DIFFERENTIALLY PRIVATE SYNTHETIC DATA VIA FOUNDATION MODEL APIS 1: IMAGES

Zinan Lin
Microsoft Research
zinanlin@microsoft.com

Sivakanth Gopi
Microsoft Research
sigopi@microsoft.com

Janardhan Kulkarni
Microsoft Research
jakul@microsoft.com

Harsha Nori
Microsoft Research
hanori@microsoft.com

Sergey Yekhanin
Microsoft Research
yekhanin@microsoft.com

ABSTRACT

Generating *differentially private (DP) synthetic data* that closely resembles the original private data is a scalable way to mitigate privacy concerns in the current data-driven world. In contrast to current practices that train customized models for this task, we aim to *generate DP Synthetic Data via APIs (DPSDA)*, where we treat foundation models as blackboxes and only utilize their inference APIs. Such API-based, training-free approaches are easier to deploy as exemplified by the recent surge in the number of API-based apps. These approaches can also leverage the power of large foundation models which are only accessible via their inference APIs. However, this comes with greater challenges due to strictly more restrictive model access and the need to protect privacy from the API provider.

In this paper, we present a new framework called *Private Evolution (PE)* to solve this problem and show its initial promise on synthetic images. Surprisingly, PE can match or even outperform state-of-the-art (SOTA) methods *without any model training*. For example, on CIFAR10 (with ImageNet as the public data), we achieve FID$\leq$7.9 with privacy cost $\epsilon = 0.67$, significantly improving the previous SOTA from $\epsilon = 32$. We further demonstrate the promise of applying PE on large foundation models such as Stable Diffusion to tackle challenging private datasets with a small number of high-resolution images. The code and data are released at https://github.com/microsoft/DPSDA.

1 INTRODUCTION

While data-driven approaches have been successful, privacy is a major concern. For example, statistical queries of a dataset may leak sensitive information about individual users (Dwork et al., 2014). Entire training samples can be reconstructed from deep learning models (Haim et al., 2022; Fredrikson et al., 2015; Carlini et al., 2021a; 2023a;b; 2019; 2021b; Choquette-Choo et al., 2021; Tramèr et al., 2022; Wang et al., 2023). *Differential privacy (DP)* is the gold standard in quantifying and mitigating these concerns (Dwork et al., 2006). DP algorithms ensure that information about individual samples in the original data cannot be inferred with high confidence from algorithm outputs. *Differentially private synthetic data* is the holy grail of DP research (Hu et al., 2023; Jordon et al., 2019; Lin et al., 2020; Beaulieu-Jones et al., 2019; Dockhorn et al., 2022; Yin et al., 2022; Yu et al., 2021; He et al., 2022; Li et al., 2021; Ghalebikesabi et al., 2023; Yue et al., 2022; Harder et al., 2023; 2021; Savage, 2023; Lin, 2022; Tang et al., 2023). The goal is to generate a synthetic dataset that is statistically similar to the original data while ensuring DP. The benefits are: (1) Thanks to the post-processing property of DP (Dwork et al., 2014), we can use any existing *non-private* algorithm (e.g., training machine learning (ML) models) as-is on the synthetic data without incurring additional privacy loss. This is more scalable than redesigning and reimplementing every algorithm for DP. (2) Synthetic data can be shared freely with other parties without violating privacy. This is useful in situations when sharing data is necessary, such as when organizations (e.g., hospitals) want to release datasets to support open research

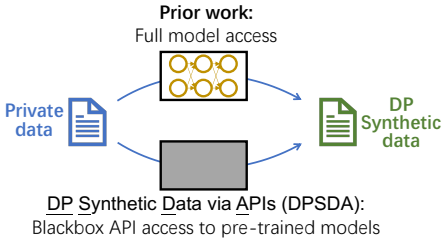


Figure 1: We consider the problem of generating DP synthetic data with API access to pre-trained models without any model training. This is in contrast to prior work which assumes full access to pre-trained models and requires training.

Figure 2: Private Evolution (PE) framework for DP synthetic data. **Left: Intuition of PE.** Though private data and pre-trained generative models have very different distributions, the support of the former is likely to be covered by the support of the latter. We gradually shift the distribution of generated data toward private data through PE. **Right: Algorithm of PE.** We maintain a sample set (population), and iteratively select the most similar ones to the private samples (parents) and mutate them to generate the next population (offspring). The initial population and offspring are generated with foundation model APIs. Parent selection is done in DP using private samples.

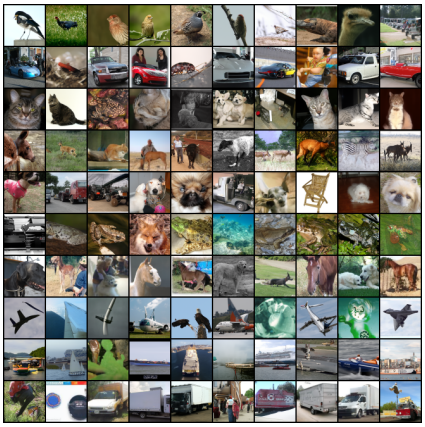


Figure 3: Generated samples on CIFAR10 with $(0.67, 10^{-5})$-DP. Each row corresponds to one class. FID=7.87. See App. I for real and generated images side-by-side.

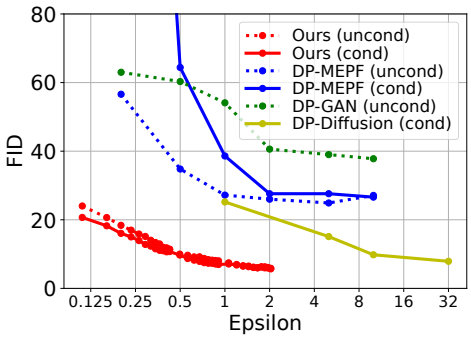


Figure 4: FID (Heusel et al., 2017) (lower is better) v.s. privacy cost ϵ on CIFAR10 ($\delta = 10^{-5}$). (Un)cond means (un)conditional generation. Ours achieves the best privacy-quality trade-off compared to DP-MEPF (Harder et al., 2023), DP-GAN, DP-Diffusion (Ghalebikesabi et al., 2023).

initiatives (Beaulieu-Jones et al., 2019; Lin et al., 2020). (3) Since synthetic data is DP, developers can look at the data directly, which makes algorithm development and debugging a lot easier.

At the same time, with the recent advancement of large foundation models, API-based solutions are gaining tremendous popularity, exemplified by the surge of GPT4-based applications. In contrast to the traditional paradigm that trains/fine-tunes customized ML models for each application, API-based solutions treat ML models as blackboxes and only utilize APIs[1] that provide the input/output functions of the models. In fact, many foundation models including GPT4, Bard, and DALLE2 only provide API access without releasing model weights or code. Key reasons for the success of API-based solutions are that APIs offer a clean abstraction of ML and are readily available and scalable. Therefore, implementing and deploying these API-based algorithms is easier and faster even for developers without ML expertise. Such an approach can also leverage powerful foundation models that are only accessible through APIs. Unfortunately, SOTA DP synthetic data algorithms today are still in the old paradigm (Ghalebikesabi et al., 2023; Li et al., 2021): they need a customized training process for each dataset, whose implementation requires significant ML engineering efforts (§ 3.1).

Motivated from these observations, we ask the following ambitious question (Fig. 1):

Can we generate DP synthetic data using blackbox APIs of foundation models?

We treat API providers as *untrusted* entities so we also want to protect user privacy from them, i.e., the API queries we make during generation should also be DP. If successful, we can potentially democratize the deployment of DP synthetic data in the industry similar to how API-based solutions have facilitated other applications. This is a challenging task, however, as we do not have access to model weights and gradients by assumption. In this paper, we conduct the first exploration of the potential and limits of this vision on DP synthetic *images*. Surprisingly, we show that not only is such a vision realizable, but that it also has the potential to match or improve SOTA training-based DP synthetic image algorithms despite more restrictive model access. Our contributions are:

[1]See https://platform.openai.com/docs/introduction for examples of APIs. For example, a text completion API can complete a text prompt using a foundation model such as GPT4. An image variation API can produce variations of a given image using a foundation model such as DALLE2.

(1) New problem (§ 3). We highlight the importance of *DP Synthetic Data via APIs (DPSDA)*. Such algorithms are easy to implement and deploy and can leverage the foundation models behind APIs.

(2) New framework (§ 4). We propose *Private Evolution (PE)* algorithm for achieving our goal (Fig. 2). We consider using 2 popular APIs: random generation and sample variation (i.e., generating a sample similar to the given one).[4,5] The key idea is to iteratively use private samples to vote for the most similar samples generated from the blackbox model and ask the blackbox models to generate more of those similar samples. We theoretically prove that the distribution of the generated samples from PE will converge to the private distribution under some modeling assumptions (App. D). PE only requires (existing) APIs of the foundation models, and does not need any model training.

(3) Experimental results (§ 5).[2] Some key results are: (a) Surprisingly, without any training, PE can still outperform SOTA training-based DP image generation approaches on some datasets (Figs. 3 and 4). For example, to obtain FID≤ 7.9 on CIFAR10 dataset, PE (with blackbox access to an ImageNet-pre-trained model) only needs $\epsilon = 0.67$. In contrast, DP fine-tuning of an ImageNet-pre-trained model (prior SOTA) requires $\epsilon = 32$ (Ghalebikesabi et al., 2023).
(b) We show that PE works even when there is significant distribution shift between private and public data. We create a DP synthetic version (with $\varepsilon = 7.58$) of Camelyon17, a medical dataset for classification of breast cancer metastases, using the same ImageNet-pre-trained model. A downstream classifier trained on our DP synthetic data achieves a classification accuracy of 79.56% (prior SOTA based on DP fine-tuning is 91.1% with $\epsilon = 10$ (Ghalebikesabi et al., 2023)).
(c) We set up new challenging benchmarks that the DP synthetic image literature has not studied before. We show that with powerful foundation models such as Stable Diffusion (Rombach et al., 2022), PE can work with high-resolution (512x512) image datasets with a small size (100 images), which are common in practice but challenging for current DP synthetic image algorithms.

2 Background and Related Work

Differential Privacy (DP). We say a mechanism $\mathcal{M}$ is (ϵ, δ)-DP if for any two neighboring datasets $\mathcal{D}$ and $\mathcal{D}'$ which differ in a single entry (i.e., $\mathcal{D}'$ has one extra entry compared to $\mathcal{D}$ or vice versa) and for any set S of outputs of $\mathcal{M}$, we have $\mathbb{P}\left(\mathcal{M}\left(\mathcal{D}\right) \in S\right) \leq e^{\epsilon}\mathbb{P}\left(\mathcal{M}\left(\mathcal{D}'\right) \in S\right) + \delta$. Intuitively, this means that any single sample cannot influence the mechanism output too much.

DP synthetic data. Given a private dataset $\mathcal{D}$, the goal is to generate a DP synthetic dataset $\mathcal{M}\left(\mathcal{D}\right)$ which is statistically similar to $\mathcal{D}$. One method is to *train generative models from scratch on private data* (Lin et al., 2020; Beaulieu-Jones et al., 2019; Dockhorn et al., 2022) with DP-SGD (Abadi et al., 2016), a DP variant of stochastic gradient descent. Later studies show that *pre-training generative models on public data* before fine-tuning them on private data with DP-SGD (Yin et al., 2022; Yu et al., 2021; He et al., 2022; Li et al., 2021; Ghalebikesabi et al., 2023; Yue et al., 2022) gives better privacy-utility trade-offs due to knowledge transfer from public data (Yin et al., 2022; Ganesh et al., 2023), smaller gradient spaces (Li et al., 2022), or better initialization (Ganesh et al., 2023). This approach achieves SOTA results on several data modalities such as text and images. In particular, DP-Diffusion (Ghalebikesabi et al., 2023) achieves SOTA results on DP synthetic images by pre-training diffusion models (Sohl-Dickstein et al., 2015; Ho et al., 2020) on public datasets and fine-tuning them on the private dataset. Some other methods do not depend on DP-SGD (Jordon et al., 2019; Harder et al., 2023; 2021; Vinaroz et al., 2022; Cao et al., 2021). For example, DP-MEPF (Harder et al., 2023) trains generative models to produce synthetic data that matches the (privatized) statistics of the private features.

Note that all these methods obtain generative models *whose weights are DP*, which can then be used to draw DP synthetic data. It is stronger than our goal which only requires DP synthetic data (Lin et al., 2021). In this paper, we do not do any model training and only produce DP synthetic data.

3 DP Synthetic Data via APIs (DPSDA)

3.1 Motivation

As discussed in § 2, SOTA DP synthetic data algorithms require training or fine-tuning generative models with DP-SGD. There are some obstacles to deploying them in practice.

(1) Significant engineering effort. Deploying normal ML training pipelines is hard; deploying *DP* training pipelines is even harder because most ML infrastructure is not built around this use case.

[2] In the experiments of this paper, we only experimented with APIs from local models where the user has full control of the model weights and runs it in a controlled environment.

Recently, there has been significant progress in making DP training more efficient (Li et al., 2021; He et al., 2022) and easy to use (Opacus (Yousefpour et al., 2021) and Tensorflow Privacy). However, incorporating them in new codebases and new models is highly non-trivial. For example, Opacus requires us to implement our own per-sample gradient calculator for new layers. Common layers and loss functions that depend on multiple samples (e.g., batch normalization) are often not supported.

