL, and has been effective in critical areas, such as navigation tasks. Meanwhile, concerns about privacy leakage from offline RL datasets have emerged.**

To safeguard private information in offline RL datasets, we propose the first differential privacy (DP) offline dataset synthesis method, PrivORL, which leverages a diffusion model and diffusion transformer to synthesize *transitions and trajectories*, respectively, under DP. The synthetic dataset can then be securely released for downstream analysis and research. PrivORL adopts the popular approach of pre-training a synthesizer on public datasets, and then fine-tuning on sensitive datasets using DP Stochastic Gradient Descent (DP-SGD). Additionally, PrivORL introduces curiosity-driven pre-training, which uses feedback from the curiosity module to diversify the synthetic dataset and thus can generate diverse synthetic transitions and trajectories that closely resemble the sensitive dataset. Extensive experiments on five sensitive offline RL datasets show that our method achieves better utility and fidelity in both DP transition and trajectory synthesis compared to baselines. The replication package is available at the GitHub repository.[1]

I. Introduction

Recent studies highlight that privacy leakage risks are prevalent in RL systems, like using membership inference attacks (MIAs) to infer the environment information or training data [1], [2]. Pan et al. [1] present that attackers can use MIAs to steal map information from RL agents. Reinforcement Learning from Human Feedback method trains models with human evaluations [3]. Human feedback data, such as ratings or preference labels, can hold sensitive user information [4]. Similarly, offline RL faces comparable privacy leakage challenges. Du et al. [5] propose ORL-Auditor, which infers the training trajectories of agents, an approach that can also be considered a form of MIA.

*Equal Contributions. Zheng and Kecen work as independent researchers and remote interns at UVA.

[1]https://github.com/2019ChenGong/PrivORL

Network and Distributed System Security (NDSS) Symposium 2026
23-27 February 2026, San Diego, CA, USA
ISBN ISBN 979-8-9919276-8-0
https://dx.doi.org/10.14722/ndss.2026.240149
www.ndss-symposium.org

Privacy-preserving synthetic data generates artificial datasets that retain key characteristics of real data, enabling secure sharing while reducing privacy risks [6]. Differential Privacy (DP) dataset synthesis [7], [8] offers a theoretical guarantee for quantifying privacy leakage from real data through the use of synthetic datasets. It protects an individual's data privacy within a dataset throughout the data training.

The offline RL dataset comprises either *transitions* or *trajectories*. Transitions refer to individual steps in the RL process, capturing the movement from one state to another based on an action taken by an agent, along with the associated reward. In contrast, trajectories encompass complete sequences of such transitions, providing a holistic view of an agent's behavior over time, including all states, actions, and rewards. Both representations are fundamental to offline RL datasets, as transitions offer granular insights into decision-making, while trajectories enable analysis of long-term patterns and dependencies. In practice, as introduced in Section II-B, it is common for users to contribute transitions or trajectories to the offline RL datasets. Therefore, it is necessary to consider both transition-level and trajectory-level privacy protection.

Existing Methods. Previous works have proposed synthesizing offline RL dataset [9], [10], [11]. However, these methods do not provide formal privacy guarantees (i.e., no DP) for the synthetic datasets. The original data still faces the risk of privacy leakage [12], [13]. Meanwhile, DP data synthesis methods have been increasingly developed for other modalities, e.g., images [14], [15], tabular data [16], [17], text data [7], network records [8], and so on, but these are not tailored to offline RL datasets, which pose unique challenges due to their sequential and dynamic nature, which are introduced as follows.

Our Proposal. We focus on presenting the feasibility of applying DP to offline RL dataset synthesis and on leveraging existing DP primitives to address challenges unique to this setting. To this end, we propose PrivORL (Differentially **Priv**ate Dataset Synthesizer for **O**ffline **R**einforcement **L**earning), the first DP synthesizer for offline RL datasets. PrivORL comprises PrivORL-n and PrivORL-j, which perform *transition*-level and *trajectory*-level DP dataset synthesis, respectively.

We start with adapting existing DP diffusion models, as diffusion models achieve remarkable synthetic performance in complex synthesis tasks [18], [19], [20], [21]. However, directly applying diffusion models to DP offline RL dataset

synthesis has a couple of challenges: (1) DP noise must be introduced during synthesizer training, which introduces instability into the training process [22], [14]. (2) The success of agents trained in offline RL is highly dependent on the diversity of the dataset (we elaborate more in Section IV-C). (3) Trajectory-level DP synthesis presents unique challenges, such as high dimensionality and temporal dependencies in the dataset. We provide more discussions in Appendix H-A of [23]. Besides, how to effectively create a diverse DP synthetic dataset remains an open question [9].

We then introduce how PrivORL-j addresses the third unique challenge. Both PrivORL-n and PrivORL-j leverage the same paradigm for tackling the first and second common challenges, which we discuss below.

- First, we adopt the popular paradigm [14], [24] of pre-training with public datasets and only fine-tuning on sensitive data under DP. It is important to use a public dataset during the pre-training phase to achieve fast convergence and generate reasonable synthetic datasets.
- To solve the second challenge, inspired by the random network distillation (RND) concept in RL and bug detection [25], [26], we propose using the curiosity module. The curiosity module quantifies the 'novelty' of synthetic data. However, unlike works [25], [26], diffusion model training lacks a reward mechanism that integrates novelty feedback (as described in Section II-C). We propose replacing a portion of real data with high-novelty synthetic data, encouraging synthesizers to capture underlying characteristics of high-novelty data for more diverse synthesis.
  In addition, we propose using the curiosity module during the pre-training phase instead of the fine-tuning phase. The high-level motivation is that pre-training offers greater flexibility and tolerance for instability.
- To address the challenges of high dimensionality, PrivORL-j manages long trajectories by splitting trajectories into fragments and using a conditional synthesizer to capture the relationship between fragments, enabling the synthesis of fragments that can be seamlessly stitched into trajectories. To capture long-range temporal dependencies in trajectory-level DP, PrivORL-j extends PrivORL-n by integrating a transformer [27] into its diffusion model used in PrivORL-n, modeling complex temporal relationships in trajectories.

We elaborate on differences in synthesizing transition-level versus trajectory-level datasets in Section IV-B.

Evaluations. We conduct experiments on five types of sensitive datasets, three `Maze2D`, one `Kitchen`, and one `Mujoco` datasets, to present the effectiveness of PrivORL-n and PrivORL-j mainly from the following two perspectives.

Utility: For DP transition synthesis under privacy budgets $\epsilon = \{1, 10\}$, in `Maze2D` domain, agents trained on the synthetic dataset generated by PrivORL-n, using three prominent offline RL algorithms, achieve an average normalized return of 51.9 and 69.3 across studied sensitive datasets. This performance exceeds the baseline returns of 11.7, 3.4, 2.0, and 3.2 at $\epsilon = 1$, and 18.9, 3.9, 5.1, and 7.2 at $\epsilon = 10$, achieved by PGM [17], PrivSyn [16], PATE-GAN [28], and PrePATE-GAN. In ablation studies, without the curiosity module and pre-training, PrivORL-n reduces to the pre-training DP diffusion [24] and DPDM [29] proposed in DP image synthesis. Removing pre-training and the curiosity module from PrivORL-n reduced the performance, with average normalized returns of trained agents dropping by 25.9% and 16.3% in `Maze2D-umaze` dataset under $\epsilon = 10$. For DP trajectory synthesis at $\epsilon = \{1, 10\}$, PrivORL-j achieves average 15.0 and 10.4 higher returns than DP-Transformer [30], in the `Maze2D` domain.

Fidelity: In transition synthesis, PrivORL-n outperforms the baseline models in both marginal and correlation statistics [31]. Regarding trajectory synthesis, under $\epsilon = 10$, PrivORL-j achieves an average of 15.9% improvements of TrajScores compared to DP-Transformer [30] across studied datasets. The fidelity metrics are introduced in Section VIII-C.

Besides, Section IX-D shows that synthetic transitions generated by PrivORL-n exhibit strong resistance to an advanced white-box membership inference attack (MIA) method [12], outperforming synthesis without DP protection.

Contributions. In summary, our contributions are:

- We introduce PrivORL, a DP offline RL dataset synthesizer for both transition and trajectory. It facilitates the sharing of datasets and promotes privacy protection.
- We introduce the curiosity module to synthesizer training, which enables PrivORL to generate synthetic data that is both more diverse and of higher utility.
- We conduct a comprehensive evaluation of PrivORL. The results show that PrivORL outperforms the baselines by synthesizing transitions or trajectories of higher utility and fidelity across datasets from five tasks in three domains.

II. Backgrounds

A. Reinforcement Learning

Reinforcement Learning (RL) aims to train a policy, denoted as π (also referred to as an agent [32]), to solve sequential decision-making tasks. The sequential decision-making tasks can be formulated as a five-tuple Markov Decision Processes (MDP) [33], $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma \rangle$, where $\mathcal{S}$ and $\mathcal{A}$ represent the state space and action space, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ indicates the reward received, and the transition function $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$. $\gamma \in (0, 1)$ is the discount factor when computing accumulated rewards. At each timestep t, the agent π takes an action a_t at the state s_t. Then, the agent obtains a reward $r_t \sim \mathcal{R}(s_t, a_t)$ from the reward function, and the MDP transitions to the next state s_{t+1}. RL algorithms aim to train the agent π^* to maximize the expected cumulative reward for the specific task, $\pi^* = \arg\max_\pi \mathbb{E}_{a_t \sim \pi} \left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, a_t)\right]$ [10]. The agent learns through a trial-and-error paradigm by interacting with the environment.

Offline Reinforcement Learning. In certain scenarios, such as healthcare [34], [35], online RL is unsuitable. It is impractical to conduct trial-and-error experiments on patients. During training, offline RL relies on learning from a static dataset D, which is collected from various users and can be composed

in two ways: one where a single user contributes an *entire trajectory* or one where a single user contributes a *single transition* [36], [37]. Specifically, a trajectory-based dataset can be represented as,

$$D = \left\{ \left(s_0^i, a_0^i, r_0^i, s_1^i, \cdots, s_{|\tau|}^i, a_{|\tau|}^i, r_{|\tau|}^i \right) \middle| \, i = 1, \ldots, N \right\}.$$

Alternatively, a transition dataset consists of a series of four-tuples, which is defined as follows,

$$D = \{(s_t, a_t, r_t, s_{t+1})\}_{t=1}^{N}.$$

where N is the offline dataset size. The dataset D in offline RL is collected by various strategies [38]. The rationality of these two datasets can be illustrated with examples. A single user contributing a transition, such as a doctor observing a diabetic patient's blood sugar level, adjusting the insulin dosage, and recording the resulting change and patient response, is practical for analyzing the immediate impact of isolated decisions, offering a focused way to optimize specific actions [37]. Conversely, a user contributing an entire trajectory, like a driver documenting a full trip from home to work—including every turn, acceleration, and stop along with feedback like fuel efficiency or safety—is reasonable for long-term outcomes, capturing how actions interplay over time [39]. These two offline RL datasets reflect realistic data collection scenarios and serve distinct purposes [38]. Thus, the transition and trajectory DP protection are both necessary. We discuss the uniqueness of offline RL trajectories in Appendix H-A of [23].

B. Differential Privacy

Differential privacy (DP) [40] protects an individual's data privacy within a dataset throughout the data processing phase. We present the concept of DP as follows.

Definition 1 (Differential Privacy [40]): A randomized mechanism $\mathcal{Q}$ satisfies (ε, δ)-differential privacy (DP) ($\varepsilon > 0$ and $\delta > 0$), if and only if, for any two neighboring datasets D, D' and any $\mathcal{O}$, the following is satisfied,

$$\Pr[\mathcal{Q}(D) \in \mathcal{O}] \leq e^{\varepsilon} \Pr[\mathcal{Q}(D') \in \mathcal{O}] + \delta. \tag{1}$$

where $\mathcal{O}$ denotes the set of all possible outputs from $\mathcal{Q}$.
The privacy budget parameters, ε and δ, are both non-negative and measure the privacy loss in the data. A lower ε value indicates better privacy protections, while a smaller δ reduces the likelihood that the privacy guarantees provided by ε will be compromised. Two datasets D, D' differing by a single *transition* or *trajectory* are considered neighbors. This can be interpreted as transition-level or trajectory-level DP.

Differentially Private Stochastic Gradient Descent. In machine learning, the most popular way to train the model to satisfy DP is DP-SGD [22]. This method modifies the standard SGD, which computes gradients based on Poisson sampling of mini-batches, clips the original gradients of the model's parameters, and adds random Gaussian noise to the clipped gradient throughout training. We first denote the 'Clip' operation as, $\text{Clip}_C(\mathbf{g}) = \min\left\{1, \frac{C}{||\mathbf{g}||_2}\right\}\mathbf{g}$, where $\mathbf{g}$ is the original gradient and C is a hyper-parameter. The 'Clip' operation scales the norm of the gradient down to less than C, and the model parameters θ are updated via,

$$\eta \mathbb{E}_{x_i \in x}\left[\text{Clip}_C\left(\nabla \mathcal{L}(\theta, x_i)\right) + C\mathcal{N}(0, \sigma^2 \mathbb{I})\right], \tag{2}$$

where η is the learning rate; L is the loss function; x indicates a mini-batch Poisson sampled with a sample rate q (and we denote the size of it by $|x|$); $\nabla \mathcal{L}(\theta, x_i)$ represents the gradient for x_i. $\mathcal{N}(0, \sigma^2 \mathbb{I})$ is the Gaussian noise with the variance σ, and B is the batch size. When there are many iterations of DP-SGD in our method, composition theorems (privacy accounting) can be used to derive the final values of (ϵ, δ). We defer those derivations to Appendix B.

C. Diffusion Model

Diffusion models [21] are a class of likelihood-based generative models that present excellent generative capabilities in various fields [10], [27], [41]. Prior works [6], [14] show that diffusion models provide more stable training dynamics and consistently outperform GAN- and VAE-based approaches in terms of synthesis quality. Most state-of-the-art DP dataset synthesis methods adopt diffusion models as their primary synthesizers [6], [41]. Following this trend, we use diffusion models as the DP synthesizers. Table I shows that diffusion-based methods achieve superior downstream performance compared to GAN-based methods.

Traditional diffusion models. Diffusion models [21] consist of two processes: (1) The *forward* process that progressively corrupts clean data x_0 by adding Gaussian noise, which outputs a noisier data sequence: $\{x^1, \ldots, x^T\}$, and T is the number of noising steps. As the number of steps increases, the data becomes noisier, gradually resembling Gaussian noise more closely and deviating further from the original data sample with each step. (2) The *reverse* process progressively denoises a noise to clean data using a trainable neural network. The forward process between adjacent noisy data, i.e., x^{t-1} and x^t, follows a Gaussian distribution. Then, the forward process of diffusion models is defined as, $p\left(x^t \left| x^{t-1}\right.\right) = \mathcal{N}\left(x^t; \sqrt{1-\beta_t}x^{t-1}, \beta_t \mathbb{I}\right)$, where β_t regulates the magnitude of the added noise at each step and is usually pre-defined by users. We note $\bar{\alpha}_t := \prod_{s=1}^{t}(1-\beta_s)$. The likelihood between the clean data x_0 and noisy data in step t is, $p\left(x^t \left| x^0\right.\right) = \mathcal{N}\left(x^t; \sqrt{\bar{\alpha}_t}x^0, (1-\bar{\alpha}_t)\,\mathbb{I}\right)$. Therefore, we sample x^t directly from x^0 in closed form instead of adding noise t times as, $x^t = \sqrt{\bar{\alpha}_t}x^0 + e\sqrt{1-\bar{\alpha}_t},\ e \sim \mathcal{N}(0, \mathbb{I})$. The final objective of diffusion models is defined as,

$$\mathcal{L}(\theta, x) = \mathbb{E}_{t \sim \mathcal{U}(1,T), x^t \sim p(x^t | x^0 = x)} \left\| e - e_\theta\left(x^t, t\right)\right\|^2, \tag{3}$$

where $e \sim \mathcal{N}(0, \mathbb{I})$ and e_θ is a neural network parameterized with θ that is updated to minimize Equation (3). $\mathcal{U}(1, T)$ is the uniform distribution ranging from 1 to T. Thus, e_θ learns to predict the noise e of the noisy data at any step t. After being trained well, we apply e_θ to gradually denoise a random Gaussian noise to clean the data.