(2) Inapplicability of API-only models. It may be appealing to take advantage of the powerful foundation models in DP synthetic data generation. However, due to the high commercial value of foundation models, many companies choose to only release inference APIs of the models but not the weights or code. Examples include popular models such as DALLE 2 (Ramesh et al., 2022) and GPT 3/4 (Brown et al., 2020; OpenAI, 2023) from OpenAI and Bard from Google. In such cases, existing training-based approaches are not applicable.[3]

In contrast, DP synthetic data approaches that only require model inference APIs could potentially be deployed more easily, as they do not require ML or DP expertise to conduct modifications inside the model and require minimal modifications when switching to a different model (as long as they support the same APIs). In addition, such an approach is compatible with both the models running locally and the models behind APIs.

3.2 Problem Formulation

We now give a formal statement of DPSDA. We first define a core primitive for DP synthetic data.

DP Wasserstein Approximation (DPWA). Given a private dataset $S_{\text{priv}} = \{x_i : i \in [N_{\text{priv}}]\}$ with N_{priv} samples (e.g., images), a distance function $d(\cdot, \cdot)$ between samples and some $p \geq 1$, the goal is to design an (ϵ, δ)-DP algorithm $\mathcal{M}$ that outputs a synthetic dataset $S_{\text{syn}} = \{x'_i : i \in [N_{\text{syn}}]\}$ with N_{syn} samples (as a multiset) whose distance to S_{priv}, $W_p(S_{\text{priv}}, S_{\text{syn}})$, is minimized. Here W_p is the Wasserstein p-distance w.r.t. the distance function $d(\cdot, \cdot)$ (see App. A for the definition).

DPSDA. We want to solve DPWA where $\mathcal{M}$ is given blackbox access to foundation models trained on public data via APIs.[1] API queries should also be (ϵ, δ)-DP as API providers cannot be trusted.

In some applications, besides the raw samples x_i, we may also care about some auxiliary information such as class labels of images. In such cases, we may write $S_{\text{priv}} = \{(x_i, y_i) : i \in [N_{\text{priv}}]\}$ (and $S_{\text{syn}} = \{(x'_i, y'_i) : i \in [N_{\text{syn}}]\}$) where y_i (and y'_i) is the auxiliary information of i-th sample.

When the distance function $d(\cdot, \cdot)$ is ℓ_2 (in the sample space), DPWA is closely related to DP Clustering (Ghazi et al., 2020; Su et al., 2016; Balcan et al., 2017) and DP Heatmaps (Ghazi et al., 2022). But a direct application of them does not work in our setting; see App. G for more discussions.

3.3 Scope of This Work

Data type. In this paper, we instantiate the above framework on *images*. We consider both unconditional (i.e., no y_i) and conditional generation (e.g., y_i can be image categories such as cats/dogs).

APIs. In our algorithm design and experiments, we use 2 APIs, both of which are either directly provided in the APIs of popular models (e.g., DALLE 2,[4] Stable Diffusion[5]) or can be easily implemented by adapting current APIs (e.g., using appropriate text prompts in GPT APIs[1]):

(1) RANDOM_API (n) that randomly generates n samples. Some APIs accept condition information such as text prompts in text-to-image generation,[4,5] which we omit in the notation for simplicity.

(2) VARIATION_API (S) that generates variations for each sample in S. For images, it means to generate similar images to the given one, e.g., with similar colors or objects.[4,5] Some APIs also support setting the variation degree: VARIATION_API (S, v), where larger v indicates more variation.[6]

4 Private Evolution (PE)

Foundation models have a broad and general model of our world from their extensive training data. Therefore, we expect that foundation models can generate samples close to private data with non-

[3]Some companies also provide model fine-tuning APIs, e.g., https://platform.openai.com/docs/guides/fine-tuning. However, they do not support DP fine-tuning and do not provide gradients. Also, uploading sensitive data to these APIs controlled by other companies can lead to privacy violations.

[4]See https://platform.openai.com/docs/guides/images/usage.

[5]See https://huggingface.co/docs/diffusers/api/pipelines/stable_diffusion/overview.

[6]If this is not implemented, we can simply compose the VARIATION_API v times to achieve it.

negligible probability. The challenge is that by naively calling the APIs, the probability of drawing such samples is quite low. We need a way to *guide* the generation towards private samples.

Inspired by *evolutionary algorithms (EA)* (Davis, 1987) (App. B), we propose *Private Evolution (PE)* framework for generating DP synthetic data via APIs. See Fig. 2 for the intuition behind PE. The complete algorithm is in Alg. 1. Below, we discuss the components in detail.

Algorithm 1: Private Evolution (PE)

Input : Private samples: $S_{\text{priv}} = \{x_i\}_{i=1}^{N_{\text{priv}}}$
Number of iterations: T
Number of generated samples: N_{syn}
Noise multiplier for DP Nearest Neighbors Histogram: σ
Threshold for DP Nearest Neighbors Histogram: H
Output: Synthetic data: S_{syn}

1 $S_1 \leftarrow$ RANDOM_API (N_{syn})
2 **for** $t \leftarrow 1, \ldots, T$ **do**
3 $\quad histogram_t \leftarrow$ DP_NN_HISTOGRAM $(S_{\text{priv}}, S_t, \sigma, H)$ `// See Alg. 2`
4 $\quad \mathcal{P}_t \leftarrow histogram_t / \text{sum}(histogram_t)$ `//` $\mathcal{P}_t$ `is a distribution on` S_t
5 $\quad S'_t \leftarrow$ draw N_{syn} samples with replacement from $\mathcal{P}_t$ `//` S'_t `is a multiset`
6 $\quad S_{t+1} \leftarrow$ VARIATION_API (S'_t)
7 **return** S_T

Algorithm 2: DP Nearest Neighbors Histogram (DP_NN_HISTOGRAM)

Input : Private samples: S_{priv}
Generated samples: $S = \{z_i\}_{i=1}^{n}$
Noise multiplier: σ
Threshold: H
Distance function: $d(\cdot, \cdot)$
Output: DP nearest neighbors histogram on S

1 $histogram \leftarrow [0, \ldots, 0]$
2 **for** $x_{\text{priv}} \in S_{\text{priv}}$ **do**
3 $\quad i = \arg\min_{j \in [n]} d(x_{\text{priv}}, z_j)$
4 $\quad histogram[i] \leftarrow histogram[i] + 1$
5 $histogram \leftarrow histogram + \mathcal{N}(0, \sigma I_n)$ `// Add noise to ensure DP`
6 $histogram \leftarrow \max(histogram - H, 0)$ `// 'max', '-' are element-wise`
7 **return** $histogram$

Initial population (Line 1). We use RANDOM_API to generate the initial population. If there is public information about the private samples (e.g., they are dog images), we can use this information as prompts to the API to seed a better initialization.

Fitness function (Line 3 or Alg. 2). We need to evaluate how useful each sample in the population is for modeling the private distribution. Our idea is that, if a sample in the population is surrounded by many private samples, then we should give it a high score. To implement this, we define the fitness function of a sample x as *the number of private samples whose nearest neighbor in the population is* x. A higher fitness value means that more private samples are closest to it. More details are below:

(1) Distance function (Line 3, Alg. 2). To define "nearest neighbor", we need a distance function that measures the similarity of two samples. A naive way is to use ℓ_2 distance $d(x, z) = \|x - z\|_2$, where x is from the private dataset and z is from the population. However, it is well-known that ℓ_2 distance on pixel space is not a good metric for images. For example, a small shift of an object can result in a high ℓ_2 distance. We therefore compute the ℓ_2 distance in the embedding space:

$$d(x, z) = \|\Phi(x) - \Phi(z)\|_2 \tag{1}$$

where Φ is a network for extracting image embeddings such as inception embedding (Szegedy et al., 2016) or CLIP embedding (Radford et al., 2021).

(2) Lookahead. The above approach gives high scores to the good samples in the *current* population. However, as we will see later, these good samples will be modified through VARIATION_API for the next population. Therefore, it is better to "look ahead" to compute the distance based on the modified samples as if they are kept in the population. We modify Eq. (1) to

compute the distance between the embedding of x and the *mean embedding of k variations of z*: $d(x, z) = \left\| \Phi(x) - \frac{1}{k} \sum_{i=1}^{k} \Phi(z^i) \right\|_2$, where k is called *lookahead degree*, and $z^1, \ldots, z^k$ are variations of z obtained via VARIATION_API.

(3) Noise for DP (Line 5, Alg. 2)). Because this step utilizes private samples, we need to add noise to ensure DP. We add i.i.d. Gaussian noise from $\mathcal{N}(0, \sigma)$. The privacy analysis is presented in § 4.3.

(4) Thresholding (Line 6, Alg. 2)). When the number of generated samples is large, the majority of the histogram will be DP noise added above. To make the signal-noise ratio larger, we set a threshold H to each bin of the histogram. Similar ideas have been used in DP set union (Gopi et al., 2020).

In summary, we called the above fitness function *DP Nearest Neighbors Histogram*. Note that it is not the traditional "histogram" on a continuous space that requires binning. Instead, it is a histogram built on the generated samples: the value of i-th bin means the (privatized) number of private samples whose nearest neighbor among the generated samples is the i-th sample.

Parent selection (Line 5). We sample from the population according to the DP Nearest Neighbors Histogram so that a sample with more private samples around is more likely to be selected.

Offspring generation (Line 6). We use VARIATION_API to get variants of the parents as offsprings.

Please see App. D for the convergence analysis of PE.

4.1 Conditional Generation

The above procedure is for unconditional generation. To support conditional generation, i.e., each generated sample is associated with a label such as an image class (e.g., cats v.s. dogs), we take a simple approach: *we repeat the above process for each class of samples in the private dataset separately*. See Alg. 3 for the full algorithm.

4.2 Generating Unlimited Number of Samples

Our algorithm in Alg. 1 is preset with a fixed number of generated samples N_{syn}. What if users want more samples afterward? In prior training-based methods (Ghalebikesabi et al., 2023), this is easy to achieve: one can draw an arbitrary number of samples from the trained generative models without additional privacy cost. In this section, we want to highlight that PE can also do that, again *with API access only*. We can simply generate an unlimited number of samples by calling variation API multiple times, each with the generated dataset as input: $[\text{VARIATION_API}(S_{\text{syn}}), \ldots, \text{VARIATION_API}(S_{\text{syn}})]$. In 5.1.3, we will see that this simple algorithm is sufficient to provide more useful samples for downstream applications.

4.3 Privacy Analysis

Unlike the analysis DP-SGD which requires complicated DP composition theorems (e.g., Gopi et al. (2021); Mironov (2017)) due to subsampling, PE does not have subsampling steps and therefore the privacy analysis is rather straightforward. The DP guarantee of the unconditional version of PE (Alg. 1) can be reasoned as follows:

- **Step 1: The sensitivity of DP Nearest Neighbors Histogram (Lines 1 to 4 in Alg. 2).** Each private sample only contributes one vote. If we add or remove one sample, the resulting histogram will change by 1 in the ℓ_2 norm. Therefore, the sensitivity is 1.
- **Step 2: Regarding each PE iteration as a Gaussian mechanism.** Line 5 adds i.i.d. Gaussian noise with standard deviation σ to each bin. This is a standard Gaussian mechanism (Dwork et al., 2014) with noise multiplier σ.
- **Step 3: Regarding the entire PE algorithm as T adaptive compositions of Gaussian mechanisms**, as PE is simply applying Alg. 2 T times sequentially.
- **Step 4: Regarding the entire PE algorithm as one Gaussian mechanism with noise multiplier $\sigma/\sqrt{T}$.** It is a standard result from Dong et al. (2022) (see Corollary 3.3 therein).
- **Step 5: Computing DP parameters ϵ and δ.** Since the problem is simply computing ϵ and δ for a standard Gaussian mechanism, we use the formula from Balle & Wang (2018) directly.

For the conditional version of PE (Alg. 3), since it does the unconditional version of PE for each class separately (discussed in § 4.1), adding or removing one sample will only influence the generation results of one class. For that class, the impact due to the added/removed sample is also bounded, as seen in the privacy analysis above. Therefore, Alg. 3 is also DP. In fact, we can show that *the privacy guarantee of Alg. 3 is the same as Alg. 1*. Please refer to App. C for more details.

This privacy analysis implies that releasing all the (intermediate) generated sets $S_1, \ldots, S_T$ also satisfies the same DP guarantees. Therefore PE provides the same privacy even from the API provider.

5 EXPERIMENTS

In § 5.1, we compare PE with SOTA training-based methods on standard benchmarks to understand its promise and limitations. In § 5.2, we present proof-of-concept experiments to show how PE can utilize the power of large foundation models. We did (limited) hyper-parameter tunings in the above experiments; following prior DP synthetic data work (Yu et al., 2021; Ghalebikesabi et al., 2023), we ignore the privacy cost of hyper-parameter tuning. However, as we will see in the ablation studies (§ 5.3 and App. M), PE stably outperforms SOTA across a wide range of hyper-parameters, and the results can be further improved with better hyper-parameters than what we used. Detailed hyper-parameter settings and more results such as *generated samples* and *their nearest images in the private dataset* are in Apps. I to K.

5.1 COMPARISONS TO STATE-OF-THE-ART

Public information. We use standard benchmarks (Ghalebikesabi et al., 2023) which treat ImageNet (Deng et al., 2009) as public data. For fair comparisons, we only use ImageNet as public information in PE: *(1) Pre-trained model.* Unlike the SOTA (Ghalebikesabi et al., 2023) which trains customized diffusion models, we simply use public ImageNet pre-trained diffusion models (pure image models without text prompts) (Nichol & Dhariwal, 2021). RANDOM_API and VARIATION_API are implemented using the same pre-trained model (see App. I). *(2) Embedding (Eq. (1)).* We use ImageNet inception embedding (Szegedy et al., 2016). PE is not sensitive to embedding choice though and we get good results even with CLIP embeddings (Fig. 41 in App. M).

Baselines. We compare with DP-Diffusion (Ghalebikesabi et al., 2023), DP-MEPF (Harder et al., 2023), and DP-GAN (Harder et al., 2023; Goodfellow et al., 2020). DP-Diffusion (Ghalebikesabi et al., 2023) is the current SOTA that achieves the best results on these benchmarks. Baseline results are taken from their paper.

Outline. We test PE on private datasets that are either similar to or differ a lot from ImageNet in § 5.1.1 and 5.1.2. We demonstrate that PE can generate an unlimited number of useful samples in § 5.1.3. We show that PE is computationally cheaper than train-based methods in App. O.

5.1.1 MODERATE DISTRIBUTION SHIFT (IMAGENET → CIFAR10)

We treat CIFAR10 (Krizhevsky et al., 2009) as private data. Given that both ImageNet and CIFAR10 are natural images, it is a relatively easy task for PE (and also for the baselines). Figs. 3 to 5 show the results. Surprisingly, despite the fact that we consider strictly more restrictive model access and do not need training, PE still outperforms the SOTA training-based methods. Details are below.

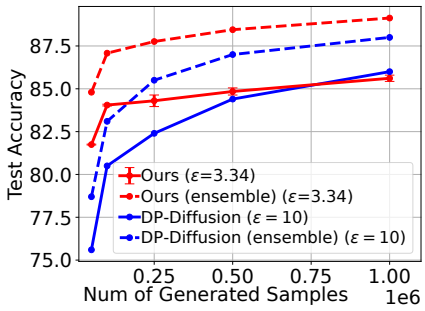


Figure 5: Downstream classification accuracy (higher is better) on CIFAR10 ($\delta = 10^{-5}$). The baseline results are taken from Ghalebikesabi et al. (2023). Two "ensemble" lines are from ensembles of 5 classifiers. The other two lines show the average accuracy of 5 independently trained classifiers with error bars. Our PE achieves better accuracy across almost all settings with smaller privacy costs.

Sample quality. Fig. 4 shows the trade-off between privacy cost and FID, a popular metric for image quality (Heusel et al., 2017). For either conditional or unconditional generation, PE outperforms the baselines significantly. For example, to reach FID$\leq$ 7.9, Ghalebikesabi et al. requires $\epsilon = 32$, Harder et al. cannot achieve it even with infinity ϵ, whereas our PE only needs $\epsilon = 0.67$.

Downstream classification accuracy. We train a downstream WRN-40-4 classifier (Zagoruyko & Komodakis, 2016) from scratch on 50000 generated samples and test the accuracy on CIFAR10 test set. This simulates how users would use synthetic data, and a higher accuracy means better utility. Fig. 5 shows the results (focus on the left-most points with num of generated samples = 50000 for now). Harder et al. (2023) achieves 51% accuracy with $\epsilon = 10$ (not shown). Compared with the SOTA (Ghalebikesabi et al., 2023), PE achieves better accuracy (+6.1%) with less privacy cost. Further with an ensemble of 5 classifiers trained on the same data, PE is able to reach an accuracy of 84.8%. For reference, the SOTA DP classifier pre-trained on ImageNet (without DP) and fine-tuned on CIFAR10 (with DP) (De et al., 2022) achieves 94.8% and 95.4% accuracies with epsilon=1 and

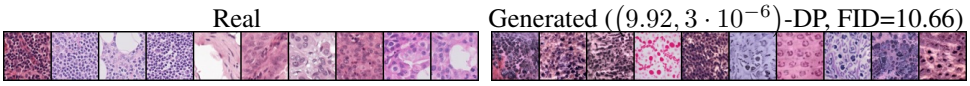


Figure 6: Real and generated images from Camelyon17. More in App. J.

2 respectively. It is not surprising that DP classifiers outperform PE (and other DP synthetic data approaches) on classification tasks, as DP classifiers are targeted at and optimized for a single task whereas DP synthetic data is general-purpose.

The above results suggest that when private and public images are similar, PE is a promising framework given its better privacy-utility trade-off and the API-only requirement.

5.1.2 Large Distribution Shift (ImageNet → Camelyon17)

Next, we consider a hard task for PE, where the private dataset is very different from ImageNet. We use Camelyon17 dataset (Bandi et al., 2018; Koh et al., 2021) as private data which contains 302436 images of histological lymph node sections with labels on whether it has cancer (real images in Figs. 6 and 22). Despite the large distribution shift, training-based methods can update the model weights to adapt to the private distribution (given enough samples). However, PE can only draw samples from APIs as is.

We find that even in this challenging situation, PE can still achieve non-trivial results. Following Ghalebikesabi et al. (2023), we train a WRN-40-4 classifier from scratch on 302436 generated samples and compute the test accuracy. We achieve 80.33% accuracy with $(10.00, 3 \cdot 10^{-6})$-DP. Prior SOTA (Ghalebikesabi et al., 2023) is 91.1% with $(10, 3 \cdot 10^{-6})$-DP. Random guess is 50%. Fig. 6 (more in Fig. 21) shows that generated images from PE are very similar to Camelyon17 despite that the pre-trained model is on ImageNet. Fig. 23 further shows how the generated images are gradually moved towards Camelyon17 across iterations.

Why PE works under large distribution shifts. Even though the diffusion model is trained on natural images, the support of the generated distribution spans the entire sample space. PE is effective in guiding the model to generate samples from the region that is low-density in the original pre-trained distribution but high-density in the private distribution. See App. J for more explanations.

Limitation. These results demonstrate the effectiveness of PE. But when public models that are similar to private data are not available and when there is enough private data, the traditional training-based methods are still more promising at this point if the privacy-utility trade-off is the only goal. However, given the benefit of API-only assumption and the non-trivial results that PE already got, it is worth further exploiting the potential of PE in future work. Indeed, we expect these results can be improved with further refinement of PE (App. M).

5.1.3 Generating Unlimited Number of Samples

We use the approach in § 4.2 to generate more synthetic samples from § 5.1.1 and train classifiers on them. The results are in Fig. 5. Similar as Ghalebikesabi et al. (2023), the classifier accuracy improves as more generated samples are used. With an ensemble of 5 classifiers, we reach 89.13% accuracy with 1M samples. *This suggests that PE has the same capability as training-based methods in generating an unlimited number of useful samples.*

We also see two interesting phenomena: *(1) The gap between PE and DP-Diffusion diminishes as more samples are used.* We hypothesize that it is due to the limited improvement space: As shown in Ghalebikesabi et al. (2023), even using an ImageNet pre-trained classifier, the best accuracy DP-Diffusion achieves is close to the best points in Fig. 5. *(2) The benefit of PE is more evident over ensembling, especially when having more generated samples.* We hypothesize it is due to different ways of generating more samples. In Ghalebikesabi et al. (2023), the newly generated samples are from the same distribution as the first 50000 samples. In contrast, the newly generated samples in PE are from a different distribution (see § 4.2), which could be more diverse and therefore are more beneficial for ensembling approaches.

5.2 More Challenging Benchmarks with Large Foundation Models

We demonstrate the feasibility of applying PE on large foundation models with Stable Diffusion (Rombach et al., 2022).

Data. Ideally we want to experiment with a dataset that has no overlap with Stable Diffusion's training data.[7] We take the safest approach: we construct two datasets with photos of the author's

[7]The training set of Stable Diffusion is public. However, it is hard to check if a public image or its variants (e.g., cropped, scaled) have been used to produce images in it. Therefore, we resort to our own private data.

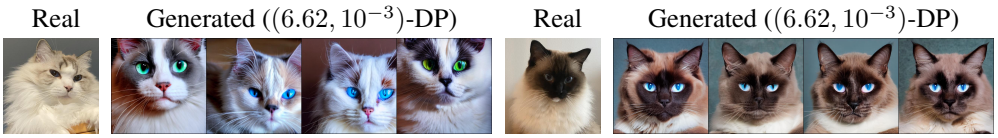


Figure 8: Real & generated images from Cat Cookie (left) and Cat Doudou (right). More in App. K.

two cats that have never been posted online. Each dataset has 100 512x512 images. Such high-resolution datasets with a small number of samples represent a common need in practice (e.g., in health care), but are challenging for DP synthetic data: to the best of our knowledge, no prior training-based methods have reported results on datasets with a similar resolution or number of samples. The dataset is released at https://github.com/microsoft/DPSDA as a new benchmark. See App. K for all images.

API implementation. We use off-the-shelf Stable Diffusion APIs (see App. K).

Results. We run PE for these two datasets with the same hyperparameters. Fig. 8 show examples of generated images for each of the cat datasets. We can see that Private Evolution correctly captures the key characteristics of these two cats. See App. K for all generated images.

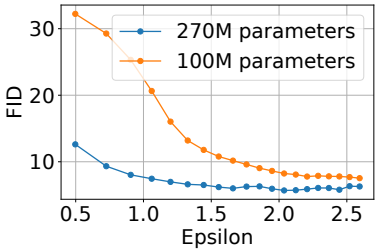


Figure 7: Ablation studies on the pre-trained model. Both are public diffusion models trained on ImageNet (Nichol & Dhariwal, 2021). The 270M network is conditional, whereas the 100M network is unconditional.

5.3 Ablation Studies

Pre-trained network. Fig. 7 shows the results with two different ImageNet pre-trained networks: one is larger (270M) with ImageNet class labels as input; the other one is smaller (100M) without label input (see App. H for implementation details). In all experiments in § 5.1, we used the 270M network. Two takeaways are: (1) The 270M network (trained on the same dataset) improves the results. This is expected as larger and more powerful models can learn public distributions better. This suggests the potential of PE with future foundation models with growing capabilities. (2) Even with a relatively weak model (100M), PE can still obtain good results that beat the baselines (though with a slower convergence speed), suggesting the effectiveness of PE.

More ablation studies on the lookahead degree k, the number of generated samples N_{syn}, the threshold H, and the embedding network are in App. M, where we see that PE obtains good results across a wide range of hyper-parameters, and the main results can be improved with better hyper-parameters.

6 Limitations and Future Work

Algorithms. (1) We did not take the number of API calls into account when optimizing PE. One future work is to optimize the number of API calls along with privacy-utility tradeoffs. (2) We considered two APIs: RANDOM_API and VARIATION_API. It is interesting to consider PE variants that leverage the large set of APIs.[4,5] (3) When the distributions of private data and foundation models are too different, PE achieved non-trivial classification results, but was still worse than SOTA (§ 5.1.2). It is interesting to understand the limits of PE and explore potential improvements. (4) PE requires an embedding network (Eq. (1)) that projects the samples into a space for measuring the similarity between samples. While for images there are plenty of open-source embedding networks to choose, it may not be the case for other modalities. (5) Recent papers show the phenomenon of Model Autophagy Disorder (MAD), where repeatedly *training* the next generative model using synthetic data from the previous one can result in degraded sample quality (Alemohammad et al., 2023). While PE also repeatedly uses synthetic data to create new synthetic data in the main loop (Alg. 1), it is different in two aspects: (a) Instead of purely relying on the synthetic data, PE utilizes the signals from private data to guide the generation; (b) PE does repeated *inference* instead of repeated *training*. It would be interesting to study the MAD effect in the context of PE. (6) Solving DPSDA in the Local/Shuffle DP model and in federated learning settings.

Applications. (1) New privacy-preserving vision applications that were previously challenging but are now possible due to PE's capability of generating high-resolution DP synthetic images with small dataset sizes. (2) The use of PE in other data modalities beyond images such as texts, tabular data, and time series data. (3) Besides DP, there are other parallel/orthogonal privacy concerns, notions, and metrics (Issa et al., 2019; Lin et al., 2022; 2023b; Commission). It is interesting to study if PE can be used to generate privacy-preserving synthetic data with respect to these privacy notations.

7 ETHICS STATEMENT

PE uses the APIs of *pre-trained* models. The DP guarantee of PE is rigorous for the data used in the PE algorithm (i.e., S_{priv}). That being said, PE does not address the privacy of *pre-training* data of foundation models, which is a different goal. PE has no control over the pre-training data—any privacy breaches in the pre-training data are attributable to the data holder (e.g., leaking the data publicly) or the foundation model developer (e.g., using data without permission). However, as a PE user, it is advisable to ensure no overlap between the pre-training data and S_{priv} for liability reasons. Depending on whether the APIs are from *blackbox models*, which can only be accessed through APIs (e.g., DALLE3), or *local models*, whose weights and architectures are accessible by the users (e.g., Stable Diffusion), it has different implications.

- *Using APIs from blackbox models.* Since most blackbox models do not reveal their training dataset, it is safer to only consider S_{priv} that was never been shared or posted online. For instance, a hospital who wants to share a DP synthetic version of its proprietary medical records can safely run PE if it has never released these medical records to any other party, making it impossible for those records to be in the pre-training data of any foundation model.
- *Using APIs from local models.* For local models, we have full control over the model weights and architectures. We can pre-train the models on data that surely has no overlap with the private data. In all experiments of the paper, we use local models including Improved Diffusion (Nichol & Dhariwal, 2021) and Stable Diffusion (Rombach et al., 2022). We directly take the pre-trained models from prior work, and we make sure that the private data and the pre-training data have no overlap.