We note that the noise prediction network serves as the core component of diffusion models. Following prior work [14], we

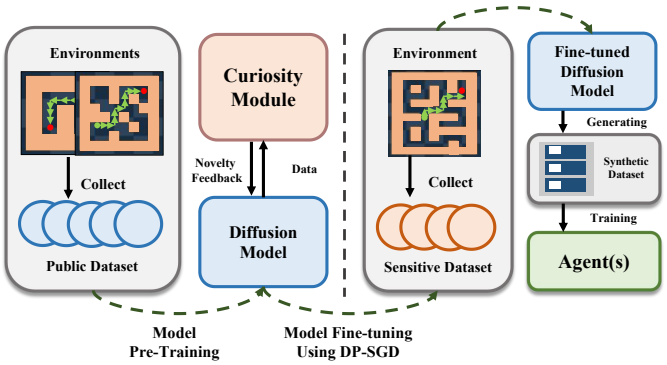


Fig. 1: High-level illustration of the overall workflow of PrivORL. Initially, PrivORL pre-trains synthesizers on public datasets, guided by a curiosity module. It then fine-tunes the model on sensitive datasets using DP-SGD. Finally, the fine-tuned model generates synthetic datasets for agent training.

use e_θ as a shorthand to represent the entire diffusion model for simplicity. This notation includes the noise prediction network and, when applicable, any additional modules.

Conditional diffusion models. Conditional diffusion models extend diffusion models by incorporating conditional inputs to guide the generation process. They are widely used for tasks such as sequence synthesis. Section VI-B presents that, for DP trajectory synthesis, conditional inputs enable us to capture temporal dependencies among different transitions in sequences. The objective function is reformulated as $\mathcal{L}(\theta, x, c)$, where c represents conditional inputs, and the noise prediction network at step t is defined as $e_\theta(x^t, t, c)$.

Diffusion Transformer. Diffusion transformers leverage transformer-based architecture [42] for noise prediction. They excel at modeling sequential data with temporal dependencies, such as text [7] or structured sequences [27], making them suitable for trajectory-level DP synthesis, where temporal dependencies in sequences must be preserved. Since transformers lack inherent sequential awareness, positional encoding is essential to capture the order of inputs [42].

Following prior works [9], [10], we adopt the Elucidated Diffusion Model (EDM) [43] for transition synthesis and the Diffusion Transformer [27] for trajectory synthesis. These models use an MLP and a Transformer, respectively, as the noise prediction network. We present more details of architectural designs in Appendix F-A.

III. Problem Setup and Preliminaries

A. Threat Model

We assume that the data provider holds a highly sensitive dataset, such as medical records, which can be used for offline RL agent training. Directly sharing such datasets poses significant privacy risks. To mitigate these risks, several approaches advocate generating synthetic datasets as substitutes, while adversaries can still infer the sensitive dataset information [12] through the synthetic datasets.

DP dataset synthesis addresses this challenge by providing formal, mathematically rigorous guarantees that limit the influence of any individual record on the generated data. This offers general protection and guarantees that, even with auxiliary knowledge, an attacker's ability to infer specific private details about individual transitions or trajectories remains strictly bounded. DP-based synthesis has gained traction across multiple domains, including image [14], text data [7], and tabular data [16], making it a promising paradigm for privacy-preserving offline RL datasets.

B. Problem Statement

We aim to generate new transition-based and trajectory-based datasets that statistically mirror the original dataset under DP. Specifically, we possess a set of sensitive offline RL data denoted by D_s, and generate a set of synthetic data individuals, $\hat{D}$. Agents trained on the sensitive dataset exhibit a similar level of performance compared to those trained on the synthetic dataset. Besides, the synthetic dataset has statistical characteristics similar to those of the original dataset.

C. Adapting Existing Methods

Marginal-based Solutions. We first consider the marginal-based solutions [16], notably PrivSyn [16]. In transition synthesis, we can treat the data as a table, where each row is a transition, and each column is one element of the transition (state, action, reward, and state can take multiple columns as they can be multi-dimensional). The challenge with marginal-based methods is their difficulty in managing data with large dimensions. These methods are notably slow when processing large-scale sensitive datasets. Section IX-A empirically shows that directly adapting those methods does not work well. In trajectory synthesis, trajectory lengths vary rather than being fixed, rendering traditional DP tabular synthesis unsuitable.

ML-based Solutions. Another approach is to adapt more complex machine-learning models, such as Generative Adversarial Networks (GANs) and diffusion models. For GAN-based methods, we adopt PATE-GAN [28]. As presented in previous works [44], GAN suffers from problems of unstable training and sometimes fails to fit the distribution of sensitive datasets [45]. Diffusion-based methods encounter similar challenges when processing complex datasets, such as images [29].

Leveraging Pre-training. Diffusion models achieve excellent synthesis performance in various fields [24], [10], [14], [21], [46], but one challenge particular to training diffusion models with DP is that diffusion models are typically larger, and thus need more training. In the era of 'large models,' it is increasingly common to begin with a pre-trained model and then proceed to fine-tune it for better performance [24], [20]. Thus, we adopt this setting and adapt the pre-training method in Ghalebikesab et al. [24]. However, it is less clear how to best use pre-training to enhance marginal-based solutions (because the lower-dimensional marginals are obtained in one-shot with DP, and there is no convergence issue).

IV. Introducing PrivORL

We introduce PrivORL with two variants PrivORL-n and PrivORL-j for handling transition-level and trajectory-level DP

definitions. PrivORL has a unified synthesizer training the synthesis paradigm. As presented in Figure 1, PrivORL first pre-trains synthesizers on datasets without privacy concerns (i.e., public datasets), guided by feedback from the curiosity module. PrivORL then fine-tunes the pre-trained model on sensitive datasets using DP-SGD. Finally, the fine-tuned synthesizer generates synthetic datasets for agent training.

A. Differences Between PrivORL-n and PrivORL-j

However, there are some differences between PrivORL-n and PrivORL-j, which are introduced as follows.

- *Data structure.* PrivORL-n models each transition independently, whereas PrivORL-j models the entire trajectory, capturing temporal dependencies (e.g., state transition function). Therefore, PrivORL-n models each transition independently using diffusion models, whereas PrivORL-j should model to capture temporal dependencies with sequence generation models like diffusion transformers.
- *Data Dimensionality.* Trajectories have significantly larger dimensions than transitions. In PrivORL-n, each transition is generated independently, processing small-scale data. Conversely, PrivORL-j handles long sequences, risking a memory explosion. As detailed in Section VI-B, PrivORL-j handles long trajectories by dividing them into fragments and using a conditional synthesizer to model their interconnections, allowing for the generation of fragments that seamlessly combine into cohesive trajectories.
- *Data generation.* PrivORL-n generates each transition independently, whereas PrivORL-j synthesizes trajectory fragments and stitches them into a complete trajectory, requiring consideration of inter-fragment dependencies.

B. Design Overall

We use different architectures, training, and synthesis to match the characteristics of transition and trajectory.

- Following prior work [9], [10], we use the Elucidated Diffusion Model (EDM) [43] for PrivORL-n (unconditional generation) and the Diffusion Transformer [27] for PrivORL-j, conditioned on the link transition to capture fragment relationships, as introduced in Section VI-A. EDM and Diffusion Transformer use an MLP and a Transformer for noise prediction. Architectural details appear in Appendix F-A.
- *Training.* Section II-C details the training process of diffusion models. The input of the noise prediction network e_θ is noisy data, and the output is the prediction of added noise. Equation (3) minimizes the output of e_θ and real added noise e. Thus, e_θ learns to predict the noise e added to noisy data at any step t. For PrivORL-n, e_θ takes as input noisy transitions and outputs the corresponding noise. For PrivORL-j, e_θ receives noisy trajectory fragments with the link transition, and outputs the noise. Section VI-B explains how link transitions are incorporated into the training.
- *Synthesis.* As introduced in Section II-C, once trained, synthesizers use e_θ to generate data through denosing the noise sampled from a Gaussian distribution. PrivORL-n uses e_θ to denoise Gaussian noises and generate transitions. As shown in Section VI-C, PrivORL-j generates fragments sequentially and stitches fragments to full trajectories.

We elaborate on the technical details of PrivORL-n and PrivORL-j in Section V and Section VI, respectively.

C. Motivation of Leveraging Curiosity-Driven Pre-training

Offline RL requires the algorithm to understand the dynamics of the environment's MDP from datasets [38]. Then, agents are trained to achieve the maximum possible cumulative reward when interacting with the environment. To better understand the environment's MDP, a diverse training dataset is necessary, which helps ensure that the agent encounters a comprehensive spectrum of states, actions, and rewards, representing a wide range of environmental scenarios [38], [47]. Thus, the success of offline RL agents relies heavily on the breadth and diversity of the datasets.

Inspired by the random network distillation in RL and bug detection [25], [26], we propose using the curiosity module, which quantifies the 'novelty' of synthetic data individuals (as detailed in Section V-A). To incorporate the novelty feedback into synthesizer pre-training, we replace a portion of real data with high-novelty synthetic data, encouraging synthesizers to capture underlying characteristics of high-novelty data. The high-level motivation of curiosity-driven during pre-training instead of fine-tuning phase is that we have more flexibility and can tolerate more instability in pre-training phase, as supported by Table V. Technical details appear in Section V-A.

V. Transition-Level DP: PrivORL-n

This section focuses on transition-level DP synthesis and introduces the curiosity module.

A. Curiosity Scores

We propose a "curiosity module" to assess the diversity (or novelty) of the synthetic dataset and promote diverse data synthesis. Our approach is based on random network distillation (RND) [48]. This section details how to measure the novelty (whether a transition is frequently or rarely generated by the synthesizer) of synthetic data. RND uses prediction errors to represent the difference between the outputs of a fixed target network and a trainable predictor network. Therefore, when synthetic data is rarely generated, and the predictor predicts that it is unfamiliar, the predictor network's output deviates from the target network's output, resulting in a higher error.

RND uses two randomly initialized networks: a fixed *target* network f and a *predictor* network $\hat{f}$ that aims to learn the output of f. During pre-training, synthetic data x is input to both f and $\hat{f}$. The target network outputs a random vector $f(x) \rightarrow \mathbb{R}^d$ (d is the vector dimension), which $\hat{f}$ tries to match. Specifically, let the f and $\hat{f}$ be parameterized by ϕ and $\hat{\phi}$. The objective is to minimize the following objective,

$$c(x) = \left\| \hat{f}_{\hat{\phi}}(x) - f_{\phi}(x) \right\|_{\ell_2}^2 . \tag{4}$$

This error is also the *curiosity score* $c(x)$ to quantify the 'novelty' of a synthetic data x and should be higher for 'novel' synthetic data than for those previously generated repeatedly.

This section details how to incorporate the novelty feedback into the diffusion model pre-training.

B. Curiosity-driven Updating

This section explains how curiosity scores guide synthesizer updates. As described in Section II-C, diffusion model training lacks a reward mechanism that integrates novelty feedback. In fact, diffusion models fit data distributions by capturing statistical characteristics of training datasets [21]. To address this challenge, we include high-curiosity synthetic data in the training set, enabling the model to learn its traits and generate diverse data. In particular, in one iteration, we sample a batch of data X from the training dataset with a batch size of B. We then generate an equivalent number of synthetic data using the synthesizers, denoted as $\hat{X}$. The curiosity module measures the curiosity scores for synthetic data. We rank these scores and select the top-p synthetic data from $\hat{X}$ to replace the same number of data in the training batch dataset X. The p is the *curiosity rate* ranging from 0 to 1, which controls how strongly curiosity is applied. Then, the modified dataset X_r is used to train synthesizers. Appendix C presents why our method can benefit the diversity of synthetic datasets.

Transitions Synthesis. For each transition, we first sample a Gaussian noise x^T (T is the number of noising steps), and we use the diffusion model to denoise x^T to the less noisy x^{T-1}. For any t in range of 1 to T, this denoising entails the computation of the estimated noise mean μ for x^{t-1} [21],

$$\mu = \frac{1}{\sqrt{\alpha_t}} \left(x^t - \frac{1-\alpha_t}{\sqrt{1-\beta_t}} e_\theta \left(x^t, t \right) \right), \tag{5}$$

where α_t and β_t are hyper-parameters as introduced in Section F-A, and t is the current step. The x^{t-1} can be obtained as [21], $x^{t-1} = \mu + \sigma_t e,\ e \sim \mathcal{N}(0, \mathbb{I})$. The σ_t regulates the magnitude of the added noise and is pre-defined by users. Repeating this denoising process until $t = 0$, the x^0 indicates our final synthetic transition.

C. Standard DP-SGD Fine-Tuning.

We leverage DP-SGD (as described in Section II-B) to fine-tune the pre-trained diffusion model on the sensitive transitions to satisfy DP. Algorithm 1 presents the training of PrivORL-n.

Algorithm 1: Workflow of PrivORL-n

Input: The public and sensitive transition set: D and D_s; Synthesizer e_θ parameterized with θ; Target and predict networks f_ϕ, $f_{\hat{\phi}}$ parameterized with ϕ and $\hat{\phi}$; The curiosity rate: p.
// Curiosity-Driven Pre-training
while *training epochs* $<$ *target epochs* **do**
  $X \leftarrow$ Randomly select B data from the D;
  $\hat{X} \leftarrow$ Generate B trajectory fragments using e_θ;
  for $x \in \hat{X}$ **do**
    $c(x) = \left\| f_{\hat{\phi}}(x) - f_\phi(x) \right\|_{\ell_2}^2$
    Update $\hat{\phi}$ by minimizing $c(x)$;
  end
  Sort set $\hat{X}$ according to c and get the top-p sorted set X_o;
  $X_r \leftarrow$ Replace a subset of X with from X_o ;
  Train e_θ on X_r using Eq. (3);
end
// Private Fine-tuning
Fine-tune e_θ on D_s with DP-SGD (using Eq. (3)) ;
Output: The well-trained diffusion model e_θ.

VI. Trajectory-Level DP: PrivORL-j

Moving from transitions to trajectories, we face two questions: (1) How to support generate high-dimensional long trajectories? (2) How can we capture long-range temporal dependencies in trajectory-level DP synthesis?

To tackle high-dimensionality challenges, PrivORL-j divides long trajectories into fragments and employs a conditional diffusion transformer to model their interconnections, enabling the generation of fragments that seamlessly combine into cohesive trajectories. To capture long-range temporal dependencies in trajectory-level DP synthesis, PrivORL-j enhances PrivORL-n by incorporating a transformer [27] into its diffusion model, effectively modeling intricate temporal relationships in trajectories. We introduce PrivORL-j as follows.

A. Dataset Proprocess

As previously mentioned, our use of transformers involves working with fragments. To enable this, we preprocess the data into fragments, consisting of consecutive transitions. We first segment the trajectory into equal-length segments, referring to processes in text sequences. Any sub-trajectory that does not reach the required length is padded. We should partition the trajectories from both the public dataset $D = \{\tau_n\}_n$ and sensitive dataset $D_s = \{\tau_i\}_i$ into trajectory fragments, resulting in $D' = \{(\tau_n^s, S_n)\}_n$ and $D'_s = \{(\tau_i^s, S_i)\}_i$, respectively. Here, each trajectory τ is divided into N fragments, $\tau \rightarrow \{\tau^{s_i}\}$. The linking transition S_i connects consecutive fragments: for a given fragment, S_i is defined as the preceding transition of the first transition of the fragment or zero-padded for the initial fragment. For instance, we consider a complete trajectory $\tau = (\cdots, s_p, a_p, r_p, s_{p+1}, a_{p+1}, r_{p+1}, \cdots)$ that might be randomly segmented into two fragments, such as $\tau_1^s = (\cdots, s_p, a_p, r_p)$ and $\tau_2^s = (s_{p+1}, a_{p+1}, r_{p+1}, \cdots)$, where the linking transition S_2 for τ_2^s is the last transition of τ_1^s, i.e., $S_2 = (s_p, a_p, r_p, s_{p+1})$.

We denote the number of transitions in a trajectory fragment by H (sometimes we call it *horizon*). Additionally, we introduce a terminal signal d_t for each transition: $d_p = 1$ indicates that s_p is a terminal state, in which case $s_{p+1} = 0$; otherwise, $d_p = 0$. Consequently, a trajectory fragment is formalized as $\tau^s = \left[(s_p, a_p, r_p, s_{p+1}, d_p)_{p=1}^H\right]$.

B. Synthesizer Training

As described in Section II-C, PrivORL-j generates trajectory fragments by progressively denoising Gaussian noise through T timesteps, guided by a conditional input. During the forward

Algorithm 2: Workflow of PrivORL-j

Input: The public and sensitive trajectory set: D and D_s; The horizon size: H; Diffusion transformer e_θ parameterized with θ; Target and predict networks f_ϕ, $f_{\hat{\phi}}$ parameterized with ϕ and $\hat{\phi}$; The curiosity rate: p.
Initialization: $D', D'_s = \varnothing$.
// Dataset Preprocess
for $\tau \in D, D_s$ **do**
  Split τ to N fragment $\left\{(\tau_i^s, S_i)_{i=1}^N\right\}$, and each τ^s has H transitions;
  if $\tau \in D$ **then** $D' \cup \left\{(\tau_i^s, S_i)_{i=1}^N\right\}$;
  else $D'_s \cup \left\{(\tau_i^s, S_i)_{i=1}^N\right\}$;
end
// Curiosity-Driven Pre-training
while *training epochs* $<$ *target epochs* **do**
  $X \leftarrow$ Randomly select B data from the D';
  $\hat{X} \leftarrow$ Generate B trajectory fragments using e_θ;
  for $\tau^s \in \hat{X}$ **do**
    $c(\tau^s) = \left\| \hat{f}_{\hat{\phi}}(\tau^s) - f_\phi(\tau^s) \right\|_{\ell_2}^2$
    Update $\hat{\phi}$ by minimizing $c(\tau^s)$;
  end
  Sort set $\hat{X}$ according to c and get the top-p sorted set X_o;
  $X_r \leftarrow$ Replace a subset of X with from X_o ;
  Train e_θ on X_r using Eq. (6);
end
// Private Fine-tuning
Fine-tune e_θ on D'_s using Eq. (7) with DP-SGD to calculate the aggregated gradient for each trajectory;
Output: The well-trained diffusion transformer e_θ.

process of the diffusion model, we incrementally add noise to the trajectory fragments, generating a sequence of noisy fragments, $\{\tau_t^s\}_{t=0}^T$. This entails training a noise prediction network to estimate the noise added at each timestep t, using the current noisy trajectory fragment τ_t^s and conditional input s. As introduced in Section II-C, PrivORL-j leverages the transformer as the prediction network, which is formulated as $e_\theta(\tau_t^s, t, S)$, and θ means the network parameters. We elaborate on the training processes of PrivORL-j as follows,

- *Curiosity-Driven Pre-training.* This section describes the curiosity module pre-training in PrivORL-j.
- *Input Embeddings.* We detail the process of treating the inputs of the noise prediction network as an embedding sequence, which prepares them for transformer training.
- *Sequence Processing.* It uses the input embeddings to train the transformer, enabling accurate prediction of the added noise in diffusion processing.
- *Private Fine-tuning.* We describe how PrivORL-j fine-tunes the synthesizer on sensitive fragments using DP-SGD.

Curiosity-Driven Pre-training. This process is almost the same as what we introduced in Section V. In PrivORL-j, we measure the novelty of trajectory fragments instead of single transitions in PrivORL-n. Then, the modified training dataset is leveraged to pre-train the synthesizer. As shown in Section F-A, the objective of conditional diffusion is,

$$\mathcal{L}_{\text{diff}} = \mathbb{E}_{(\tau^{s_t}, S), t}\left[\left\| e_\theta\left(\tau^{s_t}, t, S\right) - e^t \right\|_1\right] \tag{6}$$

where $\|\cdot\|_1$ denotes the L_1 norm, and τ^{s_t} and e^t are the noisy trajectory fragment and true noise at the t-th timestep.

Input Embeddings: Inputs of the prediction network include three parts: (1) the noisy trajectory fragment τ_t^s; (2) timestep in the forward process t; and (3) conditional input S. Referring to previous works in the text synthesis [30], the sensitive trajectory fragments dataset D'_s should be embedded into a high-dimensional space using multilayer perceptrons (MLPs).

First, we embed the timestep t in the diffusion process [27], and recorded as $\text{TimeEmb}(t) \in \mathbb{R}^k$, where k is the dimension size of embedding. Then, we embed the conditional input S (a link transition includes a state, an action, a reward, a next state, and a terminal signal) with a separate MLP, $\text{ConditionEmb}(S) \in \mathbb{R}^{5\times k}$. For embedding the trajectory fragment τ_t^s, we divide τ_i^s into five components,

$$\left[(s_p)_{p=1}^H\right], \left[(a_p)_{p=1}^H\right], \left[(r_p)_{p=1}^H\right], \left[(s_{p+1})_{p=1}^H\right], \left[(d_p)_{p=1}^H\right],$$

and embed them with five separate MLPs. Thus, each trajectory fragment yields $5 \times H$ embeddings; concatenating one time embedding and five condition-transition embeddings results in $5\times H+5+1$ embeddings in total. As a result, PrivORL-j gets the input embedding $\mathbf{z}$, $\mathbf{z} = \text{InputEmbed}\left(\tau_t^s, t, S\right) \in \mathbb{R}^{(5\times H+6)\cdot k}$. Each embedding is treated as a token for subsequent transformer training, resulting in an input embedding $\mathbf{z}$ of $(5 \times H + 6)$ tokens, which refers to the basic units of input data for the transformer [42]. Then, we formulate the input embedding as the form of tokens, $\mathbf{z} = \{z_i\}_{i=1}^{5\times H+6}, z_i \in \mathbb{R}^k$.

Sequence Processing. Offline RL trajectory synthesis requires dynamic coherence, and state transitions must align with environmental dynamics [47], [32]. The transformer models these temporal relationships, ensuring that the generated trajectory is globally coherent, such as how earlier states in the trajectory influence subsequent state-action pairs. Then, we describe how transformers model the input embeddings, i.e., token sequence $\mathbf{z} = \{z_i\}_{i=1}^{5\times H+6}, z_i \in \mathbb{R}^k$, to generate the predicted noise.

To encode the relative positions of tokens in the sequence $\mathbf{z}$, we introduce *position embeddings*. Since transformers lack inherent sequential awareness, positional encoding is essential to distinguish the order of identical tokens. Drawing on implementation of prior work in text data processing [42], [30], we assume that i is the position index of the token ($i = \{0, 1, \ldots, 5 \times H + 5\}$, and $\text{PosEmb}(i)$ denotes the position embedding vector for the token at position i, and $\text{PosEmb}(i) \in \mathbb{R}^k$. Then, we add the positional embedding $\text{PosEmb}(i)$ to the token embedding and get the final embedded inputs of the transformer. The outputs of the transformer are the embedding matrix, $\text{Transformer}(\mathbf{z}_{\text{final}}) \in \mathbb{R}^{H\times k}$, and we decode them into noise prediction components using MLPs. The output of $e_\theta(\tau_t^s, t, S)$ is the predicted noise, matching the

dimensionality of the input trajectory fragment τ_t^s.

Private Fine-Tuning. We use DP-SGD to fine-tune the transformer on sensitive trajectories to satisfy DP at the complete trajectory level. As presented in Section VI-A, we split each trajectory into fragments of size H transitions between consecutive fragments. Distinguished from the DP-SGD applied in transition-level DP, DP-SGD is applied by aggregating the gradients of all fragments (τ^s, S) belonging to the same trajectory τ, clipping them to a maximum norm C, and adding noise at the trajectory level with a noise multiplier σ, ensuring that we protect per trajectory rather than per fragment. The parameters of the noise prediction network are updated by,

$$\mathbb{E}_\tau \left[\mathbb{E}_{(\tau^s,S)\in\tau} \left[\text{Clip}_C \left(\nabla \mathcal{L}(\theta, (\tau^s, S)) \right) \right] + C\mathcal{N}(0, \sigma^2 \mathbb{I}) \right]. \quad (7)$$

We present the workflow of PrivORL-j in Algorithm 2.

C. Dataset Synthesis via PrivORL-j

PrivORL-j just generates trajectory fragments. This section introduces how to generate a complete trajectory.

For the initial fragments of each trajectory, we first sample a Gaussian noise τ_T^s (T is the number of noising steps), and the conditional input s should be 0. Then, we use the diffusion transformer to denoise τ_T^s to the less noisy τ_{T-1}^s. For any t in range of 1 to T, this denoising entails the computation of the estimated noise mean μ for τ_{t-1}^s, defined as [21],

$$\mu = \frac{1}{\sqrt{\alpha_t}} \left(\tau_t^s - \frac{1-\alpha_t}{\sqrt{1-\beta_t}} e_\theta \left(\tau_{t-1}^s, t, S \right) \right), \quad (8)$$

where α_t and β_t are hyper-parameters as introduced in Section II-C, and t is the current step. The τ_{t-1}^s is defined as [21],

$$\tau_{t-1}^s = \mu + \sigma_t e, \quad e \sim \mathcal{N}(0, \mathbb{I}). \quad (9)$$

The σ_t regulates the magnitude of the added noise and is predefined by users. Repeating this process until $t = 0$, the τ_0^s indicates the synthetic trajectory fragment. Then, we use the final transition of synthetic trajectory fragments as conditional inputs and iterate the synthesis process until either the terminal state is generated—indicated by the fragment trajectory containing the terminal signal $d = 1$—or the predefined maximum trajectory length is reached. Figure 2 presents the visualization for trajectory synthesis and fragment stitching.

D. Processes for Discrete Variants

We uniformly represent datasets with real-valued variables to leverage diffusion models' strengths in handling continuous data [9]. For discrete variants, we embed them into the continuous variable using one-hot encoding before the training phase. During synthesis, we apply an argmax post-processing step to map the sampled synthetic continuous outputs back to valid discrete variants.

VII. Privacy Analysis

Since PrivORL leverages DP-SGD to train the synthesizers, its privacy analysis is the same as DP-SGD. Specifically, with the RDP accountant [49], K finetuning steps of PrivORL will cost $(\alpha, K\gamma)$-RDP, where α and γ denote the privacy parameters of RDP, and γ is the upper bound of a function. The ultimate DP cost of PrivORL is $\left(\gamma + \frac{\log 1/\delta}{\alpha - 1}, \delta\right)$-DP. Please refer to Appendix B for more details. Besides, we present the detailed parameters of DP-SGD, like noise scale and sampling probability, in Table IX of the Appendix. For PrivORL-n, the sampling probability is defined as the ratio of the batch transition size to the total transition dataset size. For PrivORL-j, although it is trained on trajectory fragments, the sampling probability is calculated as the ratio of the batch trajectory size to the overall trajectory dataset size, ensuring trajectory-level DP protection. We use the RDP for fair comparisons with baselines. Appendix I-A presents results under Privacy Random Variable (PRV) [50], which provides a tighter privacy analysis than RDP.

According to the post-processing theorem [40], if an algorithm satisfies (ϵ, δ)-DP, then any form of post-processing will not incur additional privacy loss. Therefore, agents trained on DP synthetic datasets (consisting of transitions or trajectories) without increasing the risk of data leakage.

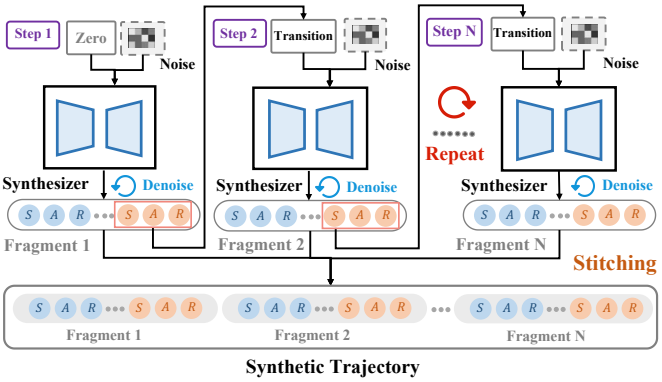


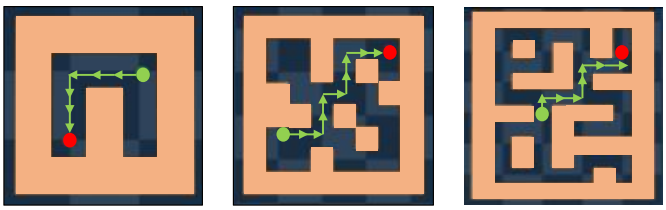
Fig. 2: Visualizations for trajectory synthesis and fragment stitching. The initial condition is zero-padded, and the synthesizer denoises to generate a fragment. We then use the final transition of each fragment as a conditional input and iterate until a terminal state is produced. Finally, the fragments are stitched sequentially to form a complete synthetic trajectory.

VIII. Experimental Setup

A. Investigated Tasks and Datasets.

We conduct the experiments across three domains from D4RL [47]: `Maze2D` [47], `Kitchen` [51], and `Mujoco` [52], all of which are commonly used in offline RL research [5], [9], [53]. D4RL is a benchmark specifically designed for evaluating offline RL algorithms.

Each environment comprises various tasks, each featuring similar yet distinct map or robot configurations. For example, in `Maze2D`, the agent controls the same robot across different tasks but is required to achieve various goals on different maps, as presented in Figure 3. To protect the privacy of the real dataset, we select the pre-training and sensitive datasets from the same domain but with different tasks. For instance, if we designate `Maze2D-umaze` as the sensitive dataset requiring protection, `Maze2D-medium` and `Maze2D-large`



(a) maze2d-umaze (b) maze2d-medium (c) maze2d-large

Fig. 3: An illustrative example from `Maze2D`. The green and red dots represent the start and end points of the 2D ball.

are designated as pre-training datasets. Further details on the selection of pre-training and sensitive datasets and processing of D4RL for transition and trajectory synthesis to match the real-application requirements are provided in Appendix D.