ACKNOWLEDGEMENT

The authors would like to thank the anonymous reviewers for their valuable feedback and suggestions. The authors would also like to thank Sepideh Mahabadi for the insightful discussions and ideas, and Sahra Ghalebikesabi for the tremendous help in providing the experimental details of DP-Diffusion (Ghalebikesabi et al., 2023). The authors would like to extend their heartfelt appreciation to Cat Cookie and Cat Doudou for generously sharing their adorable faces in the new dataset, as well as to Wenyu Wang for collecting and pre-processing the photos.

† This paper is the full version of our previous workshop paper (Lin et al., 2023a).

REFERENCES

Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pp. 308–318, 2016.

Sina Alemohammad, Josue Casco-Rodriguez, Lorenzo Luzi, Ahmed Imtiaz Humayun, Hossein Babaei, Daniel LeJeune, Ali Siahkoohi, and Richard G Baraniuk. Self-consuming generative models go mad. *arXiv preprint arXiv:2307.01850*, 2023.

Rohan Anil, Badih Ghazi, Vineet Gupta, Ravi Kumar, and Pasin Manurangsi. Large-scale differentially private bert. *arXiv preprint arXiv:2108.01624*, 2021.

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pp. 214–223. PMLR, 2017.

Maria-Florina Balcan, Travis Dick, Yingyu Liang, Wenlong Mou, and Hongyang Zhang. Differentially private clustering in high-dimensional euclidean spaces. In *International Conference on Machine Learning*, pp. 322–331. PMLR, 2017.

Borja Balle and Yu-Xiang Wang. Improving the gaussian mechanism for differential privacy: Analytical calibration and optimal denoising. In *International Conference on Machine Learning*, pp. 394–403. PMLR, 2018.

Peter Bandi, Oscar Geessink, Quirine Manson, Marcory Van Dijk, Maschenka Balkenhol, Meyke Hermsen, Babak Ehteshami Bejnordi, Byungjae Lee, Kyunghyun Paeng, Aoxiao Zhong, et al.

From detection of individual metastases to classification of lymph node status at the patient level: the camelyon17 challenge. *IEEE transactions on medical imaging*, 38(2):550–560, 2018.

Brett K Beaulieu-Jones, Zhiwei Steven Wu, Chris Williams, Ran Lee, Sanjeev P Bhavnani, James Brian Byrd, and Casey S Greene. Privacy-preserving generative deep neural networks support clinical data sharing. *Circulation: Cardiovascular Quality and Outcomes*, 12(7):e005122, 2019.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Tianshi Cao, Alex Bie, Arash Vahdat, Sanja Fidler, and Karsten Kreis. Don't generate me: Training differentially private generative models with sinkhorn divergence. *Advances in Neural Information Processing Systems*, 34:12480–12492, 2021.

Nicholas Carlini, Chang Liu, Úlfar Erlingsson, Jernej Kos, and Dawn Song. The secret sharer: Evaluating and testing unintended memorization in neural networks. In *28th USENIX Security Symposium*, 2019.

Nicholas Carlini, Steve Chien, Milad Nasr, Shuang Song, Andreas Terzis, and Florian Tramer. Membership inference attacks from first principles. *arXiv preprint arXiv:2112.03570*, 2021a.

Nicholas Carlini, Florian Tramèr, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Ulfar Erlingsson, Alina Oprea, and Colin Raffel. Extracting training data from large language models. In *30th USENIX Security Symposium*, USENIX Security '21, pp. 2633–2650. USENIX Association, 2021b.

Nicholas Carlini, Jamie Hayes, Milad Nasr, Matthew Jagielski, Vikash Sehwag, Florian Tramer, Borja Balle, Daphne Ippolito, and Eric Wallace. Extracting training data from diffusion models. *arXiv preprint arXiv:2301.13188*, 2023a.

Nicholas Carlini, Daphne Ippolito, Matthew Jagielski, Katherine Lee, Florian Tramer, and Chiyuan Zhang. Quantifying memorization across neural language models. *International Conference on Learning Representations*, 2023b.

Alisa Chang and Pritish Kamath. Differentially private clustering in google's differential privacy library. Google AI Blog, 2023. URL https://ai.googleblog.com/2021/10/practical-differentially-private.html. https://ai.googleblog.com/2021/10/practical-differentially-private.html.

Christopher A Choquette-Choo, Florian Tramer, Nicholas Carlini, and Nicolas Papernot. Label-only membership inference attacks. In *Proceedings of the 38th International Conference on Machine Learning*, ICML '21, pp. 1964–1974. JMLR, Inc., 2021.

European Commission. Article 29 data protection working party.

Luke N Darlow, Elliot J Crowley, Antreas Antoniou, and Amos J Storkey. Cinic-10 is not imagenet or cifar-10. *arXiv preprint arXiv:1810.03505*, 2018.

Lawrence Davis. Genetic algorithms and simulated annealing. 1987.

Soham De, Leonard Berrada, Jamie Hayes, Samuel L Smith, and Borja Balle. Unlocking high-accuracy differentially private image classification through scale. *arXiv preprint arXiv:2204.13650*, 2022.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in Neural Information Processing Systems*, 34:8780–8794, 2021.

Tim Dockhorn, Tianshi Cao, Arash Vahdat, and Karsten Kreis. Differentially private diffusion models. *arXiv preprint arXiv:2210.09929*, 2022.

Jinshuo Dong, Aaron Roth, and Weijie J Su. Gaussian differential privacy. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 84(1):3–37, 2022.

Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography: Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006. Proceedings 3*, pp. 265–284. Springer, 2006.

Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014.

Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pp. 1322–1333, 2015.

Arun Ganesh, Mahdi Haghifam, Milad Nasr, Sewoong Oh, Thomas Steinke, Om Thakkar, Abhradeep Thakurta, and Lun Wang. Why is public pretraining necessary for private model training? *arXiv preprint arXiv:2302.09483*, 2023.

Sahra Ghalebikesabi, Leonard Berrada, Sven Gowal, Ira Ktena, Robert Stanforth, Jamie Hayes, Soham De, Samuel L Smith, Olivia Wiles, and Borja Balle. Differentially private diffusion models generate useful synthetic images. *arXiv preprint arXiv:2302.13861*, 2023.

Badih Ghazi, Ravi Kumar, and Pasin Manurangsi. Differentially private clustering: Tight approximation ratios. *Advances in Neural Information Processing Systems*, 33:4040–4054, 2020.

Badih Ghazi, Junfeng He, Kai Kohlhoff, Ravi Kumar, Pasin Manurangsi, Vidhya Navalpakkam, and Nachiappan Valliappan. Differentially private heatmaps. *arXiv preprint arXiv:2211.13454*, 2022.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

Sivakanth Gopi, Pankaj Gulhane, Janardhan Kulkarni, Judy Hanwen Shen, Milad Shokouhi, and Sergey Yekhanin. Differentially private set union. In *International Conference on Machine Learning*, pp. 3627–3636. PMLR, 2020.

Sivakanth Gopi, Yin Tat Lee, and Lukas Wutschitz. Numerical composition of differential privacy. *Advances in Neural Information Processing Systems*, 34:11631–11642, 2021.

Niv Haim, Gal Vardi, Gilad Yehudai, Ohad Shamir, and Michal Irani. Reconstructing training data from trained neural networks. *arXiv preprint arXiv:2206.07758*, 2022.

Frederik Harder, Kamil Adamczewski, and Mijung Park. Dp-merf: Differentially private mean embeddings with randomfeatures for practical privacy-preserving data generation. In *International conference on artificial intelligence and statistics*, pp. 1819–1827. PMLR, 2021.

Frederik Harder, Milad Jalali, Danica J Sutherland, and Mijung Park. Pre-trained perceptual features improve differentially private image generation. *Transactions on Machine Learning Research*, 2023.

Jiyan He, Xuechen Li, Da Yu, Huishuai Zhang, Janardhan Kulkarni, Yin Tat Lee, Arturs Backurs, Nenghai Yu, and Jiang Bian. Exploring the limits of differentially private deep learning with group-wise clipping. *arXiv preprint arXiv:2212.01539*, 2022.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

Yuzheng Hu, Fan Wu, Qinbin Li, Yunhui Long, Gonzalo Munilla Garrido, Chang Ge, Bolin Ding, David Forsyth, Bo Li, and Dawn Song. Sok: Privacy-preserving data synthesis. *arXiv preprint arXiv:2307.02106*, 2023.

Ibrahim Issa, Aaron B Wagner, and Sudeep Kamath. An operational approach to information leakage. *IEEE Transactions on Information Theory*, 66(3):1625–1657, 2019.

James Jordon, Jinsung Yoon, and Mihaela Van Der Schaar. PATE-GAN: Generating synthetic data with differential privacy guarantees. In *International conference on learning representations*, 2019.

Pang Wei Koh, Shiori Sagawa, Henrik Marklund, Sang Michael Xie, Marvin Zhang, Akshay Balsubramani, Weihua Hu, Michihiro Yasunaga, Richard Lanas Phillips, Irena Gao, et al. Wilds: A benchmark of in-the-wild distribution shifts. In *International Conference on Machine Learning*, pp. 5637–5664. PMLR, 2021.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Xuechen Li, Florian Tramer, Percy Liang, and Tatsunori Hashimoto. Large language models can be strong differentially private learners. *arXiv preprint arXiv:2110.05679*, 2021.

Xuechen Li, Daogao Liu, Tatsunori B Hashimoto, Huseyin A Inan, Janardhan Kulkarni, Yin-Tat Lee, and Abhradeep Guha Thakurta. When does differentially private learning not suffer in high dimensions? *Advances in Neural Information Processing Systems*, 35:28616–28630, 2022.

Zinan Lin. *Data Sharing with Generative Adversarial Networks: From Theory to Practice*. PhD thesis, Carnegie Mellon University, 2022.

Zinan Lin, Alankar Jain, Chen Wang, Giulia Fanti, and Vyas Sekar. Using gans for sharing networked time series data: Challenges, initial promise, and open questions. In *Proceedings of the ACM Internet Measurement Conference*, pp. 464–483, 2020.

Zinan Lin, Vyas Sekar, and Giulia Fanti. On the privacy properties of gan-generated samples. In *International Conference on Artificial Intelligence and Statistics*, pp. 1522–1530. PMLR, 2021.

Zinan Lin, Shuaiqi Wang, Vyas Sekar, and Giulia Fanti. Distributional privacy for data sharing. In *NeurIPS 2022 Workshop on Synthetic Data for Empowering ML Research*, 2022.

Zinan Lin, Sivakanth Gopi, Janardhan Kulkarni, Harsha Nori, and Sergey Yekhanin. Differentially private synthetic data via foundation model APIs 1: Images. In *NeurIPS 2023 Workshop on Synthetic Data Generation with Generative AI*, 2023a. URL https://openreview.net/forum?id=7GbfIEvoS8.

Zinan Lin, Shuaiqi Wang, Vyas Sekar, and Giulia Fanti. Summary statistic privacy in data sharing. *arXiv preprint arXiv:2303.02014*, 2023b.

Chenlin Meng, Yang Song, Jiaming Song, Jiajun Wu, Jun-Yan Zhu, and Stefano Ermon. Sdedit: Image synthesis and editing with stochastic differential equations. *arXiv preprint arXiv:2108.01073*, 2021.

Ilya Mironov. Rényi differential privacy. In *2017 IEEE 30th computer security foundations symposium (CSF)*, pp. 263–275. IEEE, 2017.

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

OpenAI. Gpt-4 technical report, 2023.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pp. 8748–8763. PMLR, 2021.

Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10684–10695, 2022.

Neil Savage. Synthetic data could be better than real data. *Nature*, April 2023. doi: 10.1038/d41586-023-01445-8. URL https://doi.org/10.1038/d41586-023-01445-8.

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, pp. 2256–2265. PMLR, 2015.

Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020.

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

Dong Su, Jianneng Cao, Ninghui Li, Elisa Bertino, and Hongxia Jin. Differentially private k-means clustering. In *Proceedings of the sixth ACM conference on data and application security and privacy*, pp. 26–37, 2016.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

Xinyu Tang, Richard Shin, Huseyin A Inan, Andre Manoel, Fatemehsadat Mireshghallah, Zinan Lin, Sivakanth Gopi, Janardhan Kulkarni, and Robert Sim. Privacy-preserving in-context learning with differentially private few-shot generation. *arXiv preprint arXiv:2309.11765*, 2023.

Florian Tramèr, Reza Shokri, Ayrton San Joaquin, Hoang Le, Matthew Jagielski, Sanghyun Hong, and Nicholas Carlini. Truth serum: Poisoning machine learning models to reveal their secrets. *arXiv preprint arXiv:2204.00032*, 2022.

Margarita Vinaroz, Mohammad-Amin Charusaie, Frederik Harder, Kamil Adamczewski, and Mi Jung Park. Hermite polynomial features for private data generation. In *International Conference on Machine Learning*, pp. 22300–22324. PMLR, 2022.

Boxin Wang, Weixin Chen, Hengzhi Pei, Chulin Xie, Mintong Kang, Chenhui Zhang, Chejian Xu, Zidi Xiong, Ritik Dutta, Rylan Schaeffer, et al. Decodingtrust: A comprehensive assessment of trustworthiness in gpt models. *arXiv preprint arXiv:2306.11698*, 2023.