For the downstream task, we selected three state-of-the-art offline RL algorithms widely used [53], [9], including EDAC [54], IQL [55], and TD3PlusBC [56]. Our implementations of offline RL algorithms are based on the open-source repository, CoRL [53], with consistent hyper-parameters settings. We refer to codes released in the repository [9], [10], implementing the offline RL dataset synthesizers. We elaborate on the details of hyper-parameters in Appendix F-A. Besides, we discuss the details of the curiosity module in Appendix F-B. Unless otherwise specified, the experimental configurations are the same as those in Section IX-A.

B. Baselines

We compare PrivORL-n with both PATE-GAN and PATE-GAN with pre-training (PrePATE-GAN) [28]. We also consider PGM [17] and PrivSyn [16], which are two marginal-based DP dataset synthesis methods. They achieve state-of-the-art performance in the 'SynMeter' library [57]. These baselines are originally implemented for DP tabular synthesis, and we edit them for DP transition synthesis.

We present variants of PrivORL-n evaluated in ablation studies to assess the role of curiosity-driven pre-training.

- **NonPrePrivORL (DPDM [29]).** This baseline omits pre-training on public datasets for PrivORL-n, exploring the significance of pre-training in transition synthesis.
- **NonCurPrivORL (PDP-Diffusion [24]).** This baseline excludes the curiosity module in the pre-training. This variant is equivalent to the pre-training DP diffusion method proposed by Ghalebikesab et al. [24].

For DP trajectory synthesis, we compare PrivORL-j with DP-Transformer [30] used in DP text synthesis. The second baseline is PrivORL-j-U, which does not consider the temporal relationship in the fragment trajectory. Specifically, we use U-Net as the noise prediction network instead of the transformer, and the other components, like fragment synthesizing, are the same as PrivORL-j. We elaborate on baselines in Appendix E.

C. Evaluation Metrics

We outline the principles of DP dataset synthesis in Appendix H-A and introduce the evaluation metrics as follows. We elaborate on the details of these metrics in Appendix H-B

Averaged Cumulative Return. This metric assesses the utility of a synthetic dataset by measuring the average total reward a trained agent accumulates over multiple trajectories in real environments. The higher normalized returns, scaled to [0, 100], indicate that the trained agent has better performance and the synthetic dataset is of higher utility [47]. we use "*normalized return*" for simplicity.

Marginal & Correlation. *Marginal* uses the mean Kolmogorov-Smirnov [58] statistic to measure the maximum distance between empirical cumulative distribution functions of each dimension. *Correlation* measures differences in pairwise Pearson rank correlations [31]. Scores range from 0 to 1, with higher values indicating greater fidelity.

TrajScore. Inspired by BERTScore [59], TrajScore uses pre-trained MLPs in an autoencoder as a trajectory encoder, similar to BERT's embedding layer [60], to compute trajectory embeddings. It calculates similarity via cosine similarity between generated and real trajectories.

IX. EMPIRICAL EVALUATIONS

This section first compares the effectiveness of PrivORL-n and PrivORL-j with baselines in downstream tasks. Then, we assess the fidelity between the synthetic and real datasets. Next, we evaluate the DP-protected synthetic datasets against MIA and analyze the impact of hyperparameters and privacy budgets on our methods. Finally, we conduct an ablation study to emphasize the importance of curiosity and pre-training.

A. The Utility of Synthetic Datasets

Experiment Design. This experiment evaluates the utility of the DP synthetic dataset with the size of 1×10^6 transitions for transition-level synthesis and 5×10^3 trajectories for trajectory-level synthesis using PrivORL-n and PrivORL-j. We also compare the utility of synthetic and real datasets. The data synthesizer is pre-trained for ten epochs on a public dataset and then fine-tuned for five epochs on a sensitive dataset, under $\epsilon = \{1, 10\}$. The privacy budget δ is not sensitive in our analysis and is set to 1×10^{-6} across all experiments [62]. An epoch means one complete pass through the entire dataset. The hyper-parameter of the curiosity rate is set at 0.3. We provide further details on the selection of pre-training and sensitive datasets in Appendix D. We train agents for 5×10^5 timesteps. Appendix D notes the small trajectory size in the `MuJoco` domain dataset, so we exclude it from trajectory synthesis.

Result Analysis. For transition-level DP synthesis, Table I presents that agents trained on synthetic transitions using PrivORL-n achieve the highest normalized returns across all tasks. In the `Maze2D` domain, under $\epsilon = \{1, 10\}$, agents trained on synthetic transitions from PrivORL-n achieve an average normalized return of 51.9 $(= (63.2 + 32.0 + 62.4 + 60.1 + 73.5 + 50.3 + 30.6 + 33.8 + 61.3)/9)$ and 69.3 $(= (69.1 + 45.6 + 80.6 + 70.3 + 90.7 + 81.0 + 60.3 + 50.4 + 75.3)/9)$ across three types of sensitive datasets. This performance surpasses the baseline returns of 11.7, 3.4, 2.0, 3.2, and 18.9, 3.9, 5.1, 7.2 (calculated similarly to that of PrivORL-n),

TABLE I: The normalized returns of agents trained on synthetic transitions using PrivORL-n and baselines under privacy budget $\epsilon = \{1, 10\}$ and real datasets. 'no-DP' means agents trained on real datasets. We show the mean $\pm$ standard deviation of the performance averaged over five seeds. The best score is marked with the gray color box.

Domains	Real Dataset	Methods	EDAC [61]			IQL [55]			TD3PLUSBC [56]		
			$\epsilon = 1$	$\epsilon = 10$	no-DP	$\epsilon = 1$	$\epsilon = 10$	no-DP	$\epsilon = 1$	$\epsilon = 10$	no-DP
Maze2D	umaze	PGM	17.2 ± 10.4	32.8 ± 9.1	70.8 ± 13.6	30.1 ± 3.0	41.6 ± 0.9	71.0 ± 2.3	-12.4 ± 3.0	-8.5 ± 2.9	73.6 ± 3.2
		PrivSyn	0.1 ± 10.3	1.3 ± 11.9		0.0 ± 1.5	2.9 ± 2.3		3.0 ± 3.0	5.7 ± 4.0	
		PATE-GAN	-8.2 ± 12.0	-16.8 ± 13.8		-10.5 ± 5.9	-14.4 ± 7.8		1.5 ± 4.2	9.1 ± 6.2	
		PrePATE-GAN	12.1 ± 6.0	18.4 ± 4.8		12.4 ± 10.3	20.2 ± 4.5		15.3 ± 6.4	26.4 ± 2.3	
		PrivORL-n	63.2 ± 10.1	69.1 ± 14.5		60.1 ± 8.6	70.3 ± 2.1		30.6 ± 11.8	60.3 ± 6.3	
	medium	PGM	12.1 ± 7,7	20.3 ± 8.7	73.0 ± 10.2	35.5 ± 1.7	46.8 ± 2.4	93.1 ± 10.7	4.3 ± 2.1	7.7 ± 1.0	53.3 ± 0.3
		PrivSyn	0.2 ± 1.4	-6.3 ± 5.2		2.0 ± 1.5	4.0 ± 2.5		30.0 ± 12.2	31.6 ± 10.0	
		PATE-GAN	10.0 ± 3.0	15.5 ± 5.9		17.7 ± 0.3	25.7 ± 0.7		15.3 ± 12.9	20.0 ± 10.2	
		PrePATE-GAN	2.3 ± 5.1	14.3 ± 6.3		-4.2 ± 3.0	-7.0 ± 2.3		-2.2 ± 2.0	-5.4 ± 1.1	
		PrivORL-n	32.0 ± 3.0	45.6 ± 1.9		73.5 ± 13.2	90.7 ± 8.6		33.8 ± 8.3	50.4 ± 6.4	
	large	PGM	0.4 ± 1.0	3.1 ± 3.7	89.9 ± 6.3	16.7 ± 3.2	21.9 ± 5.4	85.9 ± 0.2	1.6 ± 2.0	4.0 ± 1.1	96.8 ± 3.2
		PrivSyn	-8.2 ± 1.2	-9.9 ± 0.6		0.4 ± -5.2	2.5 ± 1.6		3.4 ± 1.1	2.9 ± 2.3	
		PATE-GAN	-14.3 ± 2.0	-5.2 ± 1.8		2.9 ± 0.7	5.9 ± 0.2		3.6 ± 1.0	5.8 ± 0.0	
		PrePATE-GAN	-6.6 ± 4.3	-2.4 ± 2.2		0.0 ± 0.0	0.4 ± 1.2		0.0 ± 0.0	0.0 ± 0.0	
		PrivORL-n	62.4 ± 12.2	80.6 ± 14.5		50.3 ± 8.1	81.0 ± 11.8		61.3 ± 4.7	75.3 ± 13.2	
Kitchen	partial	PGM	0.0 ± 0.0	3.5 ± 5.0	10.0 ± 0.0	2.0 ± 1.5	1.5 ± 1.7	40.0 ± 2.5	2.0 ± 0.4	2.5 ± 1.8	18.0 ± 6.0
		PrivSyn	0.0 ± 0.0	0.0 ± 0.0		0.0 ± 0.0	0.0 ± 0.0		0.0 ± 0.0	0.2 ± 0.3	
		PATE-GAN	0.0 ± 0.0	0.0 ± 0.0		0.0 ± 0.0	0.0 ± 0.0		1.9 ± 0.6	4.2 ± 6.5	
		PrePATE-GAN	0.0 ± 0.0	0.0 ± 0.0		0.0 ± 0.0	0.0 ± 0.0		0.0 ± 0.0	0.8 ± 1.2	
		PrivORL-n	0.0 ± 0.0	2.5 ± 1.5		12.5 ± 2.5	25.5 ± 2.5		7.5 ± 1.5	11.5 ± 0.0	
MujoCo	halfcheetah	PGM	0.0 ± 0.0	0.2 ± 0.1	60.8 ± 1.9	1.5 ± 0.0	4.5 ± 0.5	48.3 ± 0.5	0.0 ± 4.0	1.6 ± 0.5	48.5 ± 0.3
		PrivSyn	0.0 ± 0.0	0.0 ± 0.3		0.0 ± 1.6	2.4 ± 0.4		0.4 ± 1.0	1.3 ± 0.5	
		PATE-GAN	-2.8 ± 0.5	-3.4 ± 0.6		1.8 ± 3.1	1.7 ± 0.5		4.0 ± 5.1	2.5 ± 0.6	
		PrePATE-GAN	3.0 ± 1.3	3.0 ± 0.3		1.8 ± 0.9	3.6 ± 0.3		5.3 ± 2.3	5.6 ± 1.9	
		PrivORL-n	38.7 ± 4.9	48.8 ± 9.7		25.2 ± 0.7	36.9 ± 2.4		27.4 ± 3.2	45.2 ± 3.2	

TABLE II: The normalized returns of agents trained on synthetic trajectories using PrivORL-j and baselines under $\epsilon = \{1, 10\}$ and real datasets. 'PrivORL-j-U' denotes a variant of PrivORL-j that uses a U-Net as the noise prediction network instead of a transformer, while retaining the other components.

Domains	Real Dataset	Methods	EDAC [61]			IQL [55]			TD3PLUSBC [56]		
			$\epsilon = 1$	$\epsilon = 10$	no-DP	$\epsilon = 1$	$\epsilon = 10$	no-DP	$\epsilon = 1$	$\epsilon = 10$	no-DP
Maze2D	umaze	PrivORL-j-U	11.2 ± 9.0	32.0 ± 6.1	70.8 ± 13.6	24.6 ± 4.4	28.9 ± 5.0	71.0 ± 2.3	26.8 ± 3.7	38.8 ± 2.9	73.6 ± 3.2
		DP-Transformer	28.4 ± 4.1	39.8 ± 5.1		25.9 ± 6.5	41.2 ± 3.3		28.2 ± 2.9	49.3 ± 2.6	
		PrivORL-j	45.5 ± 9.1	52.2 ± 2.6		42.1 ± 2.4	49.8 ± 6.8		38.7 ± 6.5	49.9 ± 4.9	
	medium	PrivORL-j-U	13.5 ± 1.2	16.4 ± 5.1	73.0 ± 10.2	5.5 ± 1.0	14.1 ± 2.9	93.1 ± 10.7	6.7 ± 0.2	10.8 ± 1.6	53.3 ± 0.3
		DP-Transformer	10.2 ± 0.4	19.0 ± 4.8		18.1 ± 2.2	32.8 ± 4.4		7.6 ± 1.0	26.6 ± 0.4	
		PrivORL-j	31.5 ± 1.1	35.0 ± 3.4		23.4 ± 5.2	49.3 ± 1.7		31.1 ± 10.6	38.0 ± 1.6	
	large	PrivORL-j-U	10.9 ± 0.6	19.8 ± 0.1	89.9 ± 6.3	9.8 ± 0.4	31.9 ± 1.2	85.9 ± 0.2	9.9 ± 0.3	14.3 ± 1.2	96.8 ± 3.2
		DP-Transformer	7.6 ± 3.5	21.8 ± 2.4		4.9 ± 0.8	23.4 ± 3.1		6.3 ± 0.2	28.3 ± 3.2	
		PrivORL-j	20.5 ± 0.2	30.5 ± 6.0		9.9 ± 0.3	37.7 ± 7.0		29.5 ± 6.6	35.6 ± 2.6	
Kitchen	partial	PrivORL-j-U	0.0 ± 0.0	0.0 ± 0.0	10.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	40.0 ± 2.5	0.0 ± 0.0	0.0 ± 0.0	18.0 ± 6.0
		DP-Transformer	0.0 ± 0.0	0.0 ± 0.0		0.0 ± 0.0	2.5 ± 1.8		0.0 ± 0.0	0.0 ± 0.0	
		PrivORL-j	0.0 ± 0.0	0.0 ± 0.0		7.5 ± 1.0	13.8 ± 7.5		5.0 ± 1.8	8.3 ± 2.5	

from PGM, PrivSyn, PATE-GAN, and PrePATE-GAN, under $\epsilon = \{1, 10\}$. In Maze2D, where the transition dimension is low at just 11, baselines perform well in certain scenarios; for example, agents trained on synthetic transitions from PGM using the IQL in Maze2D-medium achieve a normalized return of 41.6. In Kitchen and MujoCo, where the transition dimensions are 130, the baseline methods struggle to perform effectively. PrivORL-n achieves average return scores of 13.1 $(= (2.5+25.5+11.5)/3)$ and 43.6 $(= (48.8+36.9+45.2)/3)$, under $\epsilon = 10$, outperforming baselines.

Table IV presents the performance of synthesizers only pre-trained on public datasets, without fine-tuning on sensitive datasets. We observe that fine-tuning on sensitive datasets significantly improves the synthetic quality. Synthesizers trained only on the pre-training datasets of Maze2D-umanze achieve only 6.6 downstream agent performance. Although public datasets may resemble sensitive datasets, the training objectives differ substantially. For instance, the goal position and the layout map in Maze2D vary across datasets. Thus, a synthesizer pretrained solely on public data may not generalize well to sensitive datasets, as the optimal trajectories and MDP (introduced in Section II-A) are task-specific.

For trajectory-level synthesis, Table II shows that PrivORL-j achieves better performance than baselines. In the Maze2D, under $\epsilon = \{1, 10\}$, agents trained on synthetic trajectories from PrivORL-j achieve average normalized returns of 30.2 and 41.8, surpassing 13.2 and 23.0 obtained by PrivORL-j-U, and DP-Transformer's 15.2 and 31.4. Thus, PrivORL-j achieves average 15.0 and 10.4 higher returns than DP-Transformer. In Kitchen, PrivORL-j still outperforms baselines, presenting its better ability to handle complex trajectories.

Agents trained on real datasets achieve average returns of 78.6, 22.6, and 52.5 across Maze2D, Kitchen, and MujoCo, while PrivORL-n obtains 69.3, 13.0, and 43.5 at $\epsilon = 10$. The synthetic dataset exhibits comparable utility to the real dataset. We find that in Kitchen, the agents trained using EDAC have 0.0 averaged returns. Prior works [47], [53], [63] show that no single existing offline RL algorithm excels across all datasets

TABLE III: Comparison of fidelity metrics of the synthetic transitions using PrivORL-n and baselines ($\epsilon = 10$) to real datasets. The highest scores are highlighted in bold font. 'Margin.' and 'Correlat.' are abbreviations for 'Marginal' and 'Correlation.'

Domains	Real Dataset	PGM [17]		PrivSyn [16]		PATE-GAN [28]		PrePATE-GAN		PrivORL-n	
		Margin.	Correlat.	Margin.	Correlat.	Margin.	Correlat.	Margin.	Correlat.	Margin.	Correlat.
Maze2D	umaze	0.793	0.983	0.784	0.969	0.672	0.844	0.784	0.969	**0.948**	**0.994**
	medium	0.801	**0.995**	0.763	0.973	0.721	0.911	0.763	0.793	**0.947**	0.983
	large	0.803	0.982	0.832	0.997	0.589	0.912	0.713	0.974	**0.937**	**0.997**
Kitchen	partial	0.783	0.697	0.737	0.806	0.647	0.788	0.695	0.819	**0.861**	**0.901**
Mujoco	halfcheetah	0.776	0.921	0.862	0.946	0.768	0.937	0.849	0.945	**0.949**	**0.982**
Average		0.791	0.916	0.796	0.938	0.679	0.878	0.761	0.900	**0.928**	**0.971**

TABLE IV: Average normalized returns of agents trained on synthetic transitions ($\epsilon = 10$) using PrivORL-n, with and without fine-tuning on sensitive datasets under IQL algorithm.

Domains	Dataset	Only pretraining	PrivORL-n
Maze2D	umaze	6.6 ± 0.0	70.3 ± 2.1
	medium	6.6 ± 1.1	90.7 ± 8.6
	large	7.8 ± 4.7	81.0 ± 11.8
Kitchen	partial	0.0 ± 0.0	25.5 ± 2.5
Mujoco	halfcheetah	14.4 ± 3.7	36.9 ± 2.4

due to the inherent instability training problem in offline RL. Thus, it is usual for certain methods to appear ineffective for specific tasks. These results present that PrivORL-n and PrivORL-j both achieve better synthetic utility than baselines.

B. The Fidelity of Synthetic Datasets

Experiment Design. This section investigates whether PrivORL-n and PrivORL-j can generate transitions and trajectories with greater fidelity than baselines under privacy budget $\epsilon = 10$. To visualize the synthetic distribution, we sample 500 transitions from each synthetic dataset and use t-SNE [64] to visualize them in a two-dimensional space.

Result Analysis. Table III compares the marginal and correlation statistics of the synthetic transitions using PrivORL-n under $\epsilon = 10$ to real datasets. We observe that PrivORL-n outperforms the baselines in both marginal and correlation statistics. For correlation statistics, PrivORL-n similarly excels with an average score of 0.971, exceeding baseline scores of 0.916, 0.938, 0.878, and 0.900. Marginal provides insights into the accuracy of single-variable distributions, while correlation effectively introduces synthetic data and preserves the relational structure observed in real data [58], [31]. Based on these results, we conclude that while all methods effectively synthesize transitions with strong pairwise relationships between variables as observed in real transitions, PrivORL-n achieves higher similarity in the distribution of individual variables between the synthetic and real transitions than the baseline methods. Table XIII in Appendix I-A of [23] presents that PrivORL-j achieves an average TrajScore of 0.902, which is 0.124 higher than DP-Transformer (0.778), under $\epsilon = 10$.

Figure 5 shows the distribution of synthetic transitions from PrivORL-n and baselines compared to real transitions. These results highlight the strength of PrivORL-n strength in handling high-dimensional transitions. The

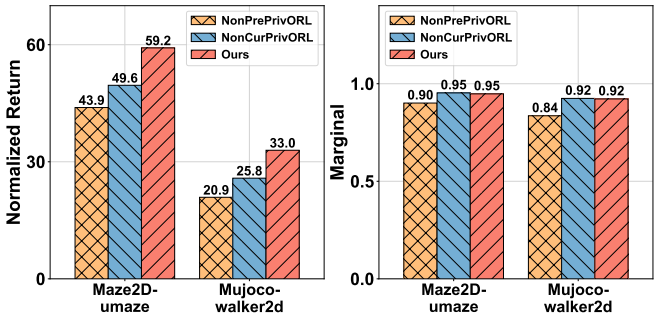


Fig. 4: The averaged normalized return across three RL algorithms under $\epsilon = 10$ and marginal as achieved by (1) PrivORL-n (Ours), (2) NonPrePrivORL, and (3) NonCurPrivORL.

number of transition dimensions is 11 and 130 in the `Maze2D-medium` and `Kitchen-partial` datasets. Baselines match the real distribution well in `Maze2D-medium` but fail in `Kitchen-partial`, while PrivORL-n consistently generates transitions resembling the real dataset.

C. Ablation Study

Experiment Design. These studies aim to explore the importance of pre-training and curiosity modules and how the PrivORL performs while we introduce the curiosity module in the fine-tuning stage. We conduct experiments on transition synthesis. We conduct experiments under $\epsilon = 10$. As mentioned in Section VIII-B, 'NonCurPrivORL' and 'NonPrePrivORL' are equal to applying DPDM and PDP-Diffusion.
Result Analysis. Figure 4 shows the utility and fidelity of synthetic transitions using various methods. Notably, in `Maze2D-umaze`, the removal of the pre-training stage and the curiosity module from PrivORL-n leads to a decrease in average normalized returns of trained agents—by 25.9% $(1 - 43.9/59.2) \times 100\%$ and 16.3% $(1 - 49.6/59.2) \times 100\%$, and Besides, 36.7% $= (1 - 20.9/33.0) \times 100\%$ and 21.9% $= (1 - 25.8/33.0) \times 100\%$ for `Mujoco-walker2d`. Table V shows that using FineCurPrivORL leads to a degradation of the utility of synthetic transitions. Specifically, the average returns decrease from 70.3, 90.7, 81.0, 25.5, 36.9 to 54.1, 65.9, 54.1, 5.2, 18.7. The curiosity module adds novelty feedback to the training, increasing variability, which destabilizes fine-tuning.

D. Defending against MIAs

Experiment Design. We explore whether PrivORL-n can defend against MIAs under DP protection. We use a white-box

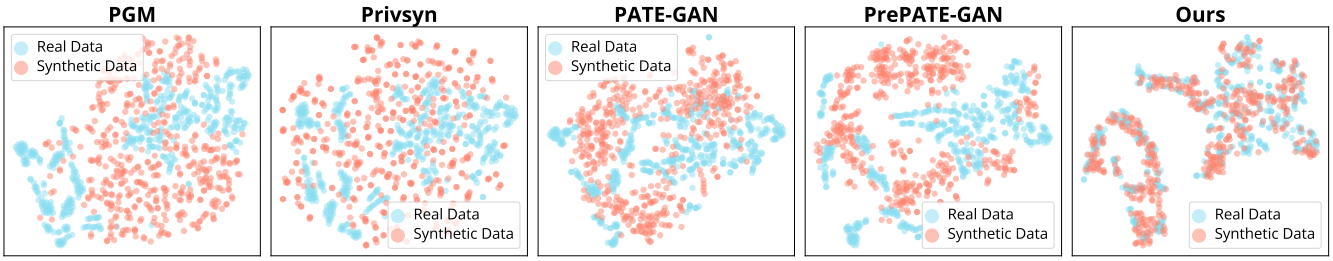


Fig. 5: The t-SNE visualizations of data distribution for `Kitchen-partial`. We synthesize the dataset using PrivORL-n and baselines under $\epsilon = 10$. We show the t-SNE visualizations for `Maze2D-medium` in Figure 7, Appendix H-E of [23].

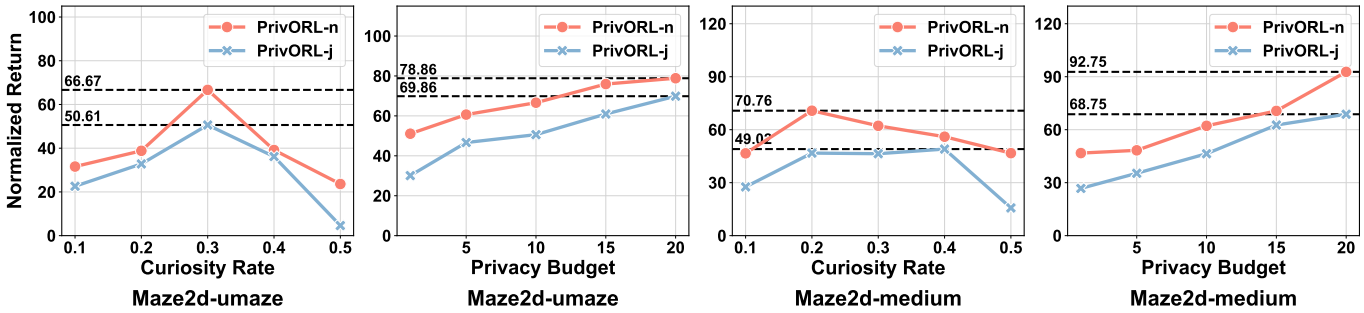


Fig. 6: The average normalized returns of agents trained using three offline algorithms on synthetic transitions and trajectories using PrivORL-n and PrivORL-j. These experiments and trajectories vary with different curiosity rates and privacy budgets.

TABLE V: The normalized returns of agents trained on a DP synthetic dataset using IQL ($\epsilon = 10$).

Method	Maze			Kitchen	Mujoco
	umaze	medium	large	umaze	halfcheetah
FineCurPrivTranR	54.1	65.9	54.1	5.2	18.7
Ours	70.3	90.7	81.0	25.5	36.9

MIA [12] to attack synthesizers, aiming to identify members of a sensitive dataset using the synthetic dataset. We use True Positive Rate@10%False Positive Rate (TPR@10%FPR) and TPR@1%FPR to evaluate the attacker's performance, and a higher metric means a higher attack success rate. Please refer to more details of implementations of MIA in [12]. Following prior research [12], [14], the fixed FPR is set as a low rate, e.g., TPR@1%FPR indicates the FPR threshold at 1%. We introduce more about this metric in Appendix H-B.

Result Analysis. We present the TPR@10%FPR and TPR@1%FPR of the MIA for diffusion models [12] under $\epsilon = \{1, 10, \infty\}$ in Table VI, where "∞" means without DP protection. From this table, we observe that the MIA is ineffective against PrivORL-n, approximating the effectiveness of random guessing. These results show that PrivORL-n can synthesize transitions without privacy leakage of real sensitive transitions. Setting the privacy budget at 10 provides an effective defense with negligible differences compared to a budget set at 1. Although as ϵ increases, PrivORL-n presents a reduction in synthetic transitions utility. We still advise using smaller ϵ values, such as 10, as recommended by previous research [14], [24], [65], to defend against unknown attacks.

TABLE VI: The TPR@10%FPR / TPR@1%FPR (%) of MIA [12] under different ϵ. The privacy budget "∞" means training PrivORL-n without DP protection.

Domains	Real Dataset	ϵ		
		1	10	∞
`Maze2D`	umaze	11.9 / 1.5	11.4 / 1.9	31.0 / 12.8
	medium	9.2 / 0.7	9.7 / 0.8	30.9 / 9.2
	large	8.9 / 0.7	10.9 / 0.6	27.6 / 8.9
`Kitchen`	partial	10.6 / 1.6	10.1 / 1.4	96.4 / 86.0
`Mujoco`	halfcheetah	10.8 / 1.7	11.0 / 1.6	95.9 / 86.0

E. Hyper-parameter and Privacy Budget

Experiment Design. This experiment studies how PrivORL-n and PrivORL-j perform under different hyper-parameter settings: (1) curiosity rate, $p = \{0.1, 0.2, 0.3, 0.4, 0.5\}$, (2) privacy budget, $\epsilon = \{1, 5, 10, 15, 20\}$, (3) horizon for PrivORL-j, $H = \{8, 16, 32, 64, 128\}$ (presented in Appendix I-C of [23]).
Result Analysis. Figure 6 presents the average normalized returns of agents trained using the studied algorithms. We observe that the performance of PrivORL-n and PrivORL-j both initially increase but then decrease as curiosity rates continue to rise. "There is no such thing as a free lunch." Excessively pursuing diversity can also increase the risk of generating erroneous data and affect the training. This highlights the need for a moderate curiosity rate for best results. Optimal curiosity rates vary across different tasks to maximize synthesis performance, e.g., for `Maze2D-umaze` and `Maze2D-medium`, the optimal curiosity rates for PrivORL-n are 0.3 and 0.2, and for PrivORL-j are 0.3 and 0.3. We observe that with an increase in the privacy budget ϵ, there is a corresponding upward trend in the performance of the

TABLE VII: The normalized return and marginal of synthetic transitions as achieved by (1) our method ($\epsilon = 10$), PrivORL-n, and (2) NonPrivORL ($\epsilon = \infty$). 'Difference' means the results of NonPrivORL minus that of PrivORL-n.

Method	Normalized return			Marginal		
	Maze		Mujoco	Maze		Mujoco
	medium	large	halfcheetah	medium	large	halfcheetah
NonPrivORL	74.8	80.5	54.9	0.98	0.97	0.97
Ours	66.7	79.0	43.6	0.95	0.94	0.95
Difference	8.1	1.5	11.3	0.03	0.03	0.02

synthetic data. As the privacy budget ϵ increases, introducing less noise during the training of synthesizers, the quality of synthetic datasets shows improvements [14], [16].

Selecting an appropriate curiosity rate for different datasets is crucial. However, tuning the curiosity rate requires repeated access to the sensitive dataset [40], which can degrade the overall privacy guarantee and negatively impact the quality of the synthetic data. A practical mitigation strategy is to tune the curiosity rate on a public or non-sensitive dataset and then apply the optimal value to the sensitive data. Figure 6 shows that a curiosity rate of approximately 0.3 achieves strong performance across datasets. Therefore, we adopt this fixed value without further tuning on sensitive data.

F. PrivORL without Privacy Protection

Experiment Design. This experiment focuses on DP transition synthesis. We explore how the DP harms the synthetic performance of PrivORL-n ($\epsilon = 10$). We compare PrivORL-n with the method 'NonPrivORL', which fine-tunes diffusion models on sensitive datasets without DP protection (i.e., $\epsilon = \infty$).

Result Analysis. Table VII illustrates that PrivORL-n, on average, leads to a 7.0 $(= (8.1 + 1.5 + 11.3)/3)$ reduction in normalized returns for agents trained using four different algorithms, to meet the requirements of the DP constraint. However, in terms of high-level statistics, PrivORL-n results in only a 0.03 $(= (0.03 + 0.03 + 0.02)/3)$ decrease in the marginal. It is understood that incorporating the DP framework may damage transition synthesis performance, as observed in various prior studies [14], [24]. For example, in DP image synthesis, PrivImage also experiences an approximate 11.6% performance reduction in downstream tasks under $\epsilon = 10$. Therefore, a reduction in performance is deemed acceptable for privacy protection. These results indicate that further development is necessary to mitigate the performance reduction caused by DP. Besides, Appendix I-B shows learning curves of the downstream agent over training steps.