Yucheng Yin, Zinan Lin, Minhao Jin, Giulia Fanti, and Vyas Sekar. Practical gan-based synthetic ip header trace generation using netshare. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pp. 458–472, 2022.

Ashkan Yousefpour, Igor Shilov, Alexandre Sablayrolles, Davide Testuggine, Karthik Prasad, Mani Malek, John Nguyen, Sayan Ghosh, Akash Bharadwaj, Jessica Zhao, Graham Cormode, and Ilya Mironov. Opacus: User-friendly differential privacy library in PyTorch. *arXiv preprint arXiv:2109.12298*, 2021.

Da Yu, Saurabh Naik, Arturs Backurs, Sivakanth Gopi, Huseyin A Inan, Gautam Kamath, Janardhan Kulkarni, Yin Tat Lee, Andre Manoel, Lukas Wutschitz, et al. Differentially private fine-tuning of language models. *arXiv preprint arXiv:2110.06500*, 2021.

Xiang Yue, Huseyin A Inan, Xuechen Li, Girish Kumar, Julia McAnallen, Huan Sun, David Levitan, and Robert Sim. Synthetic text generation with differential privacy: A simple and practical recipe. *arXiv preprint arXiv:2210.14348*, 2022.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

A Definition of Wasserstein Distance

Wasserstein distance is a widely used metric in designing (Arjovsky et al., 2017) and evaluating (Heusel et al., 2017) generative models. Given probability distributions μ, ν on a metric space, the Wasserstein distance w.r.t. to a distance function $d(\cdot, \cdot)$ is defined as $W_p(\mu, \nu) = \inf_\gamma \left[\mathbb{E}_{(x,y)\sim\gamma} d(x,y)^p\right]^{1/p}$ where the infimum is over all couplings γ of μ, ν. Also given discrete point sets S, T, we use $W_p(S, T)$ to denote the W_p-distance between uniform distributions on S and T.

B A Brief Introduction to Evolutionary Algorithms

Evolutionary algorithms (Davis, 1987) are inspired by biological evolution, and the goal is to produce samples that maximize an *objective value*. It starts with an *initial population* (i.e., a set of samples), which is then iteratively updated. In each iteration, it selects *parents* (i.e., a subset of samples) from the population according to the *fitness function* which describes how useful they are in achieving better *objective values*. After that, it generates *offsprings* (i.e., new samples) by modifying the parents, hoping to get samples with better objective values, and puts them in the population. By doing so, the population will be guided towards better objective values.

We cannot directly apply existing EA algorithms to our problem. Firstly, our objective is to produce a set of samples that are *jointly* optimal (i.e., closer to the private distribution, § 3.2), instead of optimizing an objective calculated from *individual* samples in typical EA problems. In addition, the differential privacy requirement and the restrictive model API access are unique to our problem. These differences require us to redesign all components of EA.

C More Details on Private Evolution

Alg. 3 shows the full algorithm that supports both unconditional data and conditional data. For simplicity, we assume that the private dataset S_{priv} is balanced, i.e., it has an equal number of samples in each label class. Otherwise, we can first estimate the counts using the Laplace mechanism and use these counts to generate synthetic data of appropriate size in each label class.

Algorithm 3: Private Evolution (PE) for both labeled and unlabeled data.

Input: The set of private classes: C ($C = \{0\}$ if for unconditional generation)
Private samples: $S_{\text{priv}} = \{(x_i, y_i)\}_{i=1}^{N_{\text{priv}}}$, where x_i is a sample and $y_i \in C$ is its label
Number of iterations: T
Number of generated samples: N_{syn} (assuming $N_{\text{syn}} \bmod |C| = 0$)
Noise multiplier for DP Nearest Neighbors Histogram: σ
Threshold for DP Nearest Neighbors Histogram: H

$S_{\text{syn}} \leftarrow \emptyset$
for $c \in C$ **do**
  $private_samples \leftarrow \{x_i | (x_i, y_i) \in S_{\text{priv}} \text{ and } y_i = c\}$
  $S_1 \leftarrow$ RANDOM_API $(N_{\text{syn}}/|C|)$
  for $t \leftarrow 1, \ldots, T$ **do**
    $histogram_t \leftarrow$ DP_NN_HISTOGRAM $(private_samples, S_t, \sigma, H)$ `// See Alg. 2`
    $\mathcal{P}_t \leftarrow histogram_t / \text{sum}(histogram_t)$ `//` $\mathcal{P}_t$ `is a distribution on` S_t
    $S'_t \leftarrow$ draw $N_{\text{syn}}/|C|$ samples with replacement from $\mathcal{P}_t$ `//` S'_t `is a multiset`
    $S_{t+1} \leftarrow$ VARIATION_API (S'_t)
  $S_{\text{syn}} \leftarrow S_{\text{syn}} \cup \{(x, c) | x \in S_T\}$
return S_{syn}

Privacy analysis. For ease of understanding the privacy guarantee of Alg. 3, we can consider a modified version of Alg. 3 as in Alg. 4, where we switch the order of the two for loops over t and

c. Apparently, this modified algorithm gives the same outcome as Alg. 3. The only lines that touch private data are Lines 5 and 6. The input to these lines is the entire private dataset, and the output is a histogram with size N_{syn}. Same as step 1 in the analysis of Alg. 1 (§ 4.3), each private sample only contributes one vote in the histogram. If we add or remove one sample, the resulting histogram will change by 1 in the ℓ_2 norm. Therefore, the sensitivity of these lines is 1. The following privacy analysis follows exactly the same as steps 2-5 in § 4.3, and therefore, the privacy guarantee of Alg. 3 is the same as Alg. 1.

Algorithm 4: Private Evolution (PE) for both labeled and unlabeled data. (Modified from Alg. 3) for the ease of privacy analysis.)

Input: The set of private classes: C ($C = \{0\}$ if for unconditional generation)
Private samples: $S_{\text{priv}} = \{(x_i, y_i)\}_{i=1}^{N_{\text{priv}}}$, where x_i is a sample and $y_i \in C$ is its label
Number of iterations: T
Number of generated samples: N_{syn} (assuming $N_{\text{syn}} \bmod |C| = 0$)
Noise multiplier for DP Nearest Neighbors Histogram: σ
Threshold for DP Nearest Neighbors Histogram: H

1 $S_{\text{syn}} \leftarrow \emptyset$
2 $S_1^c \leftarrow$ RANDOM_API $(N_{\text{syn}}/|C|)$ for each $c \in C$
3 $private_samples^c \leftarrow \{x_i | (x_i, y_i) \in S_{\text{priv}}$ and $y_i = c\}$ for each $c \in C$
4 **for** $t \leftarrow 1, \ldots, T$ **do**
5 **for** $c \in C$ **do**
6 $histogram_t^c \leftarrow$ DP_NN_HISTOGRAM $(private_samples^c, S_t^c, \sigma, H)$ `// See Alg. 2`
7 **for** $c \in C$ **do**
8 $\mathcal{P}_t \leftarrow histogram_t^c / \text{sum}(histogram_t^c)$ `//` $\mathcal{P}_t$ `is a distribution on` S_t^c
9 $S_t' \leftarrow$ draw $N_{\text{syn}}/|C|$ samples with replacement from $\mathcal{P}_t$ `//` S_t' `is a multiset`
10 $S_{t+1}^c \leftarrow$ VARIATION_API (S_t')
11 $S_{\text{syn}} \leftarrow S_{\text{syn}} \cup \{(x, c) | x \in S_T^c, c \in C\}$
12 **return** S_{syn}

D Theoretical Evidence for Convergence of PE

In this section, we will give some intuition for why PE can solve DPWA.

Convergence of Non-Private Evolution. We first analyze Alg. 1 when no noise is added to the histograms (i.e., we set $\sigma = 0$ and $H = 0$ in Line 3). We show that in this case, the evolution algorithm does converge to the private distribution in $O(d)$ iterations where d is the dimension of the embedding space. Under some reasonable modeling assumptions (see App. E), we prove the following theorem. Here D is the diameter of S_{priv}, $L \approx$ number of variations of each point in S_t' that are added to S_{t+1} in Line 6 of Alg. 1.

Theorem 1. *Assume that* $\log L \ll d$.[8] *With probability* $\geq 1 - \tau$, *the non-private evolution algorithm (Alg. 1 with* $\sigma = H = 0$*) outputs* S_{syn} *with Wasserstein distance* $W_p(S_{\text{priv}}, S_{\text{syn}}) \leq \eta$ *after* T *iterations*[9] $\forall p \in [1, \infty]$ *whenever*

$$T \gg \frac{d \log(D/\eta)}{\log L} + \log(N_{\text{priv}}/\tau). \tag{2}$$

This theorem is nearly tight. In each iteration of PE, we get the voting information which is about $\tilde{O}(N_{\text{priv}} \log(L))$ bits. To converge to S_{priv}, we need at least $\tilde{\Omega}(N_{\text{priv}} d)$ bits of information. Therefore we do require at least $\tilde{\Omega}(d/\log L)$ iterations to converge. Here is some intuition for how the

[8] If $\log L \gg d \log(D/\eta)$, i.e., if we generate an exponential number of points then by a simple epsilon-net argument we can prove that the algorithm will converge in a single step.

[9] Number of samples produced using VARIATION_API per iteration is $\leq L \cdot r \cdot N_{\text{priv}} = L \log(D/\eta) N_{\text{priv}}$.

proof of Thm. 1 works. Fix some private point $x \in S_{\text{priv}}$. Let $z^* \in S_t$ be its closest point in S_t. In S_{t+1}, we generate variations of z^* using the VARIATION_API (z^*). We then prove that if $\|x - z^*\| \geq \eta$, then one of the variations will get closer to x than z^* by a factor of $(1 - (\log L)/d)$ with constant probability. Repeating this for T iterations as in Eq. (2), will bring some point in S_T η-close to x.

Convergence of Private Evolution. To get some intuition on the working of PE in the presence of noise, we make a simplifying assumption. We will assume that there are B identical copies of each private point in S_{priv}. We call B as multiplicity. Note that for any DP algorithm to converge to S_{priv}, we need that the private data is well clustered with some minimum cluster size. Any cluster with too few points cannot be represented in S_{syn}, because that would violate DP. And when there is a cluster of B private points and the generated points in S_t are still far from this cluster, then it is likely that all the B private points will have a common closest point in S_t[10], i.e., they all vote for the same point in S_t as a single entity. Therefore multiplicity is a reasonable modeling assumption to make to understand the working of PE. Note that, actually it is very easy to find S_{priv} exactly using DP Set Union (Gopi et al., 2020) with the multiplicity assumption. The point of Thm. 2 is to give intuition about why PE works in practice; it is proved in App. E.2 under the same assumptions as in Thm. 1.

Theorem 2. *Let $0 \leq \varepsilon \leq \log(1/2\delta)$. Suppose each point in S_{priv} has multiplicity B. Then, with high probability ($\geq 1 - \tau$), Private Evolution (Alg. 1) with $\sigma \gg \sqrt{T \log(1/\delta)}/\varepsilon$ and $H \gg \sigma\sqrt{\log(TLN_{\text{priv}}/\tau)}$, when run for T iterations, satisfies (ε, δ)-DP and outputs S_{syn} such that $W_p(S_{\text{priv}}, S_{\text{syn}}) \leq \eta, \forall p \in [1, \infty]$, whenever T satisfies Eq. (2) and multiplicity $B \gg H$.*

Ignoring polylogarithmic factors in $d, L, N_{\text{priv}}, \log(D/\eta), \tau$, we need $T \gg d$ and $B \gg \sqrt{d \log(1/\delta)}/\varepsilon$ for Theorem 2 to hold. Thus, we should expect that PE will discover every cluster of private data of size $\gg \sqrt{d \log(1/\delta)}/\varepsilon$ in $O(d)$ iterations. We now compare this to previous work on DP clustering. Ghazi et al. (2020) gives an algorithm for densest ball, where they show an (ε, δ)-DP algorithm which (approximately) finds any ball of radius r which has at least $\gg \sqrt{d \log(1/\delta)}/\varepsilon$ private points. Thus intuitively, we see that PE compares favorably to SOTA DP clustering algorithms (though we don't have rigorous proof of this fact). If this can be formalized, then PE gives a very different algorithm for densest ball, which in turn can be used to solve DP clustering. Moreover PE is very amenable to parallel and distributed implementations. We therefore think this is an interesting theory problem for future work.

Why Private Evolution works well in practice. We have seen that in the worst case, PE takes $\Omega(d)$ iterations to converge. In our experiments with CIFAR10 and Camelyon17, where $d = 2048$ is the embedding dimension, we see that PE actually converges in only about 20 iterations which is much smaller than d. We offer one plausible explanation for this via *intrinsic dimension*. Suppose the (embeddings of) realistic images lie on a low dimensional manifold M inside $\mathbb{R}^d$ of dimension $d_{\text{intrinsic}} \ll d$ (see experimental results in App. F). Given an image z, VARIATION_API (z) will create variations of z which are also realistic images. Therefore the embeddings of these variations will also lie in the same manifold M, and PE is searching for the private points only inside the manifold M without ever going outside it. Therefore the d that matters for convergence is actually $d_{\text{intrinsic}} \ll d$. In this case, we expect that PE converges in $O(d_{\text{intrinsic}})$ iterations and discovers clusters of private points of size at least $\sqrt{d_{\text{intrinsic}} \log(1/\delta)}/\varepsilon$.

E PROOFS OF PE CONVERGENCE THEOREMS

We will slightly modify the algorithm as necessary to make it convenient for our analysis. We will make the following modeling assumptions:

- The private dataset S_{priv} is contained in an ℓ_2 ball of diameter D and RANDOM_API will also produce initial samples in the same ball of diameter D. This is a reasonable assumption in practice, as images always have bounded pixel values: for the original images in UINT8 data type, each pixel is in the range of $[0, 255]$; in diffusion models, they are usually normalized to $[-1, 1]$ (i.e., 0 corresponds to -1 and 255 corresponds to 1), and all generated images are guaranteed to be in this range.

[10] In fact, it is easy to formally prove that there will be a common approximate nearest neighbor.

- The distance function used in Alg. 2 is just the ℓ_2 norm, $d(x, z) = \|x - z\|_2$.
- The distribution of points we output is $S_{\text{syn}} = \mathcal{P}_T$ (i.e., we output a distribution of points).
- $S'_t \supset \text{supp}(\mathcal{P}_t) = \text{supp}(histogram_t)$.[11]
- $S_{t+1} = S'_t \cup \bigcup_{z \in S'_t} \text{VARIATION_API}(z)$ where $\text{VARIATION_API}(z)$ samples L samples each from Gaussian distributions $\mathcal{N}(z, \sigma_i^2 I)$ for $\sigma_i = \frac{D\sqrt{\log L}}{2^i d}$ where $1 \le i \le r$ and $r = \log(D/\eta)$ where $\eta > 0$ is the final Wasserstein distance.

E.1 Proof of Thm. 1

Proof. Fix a point $x \in S_{\text{priv}}$ and some iteration t. Suppose $z^* \in S_t$ is the closest point to x. Since x will vote for z^* in $histogram_t$, $z^* \in \text{supp}(\mathcal{P}_t) \subset S'_t$. Therefore $\text{VARIATION_API}(z^*) \subset S_{t+1}$. Let $V = \text{VARIATION_API}(z^*)$.

Claim 1. *If $\|x - z^*\| \ge \eta$, then with probability at least $1/2$, some point in V will get noticeably closer to x than z^*, i.e.,*

$$\min_{z \in V} \|x - z\|_2 \le \left(1 - \frac{\log L}{4d}\right) \|x - z^*\|_2 .$$

Proof. Let $s = \|x - z^*\|$ and let $\sigma \in \{\sigma_1, \sigma_2, \dots, \sigma_r\}$ be such that $\sigma d/\sqrt{\log L} \in [s/2, s]$. Note that such a σ exists since $s \in [\eta, D]$. We will now prove that one of the L samples $z_1, z_2, \dots, z_L \sim \mathcal{N}(z^*, \sigma^2 I_d)$ will get noticeably closer to x than z^*. Let $z_i = z^* + \sigma w_i$ where $w_i \sim \mathcal{N}(0, I_d)$.

$$\begin{aligned}\min_{i \in [L]} \|x - z_i\|_2^2 &= \|x - z^*\|_2^2 + \min_{i \in [L]} \left(\sigma^2 \|w_i\|_2^2 - 2\sigma \langle x - z^*, w_i \rangle\right) \\ &\le s^2 + \max_{i \in [L]} \sigma^2 \|w_i\|_2^2 - \max_{i \in [L]} 2\sigma \langle x - z^*, w_i \rangle\end{aligned}$$

Note that $\|w_i\|_2^2$ is a χ_d^2 random variable. By using upper tail bounds for χ_d^2 distribution and union bound over $i \in [L]$, we can bound

$$\Pr\left[\max_{i \in [L]} \|w_i\|_2^2 \ge 3d/2\right] \le L \exp(-\Omega(d)) \ll 1.$$

The distribution of $\langle x - z^*, w_i \rangle$ is the same as $\|x - z^*\|_2 \tilde{w}_i$ where $\tilde{w}_1, \dots, \tilde{w}_L$ are i.i.d. $\mathcal{N}(0, 1)$ random variables. By using the fact that max of L i.i.d. Gaussians is at least $\sqrt{\log L}$ with probability at least $3/4$ (for $L \gg 1$), we get that

$$\Pr\left[\max_{i \in [L]} \langle x - z^*, w_i \rangle \le s\sqrt{\log L}\right] \le \frac{1}{4}.$$

Combining everything we get:

$$\begin{aligned}\frac{1}{2} &\ge \Pr\left[\min_{i \in [L]} \|x - z_i\|_2^2 \ge s^2 + (3/2)d\sigma^2 - 2s\sigma\sqrt{\log L}\right] \\ &\ge \Pr\left[\min_{i \in [L]} \|x - z_i\|_2^2 \ge \max_{\lambda \in [1/2, 1]} s^2 + (3/2)d\left(\frac{\lambda s\sqrt{\log L}}{d}\right)^2 - 2s\left(\frac{\lambda s\sqrt{\log L}}{d}\right)\sqrt{\log L}\right] \\ &\ge \Pr\left[\min_{i \in [L]} \|x - z_i\|_2^2 \ge s^2 + \left(\frac{s^2 \log L}{d}\right) \max_{\lambda \in [1/2, 1]} (3\lambda^2/2 - 2\lambda)\right] \\ &\ge \Pr\left[\min_{i \in [L]} \|x - z_i\|_2^2 \ge s^2\left(1 - \frac{\log L}{2d}\right)\right] \\ &\ge \Pr\left[\min_{i \in [L]} \|x - z_i\|_2 \ge s\left(1 - \frac{\log L}{4d}\right)\right]\end{aligned}$$

where last inequality uses the fact that $\sqrt{1 - t} \le 1 - \frac{t}{2}$ for $t \le 1$. □

[11] This may not be true in the original algorithm due to sampling, we need to modify it so that $S'_t \supset \text{supp}(\mathcal{P}_t)$.

Now in T iterations, $\min_{z \in S_t} \|x - z\|_2$ will shrink by a factor of $\left(1 - \frac{\log L}{4d}\right)$ in at least $T/4$ iterations with probability $1 - \exp(-\Omega(T)) \geq 1 - \frac{\tau}{N_{\text{priv}}}$ (by standard Chernoff bounds). Note that in iterations where it doesn't shrink, it doesn't grow either since $S'_t \subset S_{t+1}$. Similarly, if $\min_{z \in S_t} \|x - z\|_2 \leq \eta$ for some iteration, it will remain so in all subsequent iterations. Therefore after $T \gg \frac{d \log(D/\eta)}{\log L}$ iterations, $\min_{z \in S_T} \|x - z\|_2 \leq \eta$ with probability at least $1 - \frac{\tau}{N_{\text{priv}}}$. By union bounding over all points we get that, with probability at least $1 - \tau$, for every point $x \in S_{\text{priv}}$ there is a point in S_T which is η-close. This proves that $W_p(S_{\text{priv}}, \mathcal{P}_T) \leq \eta$. □

E.2 Proof of Thm. 2

Proof. Since we are doing T iterations of Gaussian mechanism with noise level σ, we need to set $\sigma \gg \sqrt{T \log(1/\delta)}/\varepsilon$ to satisfy (ε, δ)-DP (Dwork et al., 2014) when $\varepsilon \leq log(1/2\delta)$. Let $x \in S_{\text{priv}}$ be a point with multiplicity B. If $z^* \in S_t$ is the closest point to x, then it will get B votes. After adding $\mathcal{N}(0, \sigma^2)$ noise, if $B \gg H \gg \sigma\sqrt{\log(TLN_{\text{priv}}/\tau)}$, then with probability at least $1 - \tau/(4T)$, the noisy votes that z^* gets is still above the threshold H. Therefore z^* will survive in S_{t+1} as well. Also since $H \gg \sigma\sqrt{\log(TLN_{\text{priv}}/\tau)}$, with probability $1 - \tau/(4T)$, points in S_t which do not get any votes (there are LN_{priv} of them) will not survive even after adding noise and thresholding by H. Therefore, by union bounding over all T iterations, with probability at least $1 - \tau/2$, the algorithm behaves identically to the non-private algorithm. Therefore by an identical proof as in the non-private analysis, we can prove that after T iterations $W_p(S_{\text{priv}}, \mathcal{P}_T) \leq \eta$ with probability at least $1 - \tau$. □

F Intrinsic Dimension of Image Embeddings

To illustrate the intrinsic dimension of image embeddings, we use the following process:

1. We (randomly) take an image x from CIFAR10.
2. We use VARIATION_API from App. I to obtain 3000 image variations of x: $x_1, \ldots, x_{3000}$, and their corresponding inception embeddings $g_1, ..., g_{3000} \in \mathbb{R}^{2048}$. 3000 is chosen so that the number of variations is larger than the embedding dimension.
3. We construct a matrix $M = [g_1 - g; \ldots; g_{3000} - g] \in \mathbb{R}^{3000 \times 2048}$, where g is the mean$(g_1, \ldots, g_{3000})$.
4. We compute the singular values of M: $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_{2048}$.
5. We compute the minimum number of singular values n needed so that the explained variance ratio[12] $\sum_{i=1}^{n} \sigma_i^2 / \sum_{i=1}^{2048} \sigma^2 \geq 0.8$. Intuitively, this n describes how many dimensions are needed to reconstruct the embedding changes M with a small error. We use it as an estimated intrinsic dimension of the image variations.

We conduct the above process with the variation degree $[98, 96, 94, 92, 90, 88, 86, 84, 82, 80, 78, 76, 74, 72, 70, 68, 66, 64, 62, 60]$ utilized in the CIFAR10 experiments (see App. I). We additionally add a variation degree of 100 which is the highest variation degree in the API that was used to generate the initial samples. We plot the estimated intrinsic dimension v.s. variation degree in Fig. 9. The raw original singular values of $M/\sqrt{3000}$ for variation degree=60 are in Fig. 10 (other variation degrees have similar trend). Two key observations are:

- As the variation degree increases, the estimated intrinsic dimension also increases. This could be because the manifold of image embeddings is likely to be non-linear, the above estimation of intrinsic dimension is only accurate when we perturb the image x to a small degree so that the changes in the manifold can still be well approximated by a linear subspace. Using a larger variation degree (and thus larger changes in the embedding space) will overestimate the intrinsic dimension.

[12] See https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html.

- Nevertheless, we always see that the singular values decrease rapidly (Fig. 10) and the estimated intrinsic dimension is much smaller than the embedding size 2048 (Fig. 9), which supports our hypothesis in App. D.

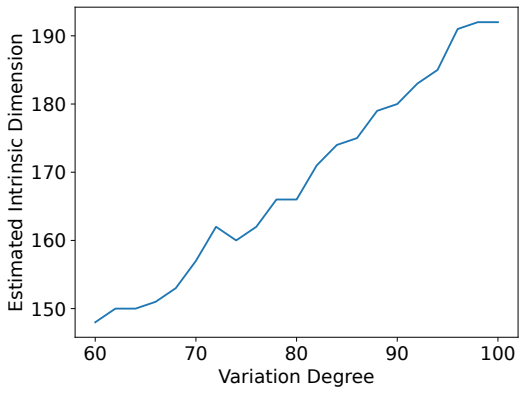


Figure 9: Estimated intrinsic dimension of inception embeddings of realistic images.

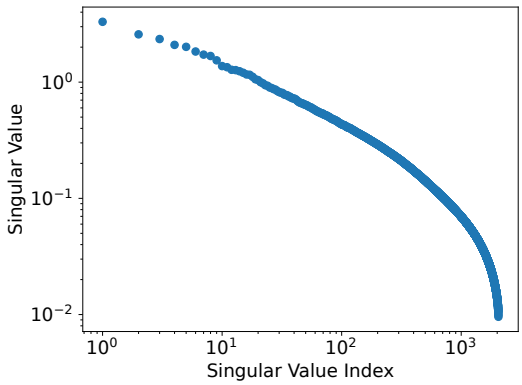


Figure 10: Singular values of inception embeddings of image variations at variation degree=60.

G Relation of DPWA to prior work

Recall that in DPWA, we want a DP algorithm to output S_{syn} which is close to the distribution of S_{priv} in Wasserstein distance w.r.t. some distance function $d(\cdot,\cdot)$. When the distance function $d(\cdot,\cdot)$ is just ℓ_2 distance between the samples (i.e., ℓ_2 in the pixel space for images), then DPWA is closely related to DP Clustering (Ghazi et al., 2020; Balcan et al., 2017; Su et al., 2016) and DP Heatmaps (Ghazi et al., 2022).

In (Ghazi et al., 2022), to give an algorithm for DP Heatmaps, the authors study DP sparse EMD[13] aggregation problem where we need to output a distribution of points which approximates the distribution of private data in EMD distance (i.e., W_1). They study this problem only in two dimensions and the running time of their algorithms (suitably generalized to higher dimensions) will be exponential in the dimension d.

The DP Clustering problem requires us to output a clustering of private data using DP. The most common clustering studied is k-means clustering where we should output k cluster centers such that k-means cost is minimized, where k-means cost is the sum of squares of ℓ_2-distance of each data point to its nearest cluster center. Note that in DPWA, if the number of synthetic data points N_{syn} is specified to be k, then DP k-means clustering and DPWA with W_2 metric are equivalent. In (Ghazi

[13]Earth's Mover Distance, which is the another name for Wasserstein metric W_1.

et al., 2022), a polynomial time DP Clustering algorithm with an additional k-means cost (over what is non-privately possible) of $k\sqrt{d\log(1/\delta)}\mathrm{polylog}(N_{\mathrm{priv}}, d)/\epsilon$ is given. This can be converted into an upper bound on the Wasserstein distance. But this is not a practical algorithm. The privacy-utility tradeoffs are bad due to the large hidden constants in the analysis and the authors don't provide an implementation. There is a practical DP Clustering algorithm (along with an implementation) given in (Chang & Kamath, 2023) (but with no theoretical guarantees).