We discuss the efficiency of PrivORL-n compared to baselines and the application scenario of offline RL dataset synthesis in Appendix G and H-D of [23], respectively. Besides, Appendix I-B of our full paper [23] shows how scaling up improves the performance of synthetic transitions. We discuss the limitations of our methods in Appendix H-C.

X. Related Work

We discuss related work briefly here, and we provide a more comprehensive discussion of related works in Appendix J.

DP Dataset Synthesis. DP dataset synthesis methods fall into two categories: (1) *Marginal-based approaches*, which replicate marginal and joint distributions of variables [16], [17], [66]. These methods work well for small tabular data but struggle with discrete values, high-dimensional attributes, and large datasets [16]. (2) *Generative model-based approaches*, which train DP synthesizers such as GANs [28], [45] and diffusion models [21], [24] using DP-SGD [22]. While PATE-GAN [28] and DP-CGAN [67] perform well on `MNIST` [68], they degrade on larger datasets. Recent work shows diffusion models outperform GANs in DP synthesis [65]. Sabra et al. [24] propose to pre-training DP-Diffusion on public data and fine-tune on sensitive data.

Offline RL with Synthetic Data. Diffusion models augment datasets with synthetic data, widely used in computer vision [14], [69] and suitable for offline RL to address data scarcity [32]. Recent works [70], [71] augmented robotic observations with text-guided diffusion models. SynthER [9] and MTDIFF [10] generate transitions with novel actions. Zhao et al. [72] used transformers for trajectory synthesis. Prior work [11] explored multi-agent dataset synthesis. However, generative models still risk MIAs, a concern from images [12], [13] to offline RL datasets [5], [1], [2], [73].

Applications of Offline RL. Offline RL excels in healthcare [34], [35], energy management [74], [75], autonomous driving [76], [77], and recommendation systems [78], [79]. In healthcare, ethical constraints limit online RL. Mila et al. [34] optimized diabetes treatment policies, while Emerson et al. [80] determined insulin doses using offline RL. Offline RL enhances data efficiency in costly data collection scenarios like autonomous driving [76], [77].

XI. Conclusions

This paper is the first to propose offline RL dataset synthesis under the DP framework and introduces PrivORL, which allows users to create datasets from both transition and trajectory levels that closely resemble sensitive datasets while safeguarding their privacy under DP. We identify the importance of synthesizing diverse offline RL datasets and propose the curiosity module to improve the diversity of synthetic datasets. PrivORL initially pre-trains diffusion models on public datasets, using feedback from the curiosity module to diversify the synthetic datasets. The pre-trained model is then fine-tuned on sensitive datasets with DP-SGD. Finally, the fine-tuned diffusion model generates synthetic datasets. For trajectory-level DP synthesis, we introduce a transformer to the diffusion model to capture long-range temporal dependencies. To handle the high dimensionality of trajectories, we split trajectories into fragments and use a conditional synthesizer to model fragment relationships, enabling seamless trajectory stitching. Experiments show that PrivORL achieves higher utility for various downstream agents' training in both transition and trajectory synthesis, compared to baselines. This work aims to advance the privacy-preserving sharing of offline RL datasets and further the progress of open offline RL research.

ACKNOWLEDGMENT

This paper was partially supported by NSF CNS-2220433, NSF CNS-2213700, and CCF-2217071. Any opinions, findings, conclusions, or recommendations in this material are those of the authors and do not reflect the views of the National Science Foundation.

REFERENCES

[1] X. Pan, W. Wang, X. Zhang *et al.*, "How you act tells a lot: Privacy-leaking attack on deep reinforcement learning." in *AAMAS*, vol. 19, no. 2019, 2019.

[2] M. Gomrokchi, S. Amin, H. Aboutalebi, A. Wong, and D. Precup, "Membership inference attacks against temporally correlated data in deep reinforcement learning," 2022.

[3] S. Chaudhari, P. Aggarwal, V. Murahari *et al.*, "Rlhf deciphered: A critical analysis of reinforcement learning from human feedback for llms," 2024.

[4] Y. He, B. Li, L. Liu, Z. Ba, W. Dong, Y. Li, Z. Qin, K. Ren, and C. Chen, "Towards label-only membership inference attack against pre-trained large language models," *arXiv preprint arXiv:2502.18943*, 2025.

[5] L. Du, M. Chen, M. Sun *et al.*, "ORL-AUDITOR: dataset auditing in offline deep reinforcement learning," in *31st Annual Network and Distributed System Security Symposium, NDSS*, 2024.

[6] Y. Hu, F. Wu, Q. Li, Y. Long, G. Garrido, C. Ge, B. Ding, D. Forsyth, B. Li, and D. Song, "Sok: Privacy-preserving data synthesis," in *2024 IEEE Symposium on Security and Privacy (SP)*, pp. 2–2.

[7] X. Yue, H. Inan, X. Li *et al.*, "Synthetic text generation with differential privacy: A simple and practical recipe," in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, Jul. 2023.

[8] D. Sun, J. Q. Chen, C. Gong, T. Wang, and Z. Li, "Netdpsyn: synthesizing network traces under differential privacy," in *Proceedings of the 2024 ACM on Internet Measurement Conference*, 2024, pp. 545–554.

[9] C. Lu, P. J. Ball, Y. W. Teh, and J. Parker-Holder, "Synthetic experience replay," in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[10] H. He, C. Bai, K. Xu *et al.*, "Diffusion model is an effective planner and data synthesizer for multi-task reinforcement learning," in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. [Online]. Available: https://openreview.net/forum?id=fAdMly4ki5

[11] Z. Zhu, M. Liu, L. Mao, B. Kang, M. Xu, Y. Yu, S. Ermon, and W. Zhang, "Madiff: Offline multi-agent learning with diffusion models," *Advances in Neural Information Processing Systems*, 2024.

[12] T. Matsumoto, T. Miura, and N. Yanai, "Membership inference attacks against diffusion models," in *2023 IEEE Security and Privacy Workshops (SPW)*, pp. 77–83.

[13] N. Carlini, J. Hayes, M. Nasr *et al.*, "Extracting training data from diffusion models," in *32nd USENIX Security Symposium*, 2023, pp. 5253–5270.

[14] K. Li, C. Gong, Z. Li, Y. Zhao, X. Hou, and T. Wang, "{PrivImage}: Differentially private synthetic image generation using diffusion models with {Semantic-Aware} pretraining," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 4837–4854.

[15] S. P. Liew, T. Takahashi, and M. Ueno, "PEARL: Data synthesis via private embeddings and adversarial reconstruction learning," in *International Conference on Learning Representations*, 2022.

[16] Z. Zhang, T. Wang, N. Li *et al.*, "{PrivSyn}: Differentially private data synthesis," in *30th USENIX Security Symposium*, 2021, pp. 929–946.

[17] R. McKenna, D. Sheldon, and G. Miklau, "Graphical-model based estimation and inference for differential privacy," in *Proceedings of the 36th International Conference on Machine Learning, ICML*, vol. 97. PMLR, 2019, pp. 4435–4444.

[18] L. Yang, Z. Zhang, Y. Song, S. Hong, R. Xu, Y. Zhao, W. Zhang, B. Cui, and M.-H. Yang, "Diffusion models: A comprehensive survey of methods and applications," *ACM computing surveys*, vol. 56, no. 4, pp. 1–39, 2023.

[19] X. Li, J. Thickstun, I. Gulrajani, and et al., "Diffusion-lm improves controllable text generation," in *NeurIPS*, 2022.

[20] W. Wu, Y. Zhao, M. Z. Shou, H. Zhou, and C. Shen, "Diffumask: Synthesizing images with pixel-level annotations for semantic segmentation using diffusion models," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 1206–1217.

[21] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," in *Advances in Neural Information Processing Systems*, 2020.

[22] M. Abadi, A. Chu, I. J. Goodfellow, and et al., "Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 308–318.

[23] C. Gong, Z. Liu, K. Li, and T. Wang, "Privorl: Differentially private synthetic dataset for offline reinforcement learning," *arXiv preprint arXiv:2512.07342*, 2025.

[24] S. Ghalebikesabi, L. Berrada, S. Gowal *et al.*, "Differentially private diffusion models generate useful synthetic images," *CoRR*, vol. abs/2302.13861, 2023.

[25] Z.-W. Hong, I. Shenfeld, T.-H. Wang *et al.*, "Curiosity-driven red-teaming for large language models," in *The Twelfth International Conference on Learning Representations*, 2024.

[26] J. He, Z. Yang, J. Shi *et al.*, "Curiosity-driven testing for sequential decision-making process," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 2024, pp. 1–14.

[27] W. Peebles and S. Xie, "Scalable diffusion models with transformers," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2023.

[28] J. Yoon, J. Jordon, and M. van der Schaar, "PATE-GAN: Generating synthetic data with differential privacy guarantees," in *International Conference on Learning Representations*, 2019.

[29] T. Dockhorn, T. Cao, A. Vahdat *et al.*, "Differentially private diffusion models," *Transactions on Machine Learning Research*, 2023.

[30] Z. Bu, Y.-X. Wang, S. Zha, and G. Karypis, "Differentially private optimization on large model at small cost," in *International Conference on Machine Learning*. PMLR, 2023, pp. 3192–3218.

[31] E. C. Fieller, H. O. Hartley, and E. S. Pearson, "Tests for rank correlation coefficients. i," *Biometrika*, vol. 44, pp. 470–481, 1957.

[32] R. F. Prudencio, M. R. O. A. Maximo *et al.*, "A survey on offline reinforcement learning: Taxonomy, review, and open problems," *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[33] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[34] M. Nambiar, S. Ghosh, P. Ong *et al.*, "Deep offline reinforcement learning for real-world treatment optimization applications," in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2023.

[35] M. Fatemi, T. W. Killian, J. Subramanian *et al.*, "Medical dead-ends and learning to identify high-risk states and treatments," in *Advances in Neural Information Processing Systems*, 2021, pp. 4856–4870.

[36] H. Emerson, M. Guy, and R. McConville, "Offline reinforcement learning for safer blood glucose control in people with type 1 diabetes," *Journal of Biomedical Informatics*, vol. 142, p. 104376, 2023.

[37] C. Yu, J. Liu, S. Nemati, and G. Yin, "Reinforcement learning in healthcare: A survey," *ACM Computing Surveys (CSUR)*, no. 1, pp. 1–36, 2021.

[38] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline reinforcement learning: Tutorial, review, and perspectives on open problems," 2020.

[39] A. Kumar, A. Zhou, G. Tucker, and S. Levine, "Conservative q-learning for offline reinforcement learning," in *NeurIPS*, 2020.

[40] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *Theory of Cryptography: Third Theory of Cryptography Conference,*, 2006, pp. 265–284.

[41] C. Gong, K. Li, Z. Lin, and T. Wang, "Dpimagebench: A unified benchmark for differentially private image synthesis," *arXiv preprint arXiv:2503.14681*, 2025.

[42] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, 2017, pp. 5998–6008.

[43] T. Karras, M. Aittala, T. Aila, and S. Laine, "Elucidating the design space of diffusion-based generative models," in *Advances in Neural Information Processing Systems*, 2022.

[44] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *Proceedings of the 34th International Conference on Machine Learning*, vol. 70, 06–11 Aug 2017, pp. 214–223.

[45] Y. Yin, Z. Lin, M. Jin, G. Fanti, and V. Sekar, "Practical gan-based synthetic ip header trace generation using netshare," in *Proceedings of the ACM SIGCOMM 2022 Conference*, 2022, pp. 458–472.

[46] K. Li, C. Gong, X. Li, Y. Zhao, X. Hou, and T. Wang, "From easy to hard: Building a shortcut for differentially private image synthesis," in *IEEE Symposium on Security and Privacy, SP*, 2025.

[47] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine, "D4rl: Datasets for deep data-driven reinforcement learning," 2021.
[48] Y. Burda, H. Edwards, A. Storkey, and O. Klimov, "Exploration by random network distillation," in *International Conference on Learning Representations*, 2018.
[49] I. Mironov, "Renyi differential privacy," *CoRR*, vol. abs/1702.07476, 2017.
[50] S. Gopi, Y. T. Lee, and L. Wutschitz, "Numerical composition of differential privacy," in *Advances in Neural Information Processing Systems*, 2021.
[51] A. Gupta, V. Kumar, C. Lynch, S. Levine, and K. Hausman, "Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning," *arXiv preprint arXiv:1910.11956*, 2019.
[52] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 5026–5033.
[53] D. Tarasov, A. Nikulin, D. Akimov *et al.*, "CORL: Research-oriented deep offline reinforcement learning library," in *3rd Offline RL Workshop: Offline RL as a "Launchpad"*, 2022.
[54] A. Nair, M. Dalal, A. Gupta, and S. Levine, "Accelerating online reinforcement learning with offline datasets," *CoRR*.
[55] I. Kostrikov, A. Nair, and S. Levine, "Offline reinforcement learning with implicit q-learning," in *International Conference on Learning Representations*, 2022.
[56] S. Fujimoto and S. S. Gu, "A minimalist approach to offline reinforcement learning," in *Advances in Neural Information Processing Systems*, vol. 34. Curran Associates, Inc., 2021, pp. 20 132–20 145.
[57] Y. Du and N. Li, "Systematic assessment of tabular data synthesis algorithms," 2024.
[58] F. J. Massey Jr, "The kolmogorov-smirnov test for goodness of fit," *Journal of the American statistical Association*, pp. 68–78, 1951.
[59] T. Zhang, V. Kishore, F. Wu, K. Q. Weinberger, and Y. Artzi, "Bertscore: Evaluating text generation with BERT," in *8th International Conference on Learning Representations*, 2020.
[60] J. Devlin, M.-W. Chang, K. Lee, and K. N. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2018. [Online]. Available: https://arxiv.org/abs/1810.04805
[61] G. An, S. Moon, J.-H. Kim, and H. O. Song, "Uncertainty-based offline reinforcement learning with diversified q-ensemble," *Advances in neural information processing systems*, pp. 7436–7447, 2021.
[62] A. Yousefpour, I. Shilov, A. Sablayrolles *et al.*, "Opacus: User-friendly differential privacy library in PyTorch," *arXiv:2109.12298*, 2021.
[63] T. Seno and M. Imai, "d3rlpy: An offline deep reinforcement learning library," *Journal of Machine Learning Research*, vol. 23, 2022.
[64] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne." *Journal of machine learning research*, vol. 9, no. 11, 2008.
[65] T. Dockhorn, T. Cao, A. Vahdat *et al.*, "Differentially private diffusion models," *Transactions on Machine Learning Research*, 2023.
[66] K. Cai, X. Lei, J. Wei, and X. Xiao, "Data synthesis via differentially private markov random fields," *Proc. VLDB Endow.*, vol. 14, no. 11, p. 2190–2202, 2021.
[67] R. Torkzadehmahani, P. Kairouz, and B. Paten, "DP-CGAN: differentially private synthetic data and label generation," in *IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops*, 2019, pp. 98–104.
[68] Y. LeCun, L. Bottou, Y. Bengio, and et al., "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. [Online]. Available: https://doi.org/10.1109/5.726791
[69] J. Yuan, J. Zhang, S. Sun, P. Torr, and B. Zhao, "Real-fake: Effective training data synthesis through distribution matching," in *The Twelfth International Conference on Learning Representations*, 2024.
[70] T. Yu, T. Xiao, A. Stone *et al.*, "Scaling robot learning with semantically imagined experience," *arXiv:2302.11550*, 2023.
[71] Z. Chen, S. Kiami, A. Gupta, and V. Kumar, "Genaug: Retargeting behaviors to unseen situations via generative augmentation," *arXiv preprint arXiv:2302.06671*, 2023.
[72] Z. Zhao, Z. Ren, L. Yang *et al.*, "Offline trajectory generalization for offline reinforcement learning," *arXiv preprint arXiv:2404.10393*, 2024.
[73] D. Ye, T. Zhu, C. Zhu, D. Wang, S. Shen, W. Zhou *et al.*, "Reinforcement unlearning," *arXiv:2312.15910*, 2023.
[74] X. Zhan, H. Xu, Y. Zhang *et al.*, "Deepthermal: Combustion optimization for thermal power generating units using offline reinforcement learning," in *AAAI*, 2022.
[75] X. Zhang, J. Sun, C. Gong *et al.*, "Mutual information as intrinsic reward of reinforcement learning agents for on-demand ride pooling," *arXiv preprint arXiv:2312.15195*, 2023.
[76] L. Zhang, R. Zhang, T. Wu *et al.*, "Safe reinforcement learning with stability guarantee for motion planning of autonomous vehicles," *IEEE Trans. Neural Networks Learn. Syst.*, 2021.
[77] D. Graves, N. M. Nguyen, K. Hassanzadeh *et al.*, "Learning robust driving policies without online exploration," in *IEEE International Conference on Robotics and Automation*. IEEE, 2021.
[78] S. Wang, X. Chen, D. Jannach *et al.*, "Causal decision transformer for recommender systems via offline reinforcement learning," 2023.
[79] Q. Zhang, J. Liu, Y. Dai *et al.*, "Multi-task fusion via reinforcement learning for long-term user satisfaction in recommender systems," in *The 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022.
[80] H. Emerson, M. Guy, and R. McConville, "Offline reinforcement learning for safer blood glucose control in people with type 1 diabetes," *J. Biomed. Informatics*, 2023.
[81] I. Mironov, K. Talwar, and L. Zhang, "Rényi differential privacy of the sampled gaussian mechanism," *CoRR*, vol. abs/1908.10530, 2019.
[82] Y. LeCun, S. Chopra, R. Hadsell, M. Ranzato, F. Huang *et al.*, "A tutorial on energy-based learning," *Predicting structured data*, 2006.
[83] A. Grover, J. Song, A. Kapoor *et al.*, "Bias correction of learned generative models using likelihood-free importance weighting," *Advances in neural information processing systems*, vol. 32, 2019.
[84] I. O. Tolstikhin, N. Houlsby, A. Kolesnikov, and et al., "Mlp-mixer: An all-mlp architecture for vision," in *Advances in Neural Information Processing Systems*, 2021, pp. 24 261–24 272.
[85] N. Papernot and T. Steinke, "Hyperparameter tuning with renyi differential privacy," in *International Conference on Learning Representations*.