G.1 Why Not Just Use DP Clustering?

We now explain why we can't just use prior work on DP Clustering to solve DPSDA say for images.

Clustering in the image space. We can use DP k-means Clustering to cluster the images w.r.t. ℓ_2 metric in the pixel space. This doesn't work because ℓ_2 distance in the pixel space doesn't capture semantic similarity. An image which is slightly shifted in pixel space gets very far in ℓ_2 distance. And the dimension of the images is too large for prior DP Clustering algorithms to work well. Their convergence and privacy-utility tradeoffs depend too strongly on the dimension.

Clustering in the embedding space. We can use DP k-means Clustering to cluster the image embeddings w.r.t. ℓ_2 metric in the embedding space. Note that this is the distance function we use in PE (Eq. (1)). Even after we find the cluster centers, it is hard to invert the embedding map (i.e., find an image whose embedding is close to a given vector in the embedding space).[14] Moreover the dimension of the embedding space is still too large for the above methods to be practical.

Our PE algorithm does much better because:

1. Its distance function is ℓ_2 in the embedding space which captures semantic similarity,
2. It exploits the intrinsic dimension of the manifold of images in the embedding space which is much smaller than the embedding dimension (see App. D and App. F) and
3. There is no need to invert points in embedding space to the image space.

In an early experiment, we have tried DP clustering in the CLIP embedding space using the practical DP Clustering algorithm in (Chang & Kamath, 2023). We then inverted the cluster centers (which are in the embedding space) using unCLIP. But we found the resulting images are too noisy compared to the images we get from PE and the FID scores are also significantly worse than that of PE.

H Implementation Details on Label Condition

There are two meanings of "conditioning" that appear in our work:

1. Whether the pre-trained networks or APIs (e.g., ImageNet pre-trained diffusion models used in § 5.1.1 and 5.1.2) support conditional input (e.g., ImageNet class label).
2. Whether the generated samples are associated with class labels from the private data.

In DP fine-tuning approaches, these two usually refer to the same thing: if we want to generate class labels for generated samples, the common practice is to use a pre-trained network that supports conditional input (Ghalebikesabi et al., 2023). However, in PE, these two are completely orthogonal.

Conditional pre-trained networks/APIs. We first explain our implementation when the pre-trained networks or APIs support conditional inputs such as class labels or text prompts. When generating the initial population using RANDOM_API (Line 1), we will either randomly draw labels from all possible labels when no prior public information is available (which is what we do in CIFAR10 and Camelyon17 experiments where we randomly draw from all possible ImageNet classes), or use the public information as condition input (e.g., the text prompt used in Stable Diffusion experiments; see App. K). In the subsequent VARIATION_API calls (Line 6), for each image, we will use its associated class label or text prompt as the condition information to the API, and the output samples from VARIATION_API will be associated with the same class label or text prompt as the input sample. For

[14] Some special embeddings such as CLIP embedding do have such an inverse map called unCLIP (Ramesh et al., 2022).

example, if we use an image with "peacock" class to generate variations, all output images will be associated with "peacock" class for future VARIATION_API calls. Note that throughout the above process, all the condition inputs to the pre-trained networks/APIs are public information; they have nothing to do with the private classes.

Conditional generation. Conditional generation is achieved by Alg. 3, where we separate the samples according to their class labels, and run the main algorithm (Alg. 1) on each sample set. We can use either conditional or unconditional pre-trained networks/APIs to implement it.

Throughout the paper, "(un)condition" refers to 2, expect the caption in Fig. 7 which refers to 1.

I More Details and Results on CIFAR10 Experiments

Pre-trained model. By default, we use the checkpoint `imagenet64_cond_270M_250K.pt` released in (Nichol & Dhariwal, 2021).[15] For the ablation study of the pre-trained network, we additionally use the checkpoint `imagenet64_uncond_100M_1500K.pt`.

API implementation. RANDOM_API follows the standard diffusion model sampling process. VARIATION_API is implemented with SDEdit (Meng et al., 2021), which adds noise to input images and lets the diffusion model denoise them. We use DDIM sampler (Song et al., 2020) and the default noise schedule to draw samples. Note that these choices are not optimal; our results can potentially be improved by using better noise schedules and the full DDPM sampling (Ho et al., 2020) which are known to work better. The implementation of the above APIs is straightforward without touching the core modeling part of diffusion models and is similar to the standard API implementations in Stable Diffusion (App. K).

Hyperparameters. We set number of iterations $T = 20$, lookahead degree $k = 8$, and number of generated samples $N_{\text{syn}} = 50000$. For RANDOM_API and VARIATION_API, we use DDIM sampler with 100 steps. For VARIATION_API, we use SDEEdit (Meng et al., 2021) by adding noise till $[98, 96, 94, 92, 90, 88, 86, 84, 82, 80, 78, 76, 74, 72, 70, 68, 66, 64, 62, 60]$ timesteps for each iteration respectively. These timesteps can be regarded as the v parameter in § 3.3.

For the experiments in Fig. 4, we use noise multiplier $\sigma = t \cdot \sqrt{2}$ and threshold $H = 2t$ for $t \in \{5, 10, 20\}$, and pick the the pareto frontier. Fig. 11 shows all the data points we got. Combining this figure with Fig. 4, we can see that PE is not very sensitive to these hyper-parameters, and even with less optimal choices PE still outperforms the baselines.

For the experiments in Fig. 5, we use noise multiplier $\sigma = \sqrt{2}$ and threshold $H = 2$ (i.e., $t = 1$).

For the experiments in Figs. 3 and 13, we use noise multiplier $\sigma = 10\sqrt{2}$ and threshold $H = 20$ (i.e., $t = 10$) and the number of PE iteration is 5 (i.e., the point of "Ours (cond)" in Fig. 4 that has FID$\leq$ 7.9).

For downstream classification (Fig. 5), we follow (Ghalebikesabi et al., 2023) to use WRN-40-4 classifier (Zagoruyko & Komodakis, 2016). We use the official repo[16] without changing any hyper-parameter except adding color jitter augmentation according to (Ghalebikesabi et al., 2023). The ensemble of the classifier is implemented by ensembling the logits.

FID evaluation. Compared to Fig. 4 in the main text, Fig. 12 shows the full results of two versions of (Harder et al., 2023). Baseline results are taken from (Harder et al., 2023; Ghalebikesabi et al., 2023).

Classification accuracy. In § 5.1.1, we show the classification accuracy at a single privacy budget $\epsilon = 3.34$. In Table 1, we further show how the classification accuracy evolves with respect to different ϵs. These results are from the first 5 PE iterations.

Generated samples. See Figs. 13 and 14 for generated images and real images side-by-side. Note that the pre-trained model we use generates 64x64 images, whereas CIFAR10 is 32x32. In Fig. 3,

[15] https://github.com/openai/improved-diffusion

[16] https://github.com/szagoruyko/wide-residual-networks/tree/master/pytorch

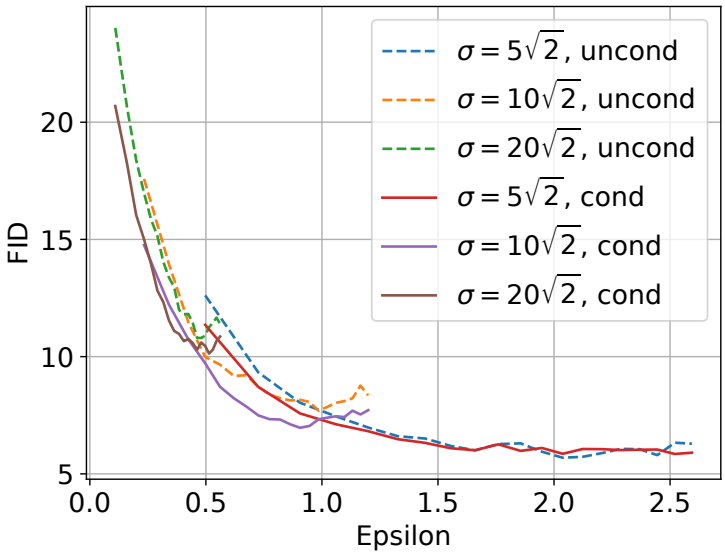


Figure 11: FID (lower is better) v.s. privacy cost ϵ ($\delta = 10^{-5}$) on CIFAR10 with different noise multipliers and thresholds. "(Un)cond" means (un)conditional generation.

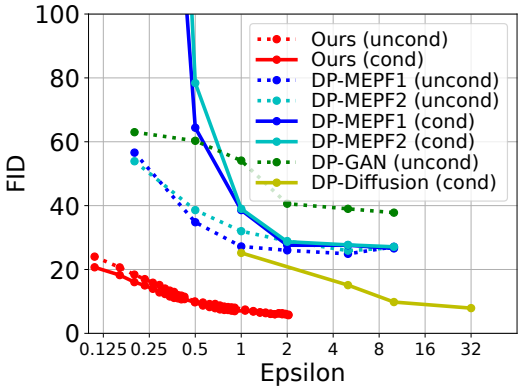


Figure 12: FID (Heusel et al., 2017) (lower is better) v.s. privacy cost ϵ on CIFAR10 ($\delta = 10^{-5}$). Baseline results are taken from (Harder et al., 2023; Ghalebikesabi et al., 2023). (Un)cond means (un)conditional generation. Ours achieves the best privacy-quality trade-off.

we show the raw generated 64x64 images; in Fig. 13, we scale them down to 32x32 for better comparison with the real images.

Nearest samples in the private dataset Figs. 15 and 16 show generated images and their nearest neighbors in the private dataset evaluated using two distance metrics: ℓ_2 distance in the inception embedding space and the pixel space. We can see that the generated images are different from private images. This is expected due to the DP guarantee.

Distributions of the distances to nearest samples. Continuing the above experiments, we further show the distribution of the distances between (1) generated samples and their nearest real samples and (2) real samples and their nearest generated samples in Figs. 17 and 18 respectively. Two key observations are: (1) During the early PE iterations, the distances tend to decrease. This means that PE is effective in pushing the generated distribution to be closer to the private distribution. (2) However, as PE continues, the distances stop decreasing. It is expected, as DP upper bounds the probability of reconstructing any sample in the private dataset.

Samples with the highest and the lowest votes. Fig. 19 shows the samples with the highest and the lowest votes in DP Nearest Neighbors Histogram across different PE iterations. We can see that DP Nearest Neighbors Histogram picks samples similar to the private data as desired. This is more obvious in the first two PE iterations, where DP Nearest Neighbors Histogram assigns high votes on the samples with correct classes and puts low votes on the samples with incorrect classes.

ϵ	Accuracy
1.36	72.46%
1.99	78.78%
2.50	80.83%
2.94	81.15%
3.34	81.74%

Table 1: Classification accuracy v.s. privacy cost ϵ on CIFAR10 ($\delta = 10^{-5}$).

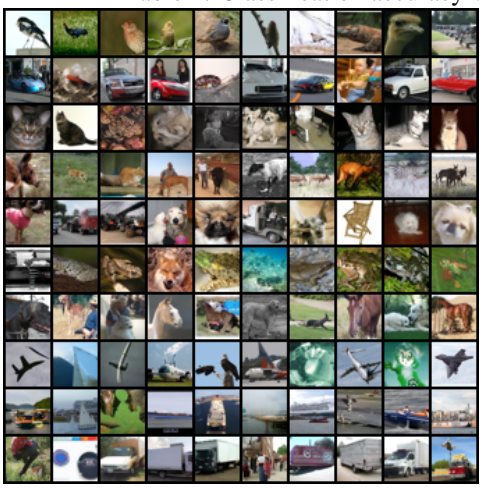


Figure 13: Generated samples on CIFAR10 with $(0.67, 10^{-5})$-DP. Each row corresponds to one class. FID=7.87.

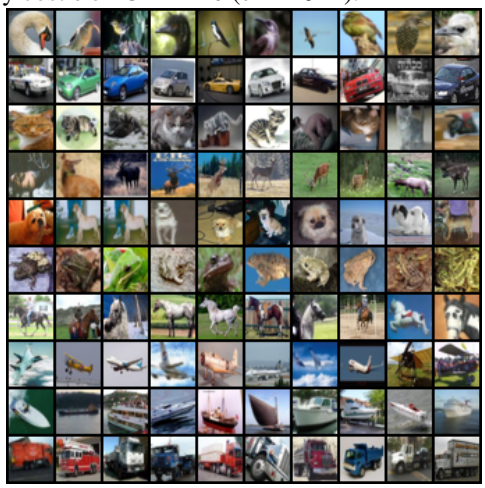


Figure 14: Real samples from CIFAR10. Each row corresponds to one class.

Distribution of counts in DP Nearest Neighbors Histogram. Fig. 20 shows that the standard deviation of counts in DP Nearest Neighbors Histogram tends to decrease with more PE iterations on CIFAR10. This means that the histogram becomes "more uniform" with more PE iterations. It is expected for the following reasons. In the beginning, only a few generated samples are close to the private data. Those generated samples get most of the votes from private samples, which results in a concentrated histogram. PE will then pick those highly voted generated samples and do more variations over them. As a result, private samples that voted for the same generated sample may now find different closest generated samples and distribute the votes, which results in a "more uniform" histogram.