Due to space limitations, please refer to our full version [23] for additional appendices.

APPENDIX A
ETHICAL CONSIDERATIONS

DP dataset synthesis provides strong guarantees against individual data leakage, making it a cornerstone for ethical synthetic data generation. However, DP-based synthesis is not without risks. While privacy is preserved, the injected noise can distort data distributions, potentially introducing bias or reducing fairness in downstream tasks. Furthermore, synthetic data can be misused, for example, to create plausible but biased datasets or to facilitate adversarial attacks such as data poisoning. To mitigate these risks, practitioners should adopt safeguards such as bias auditing, transparency reports, and watermarking synthetic data to trace misuse. Ethical deployment also requires clear communication of limitations and responsible governance to ensure that synthetic data serves its intended purpose without compromising fairness or trust.

APPENDIX B
MORE DETAILS ABOUT DP-SGD

This section introduces how to account for the privacy cost of DP-SGD using the Rényi DP, which is defined as follows.

Definition 2 (Rényi DP [81]): We define the Rényi divergence between two probability distributions Y and N as $D_\alpha\left(Y\|N\right) = \frac{1}{\alpha-1}\ln \mathbb{E}_{x\sim N}\left[\frac{Y(x)}{N(x)}\right]^\alpha$, where $\alpha > 1$ is a real number. A randomized mechanism $\mathcal{A}$ satisfies (α,γ)-RDP, if $D_\alpha\left(\mathcal{A}(D)\|\mathcal{A}(D')\right) < \gamma$ holds for any neighboring dataset D and D'.

Given the batch size B, dataset size N, clip hyper-parameter C, and noise variance σ^2, as described in Section II-B, we

TABLE VIII: Information on pre-training datasets for each dataset we utilize for fine-tuning.

Domain	Fine-tuning Datasets	Pre-training Datasets	Observations	Action Shape	Transition Shape	Transition Size
Maze2D	"maze2d-umaze"	"maze2d-open"	4	2	11	1×10^6
		"maze2d-medium"	4	2	11	2×10^6
		"maze2d-large"	4	2	11	4×10^6
	"maze2d-medium"	"maze2d-open"	4	2	11	1×10^6
		"maze2d-umaze"	4	2	11	1×10^6
		"maze2d-large"	4	2	11	4×10^6
	"maze2d-large"	"maze2d-open"	4	2	11	1×10^6
		"maze2d-umaze"	4	2	11	1×10^6
		"maze2d-medium"	4	2	11	2×10^6
FrankaKitchen	"kitchen-partial"	"kitchen-complete"	60	9	130	3680
		"kitchen-mixed"	60	9	130	136950
MuJoCo	"halfcheetah-medium-replay"	"walker2d-full-replay"	17	6	41	1×10^6
		"halfcheetah-expert"	17	6	41	1×10^6
		"walker2d-medium"	17	6	41	1×10^6

denote the sampling ratio $q = B/N$. The RDP privacy cost for one training step can be obtained via Theorem 1 [81].

Theorem 1: Let p_0 and p_1 be the probability density function of $\mathcal{N}(0, C^2\sigma^2)$ and $\mathcal{N}(1, C^2\sigma^2)$. One training step with DP-SGD satisfies (α, γ_i)-RDP for any γ_i such that,

$$\gamma_i \geq D_\alpha \left((1-q)\, p_0 + q p_1 \, \| p_0 \right). \tag{1}$$

The above theorem shows that the privacy bound γ_i of one training step can be computed using the term $D_\alpha \left((1-q)\, p_0 + q p_1 \, \| p_0 \right)$. Based on the RDP composition theorem [49], we compose RDP costs of multiple training steps through $\gamma = \sum_i \gamma_i$. We convert the RDP privacy cost (α, γ) to the $(\epsilon, \delta)-$DP privacy cost as follows.

Theorem 2 (From (α, γ)-RDP to (ϵ, δ)-DP [49]): If $\mathcal{A}$ is an (α, γ)-RDP mechanism, it also satisfies (ϵ, δ)-DP, for any $0 < \delta < 1$, where $\epsilon = \gamma + \frac{\log 1/\delta}{\alpha - 1}$.

Therefore, we can use different Gaussian noise variance σ^2 to calculate the final privacy cost $\gamma + \frac{\log 1/\delta}{\alpha-1}$ until the required privacy budget ε is reached.

Appendix C
Theoretical Analysis of Curiosity-Driven Pre-training

During pre-training, we replace a proportion $p \in [0, 1]$ of real samples in each batch with synthetic samples from the current diffusion model distribution $q_\theta(x)$. However, directly using $q_\theta(x)$ would reinforce existing patterns, limiting diversity. To encourage exploration of under-represented regions, we reweight synthetic samples using an exponential scheme [82]:

$$\begin{aligned} \tilde{p}_{\text{synth}}(x) &= \frac{\exp(\beta c(x))\, q_\theta(x)}{Z_\beta}, \\ Z_\beta &= \int \exp(\beta c(x'))\, q_\theta(x')\, dx'. \end{aligned} \tag{2}$$

where $c(x)$ is the curiosity score (Equation (4)), and $\beta > 0$ controls the sharpness of curiosity emphasis. This reweighting strategy is motivated by the Importance Sampling View [83], and the term $\exp(\beta c(x))$ biases the distribution toward novel samples in low-density regions. In practice, we approximate $\tilde{p}_{\text{synth}}(x)$ by sampling a batch data from $q_\theta(x)$ and selecting the top-k samples with the highest curiosity scores $c(x)$. This avoids computing the intractable normalization constant Z_β while still emphasizing under-represented regions.

Let $p_{\text{real}}(x)$ denote the real data distribution. This batch replacement induces an effective training distribution:

$$p_{\text{eff}}(x) = (1-p)\, p_{\text{real}}(x) + p\, \tilde{p}_{\text{synth}}(x), \tag{3}$$

where $\tilde{p}_{\text{synth}}(x)$ is the curiosity-weighted synthetic distribution. From a score-matching perspective, diffusion training minimizes,

$$\mathcal{L}(\theta) = \mathbb{E}_{x \sim p_{\text{eff}},\, t} \left[\left\| e_\theta(x^t, t) - \varepsilon \right\|^2 \right], \tag{4}$$

where x^t is the noisy version of clean data x_0 at step t. Since p_{eff} upweights high-curiosity regions, gradient contributions from under-represented modes are amplified, guiding q_θ toward a higher-coverage distribution and enhancing diversity.

Appendix D
Investigated Tasks and the Dataset

We provide more details of datasets in Appendix D of [23]. We carry out experiments across five tasks in three domains (`Maze2D`, `MuJoCo`, and `Kitchen`) sourced from D4RL [47], a benchmark recently introduced and widely studied for evaluating offline RL algorithms.

In DP trajectory synthesis, datasets often contain a limited number of trajectories, resulting in substantial DP noise during privatization, which can degrade data utility. To address this, we augment the dataset by segmenting long trajectories into shorter fragments. This approach increases the effective number of data points, diluting the relative impact of DP noise while preserving temporal dependencies within fragments. In real-world applications, where trajectory datasets may be larger, this method remains effective, as it adapts to varying dataset sizes, simulating realistic scenarios.

As presented in Section II-A, when calculating the dimensions of transitions, we consider a transition that consists of two states: an action and a reward. Besides, we present

TABLE IX: Default DP-SGD hyper-parameters under $\epsilon = 10$ and $\delta = 1 \times 10^{-6}$. Sampling rate q is set depending on the dataset size; clipping norm C is set to 1. We use the Adam optimizer with a learning rate of 3×10^{-4}, while all other hyper-parameters adhered to the default settings in [9].

Method	Parameter	Maze2D			Kitchen	Mujoco
		umaze	medium	large	partial	halfcheetah
PrivORL-n	sampling ratio q	1.28×10^{-4}	0.64×10^{-4}	0.32×10^{-4}	9.35×10^{-4}	12.67×10^{-4}
	training steps	574K	294K	433K	15K	240K
	noise multiplier σ	0.44	0.39	0.37	0.47	0.68
	batch size	128	128	128	128	128
PrivORL-j	sampling ratio q	5.18×10^{-2}	2.57×10^{-2}	1.29×10^{-2}	1.70×10^{-1}	2.53×10^{-1}
	training steps	200K	200K	200K	200K	200K
	noise multiplier σ	12.1	6.3	3.2	40.4	60.9
	batch size	1024	1024	1024	1024	1024

the division of pre-training and sensitive datasets in our experiments in Table VIII.

APPENDIX E
BASELINE IMPLEMENTATION

These baselines achieve state-of-the-art performance in the 'SynMeter' library [57]. For further information, we strongly encourage readers to refer to the 'SynMeter' [57]. We provide more details of baseline in Appendix E of our full paper [23].

APPENDIX F
HYPER-PARAMETER SETTINGS

Following prior implementations [9], [10], we use the Elucidated Diffusion Model (EDM) [43] for transition synthesis and the Diffusion Transformer [27] for trajectory synthesis.

A. Diffusion Model

Denoising Network. We use the Elucidated Diffusion Model (EDM) formulation of the denoising network [43]. The denoising network D_θ is parameterized as a Multi-Layer Perceptron (MLP) with skip connections from the previous layer, following the architecture described in [84]. We encode the noise level of the diffusion process using a Random Fourier Feature embedding, with dimensions of 16. The base network size adopts a width of 1024 and a depth of 6, resulting in approximately 6 million parameters.

Sampling of Diffusion Model. For the diffusion sampling process, we utilize the stochastic SDE sampler introduced by [43], using the default hyper-parameters designed for ImageNet. To enhance sample fidelity, we select a higher number of diffusion timesteps at 128. Our implementation is sourced from the repository.[2]

DP-SGD Training. We refer to the official repository Opacus[3] to implement DP-SGD. The hyper-parameters are detailed in Table IX. We calculate the sampling ratio q by dividing the batch size by the number of transitions within the dataset. For all datasets, we set the fine-tuning epoch to 5, and the training step is obtained by dividing the epoch by the sampling ratio. Consequently, the noise multiplier must be adjusted based on the varying number of training steps across different datasets to ensure uniform privacy protection across various training settings. The noise multiplier is calculated using the standard privacy analysis function of Opacus, and the max-grad-norm C is set to 1.0, following the default setting in Opacus. We use the same setting in all experiments for the max-grad-norm C, batch size, optimizer, and learning rate. We preprocess the dataset using splitting and random sampling techniques to simulate real-world scenarios, as detailed in Appendix D.

Transformer. For PrivORL-j, we represent the noise prediction network as the transformer-based architecture proposed by [10], which adopts a GPT2-like model. The transformer is configured as six hidden layers and four attention heads. We use T = 200 for diffusion steps, while all other hyper-parameters adhered to their default settings.

B. Curiosity Module

The curiosity module enhances the diversity of synthetic data during the pre-training phase. Both the target and prediction networks in our architecture are structured as MLPs. The target network is characterized by greater depth and higher-dimensional modeling capabilities. Throughout each epoch of the pre-training, we first generate a predefined number of synthetic transitions using the current diffusion model. Based on a designated curiosity rate, we identify and select those samples from the generated data with the highest curiosity scores. These selected samples are then replaced with the current batch of training samples to update the synthesizer. We apply the same settings across various training tasks unless differences are specifically noted. The output dimension of the target network is 32. The default curiosity rate is 0.3.

APPENDIX G
EFFICIENCY

We highlight the efficiency of PrivORL-n and PrivORL-j in the full version of our paper [23].

APPENDIX H
ADDITIONAL DISCUSSIONS

A. Synthesis Principles

To evaluate dataset synthesis in offline RL, drawing from other fields [7], [14], [16], we analyze from three perspectives: (1) utility, and (2) fidelity.

[2]https://github.com/lucidrains/denoising-diffusion-pytorch
[3]https://github.com/pytorch/opacus/

TABLE X: The average normalized returns of agents trained on synthetic datasets ($\epsilon = 10$) under two privacy accounting methods (RDP and PRV), evaluated with IQL and TD3PlusBC algorithms. The values in parentheses are the relative changes.

Domains	Datasets	IQL				TD3PlusBC			
		PrivORL-n		PrivORL-j		PrivORL-n		PrivORL-j	
		RDP	PRV	RDP	PRV	RDP	PRV	RDP	PRV
Maze2D	umaze	70.3 ± 2.1	72.5 ± 3.6 (↑2.2)	49.8 ± 6.8	55.5 ± 7.8 (↑5.7)	60.3 ± 6.3	63.1 ± 5.8 (↑2.8)	49.9 ± 4.9	52.9 ± 5.1 (↑3.0)
	medium	90.7 ± 8.6	92.1 ± 4.7 (↑1.4)	49.3 ± 1.7	41.4 ± 6.8 (↓7.9)	50.4 ± 6.4	51.3 ± 6.6 (↑0.9)	38.0 ± 1.6	38.9 ± 1.9 (↑0.9)
	large	81.0 ± 11.8	82.0 ± 4.6 (↑1.0)	37.7 ± 7.0	40.5 ± 8.1 (↑2.8)	75.3 ± 13.2	72.1 ± 6.3 (↓2.8)	35.6 ± 2.6	40.9 ± 3.6 (↑5.3)
Kitchen	partial	25.5 ± 2.5	24.0 ± 1.5 (↓1.5)	13.8 ± 7.5	15.0 ± 6.5 (↑1.2)	11.5 ± 0.0	13.0 ± 1.5 (↑1.5)	8.3 ± 2.5	8.0 ± 2.5 (↓0.3)
Average		66.9 ± 6.3	67.7 ± 3.6 (↑0.6)	41.9 ± 5.7	42.4 ± 7.3 (↑0.5)	49.4 ± 6.5	49.9 ± 5.1 (↑0.5)	33.0 ± 2.9	35.2 ± 3.3 (↑2.2)

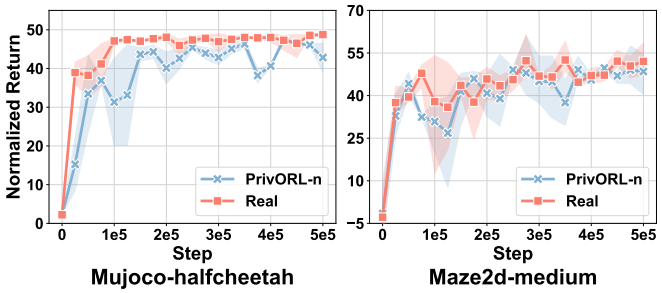


Fig. 7: Training curves of agents on real sensitive datasets and synthetic datasets ($\epsilon = 10$) using the TD3PlusBC algorithm.

Utility: A crucial aspect of dataset synthesis is that the synthetic dataset should be capable of accomplishing the downstream task. We hope that training agents with the synthesized dataset can achieve performance comparable to that of agents trained directly on the original dataset.

Fidelity: This principle ensures that the synthetic dataset has statistical characteristics similar to those of the raw dataset.

B. Evaluation Metrics

We introduce more details of evaluation metrics in Appendix H-C of our full version paper [23].

C. Limitations

We discuss the limitations of our work as follows.

- This work relies on public datasets for pre-training synthesizers. Without access to a suitable public dataset, PrivORL may reduce the performance of generating complex datasets.
- Optimal curiosity rates vary across different sensitive datasets. Section IX-E introduces that tuning the curiosity rate introduces additional privacy budget. A potential solution is incorporating DP hyper-parameter tuning [85], which provides a formal framework for selecting hyper-parameters under DP. This approach typically leverages techniques such as privacy-preserving grid search or adaptive tuning under a fixed privacy budget. Although DP hyper-parameter tuning introduces additional complexity, it offers a principled way to balance utility and privacy during parameter selection.
- DP-SGD performs well with large datasets, but when the dataset is small, the added noise is significant, leading to poor synthetic quality.
- Besides, our method faces challenges with high-dimensional datasets. The curse of dimensionality amplifies the noise introduced by DP, making it harder to preserve utility while ensuring privacy. High-dimensional feature spaces often require larger models and more iterations, which further increase computational costs and exacerbate the trade-off between privacy and data fidelity.

Future work aims to develop high-quality datasets without relying on public datasets and explore methods that can adaptively search for the optimal curiosity rate across different tasks under the DP constraint.

APPENDIX I
ADDITIONAL EXPERIMENTAL RESULTS

A. Privacy Accounting Using PRV

In this paper, we use RDP for fair comparisons with baselines, as RDP is widely adopted for privacy accounting in DP machine learning. We also investigate the use of an alternative privacy accounting method, Privacy Random Variable (PRV) [50], which provides a tighter analysis of privacy loss compared to RDP. Since PRV and RDP share similar analytical frameworks (both supporting moment accounting and composition across multiple training steps), the integration of PRV into our approach is seamless. Specifically, we replace the original RDP accountant in PrivORL with the PRV accountant implemented in the Opacus library [62].

Table X presents the average normalized returns of agents ($\epsilon = 10$) under two privacy accounting methods (RDP and PRV), evaluated with IQL and TD3PlusBC algorithms. In this table, we observe that although PRV provides a tighter analysis of privacy loss compared to RDP, the PRV's benefit is marginal in most cases. In particular, for PrivORL-n and PrivORL-j, using PRV results in only slight improvements: an average of 0.6 and 0.5 for IQL, and 0.5 and 2.2 for TD3PlusBC.

B. Training Convergence

We analyze the training curves of agents on real sensitive and synthetic datasets under $\epsilon = 10$ using the TD3PlusBC algorithm (Figure 7). In `Mujoco-halfcheetah`, agents trained on synthetic data exhibit slightly slower convergence, although their peak return remains comparable to that of real data. In `Maze2d-medium`, convergence behavior is largely similar across real and synthetic datasets, indicating that synthetic data can support effective agent learning.

APPENDIX J
MORE RELATED WORKS

We provide detailed related works discussions in Appendix J of our paper [23].

Artifact Appendix

Abstract. This artifact contains the implementations for the paper "PrivORL: Differentially Private Synthetic Dataset for Offline Reinforcement Learning." PrivORL uses a diffusion model and diffusion transformer to generate transitions and trajectories under DP, enabling secure release of synthetic datasets for offline RL research. The approach pre-trains a synthesizer on public datasets and fine-tunes it on sensitive data using DP-SGD. To enhance diversity, PrivORL introduces curiosity-driven pre-training, leveraging feedback from a curiosity module to produce realistic and varied synthetic samples. Our artifacts support this paper through: (1) *DP-Enabled Data Synthesizer.* Diffusion-based models trained with DP-SGD for privacy guarantees. (2) *Curiosity-Driven Pre-training.* A strategy to improve diversity and coverage of synthetic data. *Evaluation Suite.* Scripts for assessing utility, fidelity, and privacy to ensure reproducibility.

A. Description & Requirements

1) *How to access*: Our code can be pulled from the public repository on GitHub. Please refer to the link: https://github.com/2019ChenGong/PrivORL.

2) *Hardware dependencies*: PrivORL requires at least one NVIDIA A6000 GPU with 48 GB of memory. The Python version is Python 3.9.18.

3) *Consumption and running time*: Table I details the GPU memory consumption and runtime of PrivORL-n, PrivORL-j, and baselines.

4) *Software dependencies*: Our artifact requires the installation of `MUJOCO`[1], `CoRL`[2], `D4RL`[3], `SynMeter`[4], `Opacus`[5], and `MTDiff`[6].

B. Artifact Installation & Configuration

For installation, please refer to the link: https://github.com/2019ChenGong/PrivORL.

We provide detailed instructions for each step in the "README.md" file. All offline RL agents are trained on a server running Python 3.9.18. Additionally, we include a step-by-step guide for installing our repository using both Anaconda and Docker images. We conduct experiments on five offline datasets, "maze2d-umaze-dense-v1", "maze2d-medium-dense-v1", "maze2d-large-dense-v1", "kitchen-partial-v0", and "halfcheetah-medium-replay-v2". This repository can be easily extended to other datasets in `D4RL`. For evaluating the utility of synthetic datasets, we leverage three advanced offline RL algorithms, EDAC, IQL, and TD3PLUSBC from the `CoRL`[2] repository. Besides, users can also utilize other algorithms implemented in `CoRL`[2].

More details of codes and scripts for replicating our experiments can be found in "README.md".

[1]https://github.com/google-deepmind/mujoco/releases/tag/2.1.0
[2]https://github.com/tinkoff-ai/CORL
[3]https://github.com/Farama-Foundation/D4RL
[4]https://github.com/zealscott/SynMeter
[5]https://opacus.ai/
[6]https://github.com/tinnerhrhe/MTDiff

C. Major Claims

We list the major claims made in the paper, which can be reproduced and demonstrated in this artifact appendix.

- As presented in Table I on the utility of data synthesis, agents trained in synthetic transitions using PrivORL-n achieve the highest normalized returns in all tasks.
- In Table III on the fidelity of data synthesis, it can be observed that PrivORL-n consistently outperforms the baselines in both marginal and correlation statistics.
- As presented in Table XII on the defending against MIAs, PrivORL-n can synthesize transitions without privacy leakage of real sensitive transitions.

D. Experiment Workflow

This paper includes PrivORL-n and PrivORL-j for DP offline RL transition and trajectory synthesis. Our artifact supports them respectively. We provide an elaboration of hyperparameter settings in our repository.

After installation, we describe the process for obtaining the experimental results in our paper as quick start and detailed scripts.

Considering the time constraints of the artifact evaluation, we focus validation on the main results reported in Table I, Table III and Table XII of the paper, which represent the core claims of our work: (i) the utility and (ii) fidelity of synthetic data, (iii) defending against.

To reproduce these results, please run the following three scripts in order:

- "bash scripts/Table-I.sh" trains PrivORL-n on maze2d-medium-dense-v1, and evaluate the utility of synthetic data using IQL algorithm.
- "bash scripts/Table-III.sh" also contains training and evaluating the fidelity of synthetic data.
- "bash scripts/Table-XII.sh" trains PrivORL-n under non-DP, DP 1, and DP 10 settings, and evaluate the ability of defending against MIAs.

We provide the detailed scripts as follows:

- For DP transition-level synthesis, the first step is to pretrain the DP synthesizers in pretraining datasets, "python synther/training/train_diffuser.py –dataset ⟨the-name-of-dataset⟩ –datasets_name ⟨the-pretraining-dataset⟩ –curiosity_driven –curiosity_driven_rate 0.3 –results_folder ⟨the-target-folder⟩ –save_file_name ⟨store_path⟩ ", an instance of setting maze2d-medium-dense-v1 as the sensitive dataset is "python synther/training/train_diffuser.py –dataset maze2d-medium-dense-v1 –datasets_name "['maze2d-open-dense-v0', 'maze2d-umaze-dense-v1', 'maze2d-large-dense-v1']" –curiosity_driven –curiosity_driven_rate 0.3 –results_folder ./results_maze2d-medium-dense-v1_0.3". Then, we should fine-tune the pretrained synthesizer on the sensitive datasets using DP-SGD, "python synther/training/train_diffuser.py –dataset ⟨the-name-of-dataset⟩ –dp_epsilon 10 –results_folder ⟨the-target-folder⟩ –save_file_name ⟨store_path⟩ –load_path ⟨the-path-of-saved-pretraining-model⟩", a corresponding

TABLE I: GPU memory consumption, runtime, and normalized return: comparing PrivORL-n and PrivORL-j with baselines on synthetic datasets (`Maze2D-medium`). 'h' means hours.

Evaluation Metrics		PGM	PrivSyn	PrePATE-GAN	PrivORL-n	PrivORL-j-U	DP-Transformer	PrivORL-j
Memory	Pre-train	-	-	1.5GB	0.9GB	7.45GB	19.7GB	13.12GB
	Fine-tune	-	-	0.4GB	21.4GB	14.59GB	43.11GB	39.72GB
	Synthesis	11.1GB	9.4GB	8.1GB	4.3GB	4.1GB	9.74GB	8.15GB
Time	Pre-train	-	-	12h	1.5h	0.45h	1.37h	0.77h
	Fine-tune	-	-	11h	2h	0.96h	5.18h	3.03h
	Synthesis	6h	8.5h	0.18h	0.5h	0.24h	10.1h	1.42h

instance is "python synther/training/train_diffuser.py –dataset maze2d-medium-dense-v1 –dp_epsilon 10 –results_folder ./results_maze2d-medium-dense-v1_0.3 –load_path ./results_maze2d-medium-dense-v1_0.3/pretraining-model-4.pt –save_file_name maze2d-medium-dense-v1_samples_1000000.0_10dp_0.8.npz –load_checkpoint".

- For results in Table I, which shows the downstream task agent training on the synthetic datasets, please run the code: "python evaluation/eval-agent/cql.py –env ⟨the-name-of-synthetic-transitions⟩ –checkpoints_path ⟨store_path⟩ –config ⟨the-path-of-configuration-file⟩ –dp_epsilon ⟨the-privacy-budget-of-synthetic-transitions⟩ –diffusion.path ⟨the-path-of-saved-transitions⟩ –name ⟨the-name-of-logging⟩ –prefix ⟨the-prefix-of-name⟩ –save_checkpoints ⟨whether-to-save-ckpt⟩". An example of training agent using iql on the synthetic maze2d-medium-dense-v1 dataset is "python evaluation/eval-agent/iql.py –env maze2d-medium-dense-v1 –checkpoints_path corl_logs_param_analysis_v1_maze2d/ –config synther/corl/yaml/iql/maze2d/medium-dense-v1.yaml –dp_epsilon 10 –diffusion.path ./results_maze2d-medium-dense-v1_0.3/cleaned_pretrain_samples.npz –name CurDPsynthER –prefix 0.3CurRate –save_checkpoints False".
- For results in Table III, which evaluates the fidelity of the synthetic datasets, please run the code: "python evaluation/eval-fidelity/marginal.py –dataset ⟨the-name-of-synthetic-transitions⟩ –dp_epsilon ⟨the-privacy-budget-of-synthetic-transitions⟩ –cur_rate ⟨the-curiosity-rate-of-synthetic-transitions⟩ –load_path ⟨the-path-of-saved-transitions⟩". An instance of computing the marginal and correlation values between the synthetic and real transitions of maze2d-medium-dense-v1 env is "python evaluation/eval-fidelity/marginal.py –dataset maze2d-medium-dense-v1 –dp_epsilon 10 –load_path alter_0.3curiosity_driven_results_maze2d-medium-dense-v1_1.0/cleaned_maze2d-medium-dense-v1_samples_1000000.0_10dp_0.8.npz –cur_rate 0.3".
- For results in Table XII, which evaluates the privacy protection of the synthetic datasets through MIA methods, please change the args nondp_weight, dp1_weight, and dp10_weight to the corresponding checkpoints and run the code: "python evaluation/eval-mia/mia.py". For example, run "python evaluation/eval-mia/mia.py –dataset maze2d-medium-dense-v1 –nondp_weight 1e3data-300epoch-finetuning-without-dp-model-299.pt –dp1_weight finetuning_dp1.0-model-4.pt –dp10_weight finetuning_dp10.0-model-4.pt –sigma [0.05, 0.01] –repeat 64 –sample_num 10000".

For more details on the code, we recommend referring to our GitHub repository. Offline RL agents often suffer from unstable performance. To mitigate environmental randomness and enhance construct validity, we train each agent using five different random seeds. Each agent interacts with the environment to generate 100 trajectories, and we report the average cumulative return across these trajectories as the performance metric. Consequently, some variation in results across experiments is expected, even under identical settings.

E. Copyright

We have migrated the code repository from GitHub to Zenodo. The DOI for the repository is "https://doi.org/10.5281/zenodo.17845537".

F. Acknowledgement

We thank the authors of `CoRL`[2] for providing the code for the offline RL algorithms and `D4RL`[3] for the offline datasets used in our evaluations. Besides, we thank the authors of `MUJOCO`[1], `SynMeter`[4], `Opacus`[5] and `MTDiff`[6] for our algorithmic implementation supports.