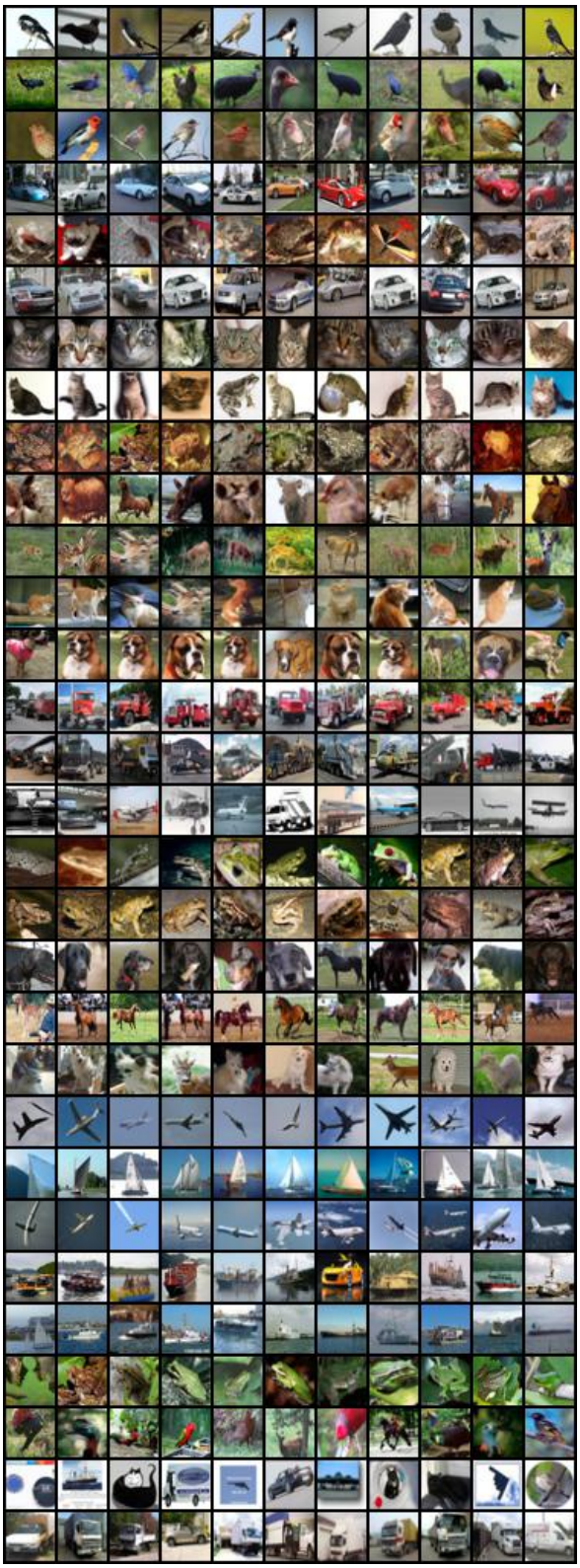
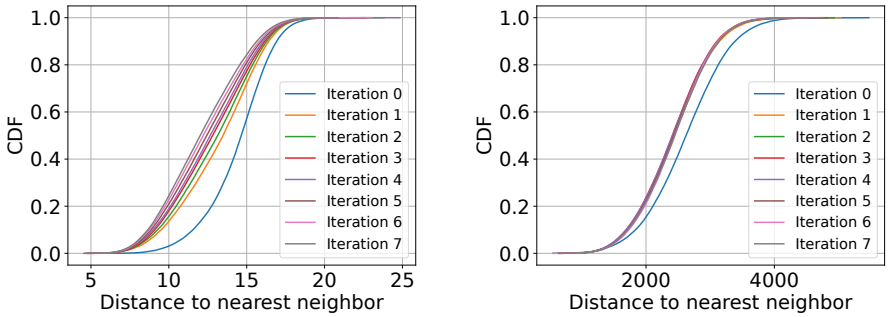


Figure 15: Nearest samples in the private dataset on CIFAR10. In each row, the first column is a generated image (from Fig. 13), and the other columns are its nearest neighbors in the private dataset, sorted by the distance in ascending order. Every three rows correspond to generated image from one class. The distance metric is ℓ_2 in the **inception embedding space.**

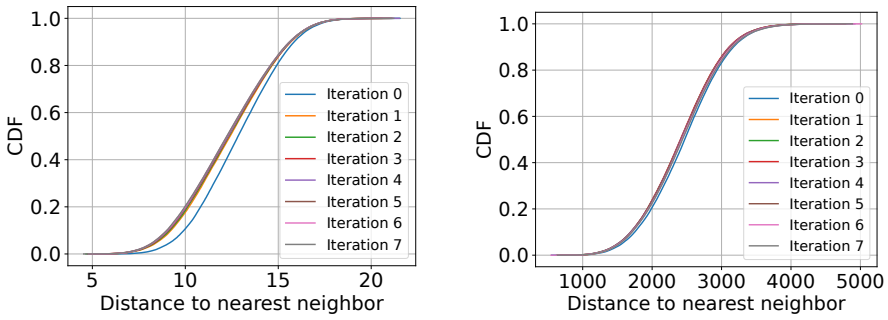
Figure 16: Nearest samples in the private dataset on CIFAR10. In each row, the first column is a generated image (from Fig. 13), and the other columns are its nearest neighbors in the private dataset, sorted by the distance in ascending order. Every three rows correspond to generated image from one class. The distance metric is ℓ_2 in the **pixel space.**



(a) The distance metric is ℓ_2 in the **inception embedding space.**

(b) The distance metric is ℓ_2 in the **pixel space.**

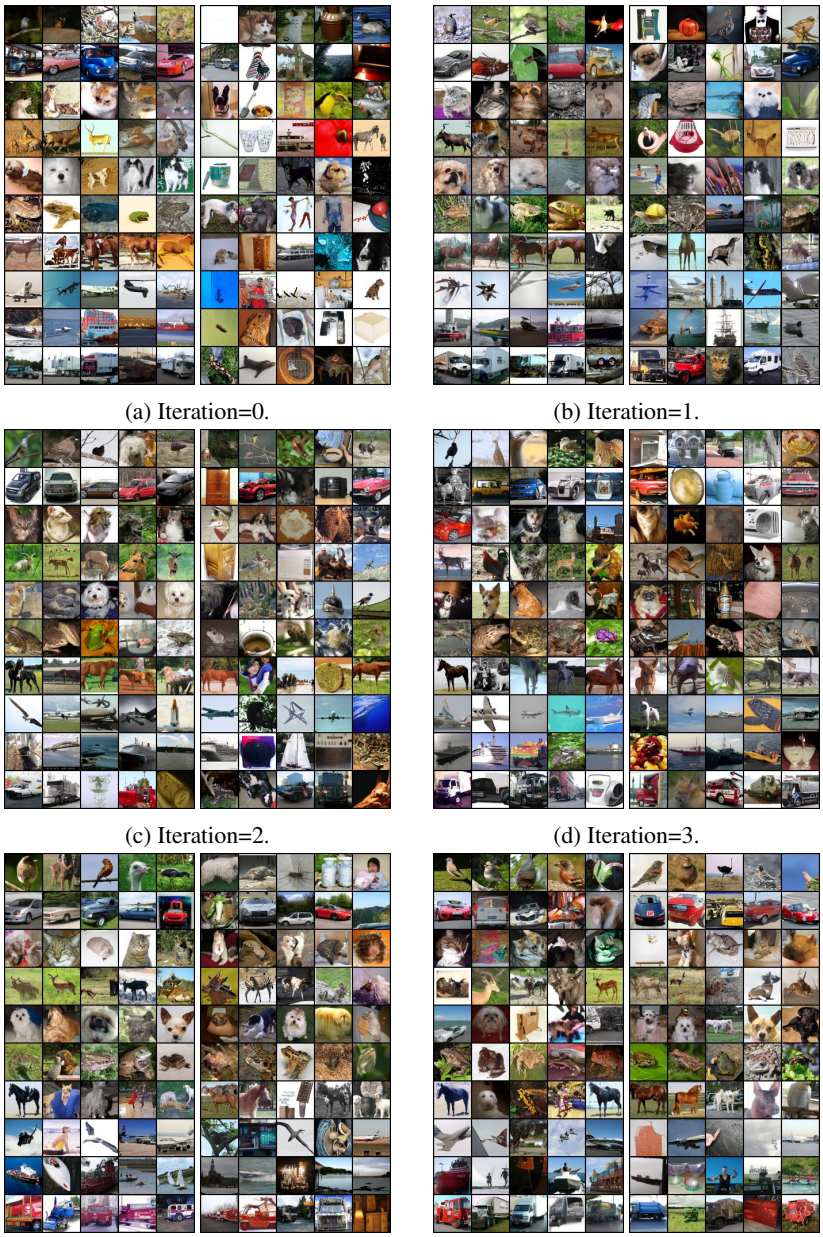
Figure 17: CDF of the distributions **between each generated sample and its nearest private samples** on CIFAR10 across different PE iterations. "Iteration 0" refers to the initial random samples from Line 1 in Alg. 1.



(a) The distance metric is ℓ_2 in the **inception embedding space.**

(b) The distance metric is ℓ_2 in the **pixel space.**

Figure 18: CDF of the distributions **between each private sample and its nearest generated samples** on CIFAR10 across different PE iterations. "Iteration 0" refers to the initial random samples from Line 1 in Alg. 1.



(a) Iteration=0. (b) Iteration=1.

(c) Iteration=2. (d) Iteration=3.

(e) Iteration=4. (f) Iteration=5.

Figure 19: Samples with the highest and the lowest counts in the DP Nearest Neighbors Histogram on CIFAR10. In each subfigure, each row corresponds to one class; the left subfigure shows the samples with the highest counts and the right subfigure shows the samples with the lowest counts.

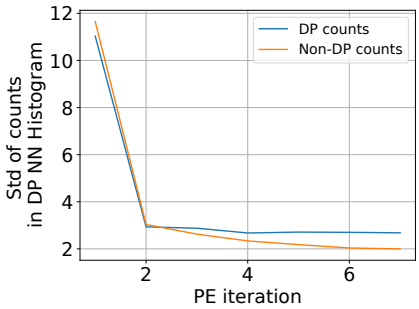


Figure 20: The standard deviation of counts in DP Nearest Neighbors Histogram across different PE iterations on CIFAR10. "DP counts" refers to the counts after adding Gaussian noise and thresholding. "Non-DP counts" refer to the counts before adding Gaussian noise and thresholding.

J More Details and Resutls on Camelyon17 Experiments

Pre-trained model. We use the checkpoint `imagenet64_cond_270M_250K.pt` released in (Nichol & Dhariwal, 2021).[17]

API implementation. Same as App. I.

Hyperparameters. We set lookahead degree $k = 8$, and number of generated samples $N_{\text{syn}} = 302436$.

About the experiments in Fig. 21. For RANDOM_API and VARIATION_API, we use DDIM sampler with 10 steps. For VARIATION_API, we take a 2-stage approach. the first stage, we use DDIM sampler with 10 steps and use SDEEdit (Meng et al., 2021) by adding noise till $[10, 10, 10, 10, 9, 9, 9, 9, 9, 8, 8, 8, 8, 8, 7, 7, 7, 7]$ timesteps for each iteration respectively. In the second stage, we use DDIM sampler with 40 steps and use SDEEdit (Meng et al., 2021) by adding noise till $[20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5]$ timesteps for each iteration respectively. These timesteps can be regarded as the v parameter in § 3.3. We use noise multiplier $\sigma = 2 \cdot \sqrt{2}$ and threshold $H = 4$.

About the experiments in § 5.1.2. For RANDOM_API and VARIATION_API, we use DDIM sampler with 10 steps. For VARIATION_API, we use DDIM sampler with 10 steps and use SDEEdit (Meng et al., 2021) by adding noise till $[10, 10, 10, 10, 9, 9, 9]$ timesteps for each iteration respectively. These timesteps can be regarded as the v parameter in § 3.3. We use noise multiplier $\sigma = 1.381$ and threshold $H = 4$.

Generated samples. See Figs. 21 and 22 for generated images and real images side-by-side. Note that the real images in Camelyon17 dataset are 96x96 images, whereas the pre-trained network is 64x64. In Fig. 22, we scale them down to 64x64 for better comparison.

Fig. 23 shows the generated images in the intermediate iterations. We can see that the generated images are effectively guided towards Camelyon17 though it is very different from the pre-training dataset.

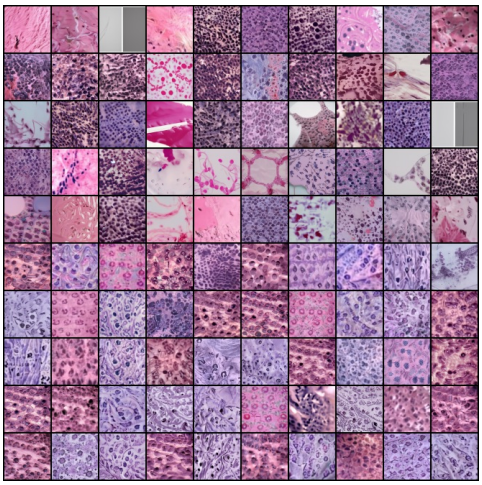


Figure 21: Generated samples on Camelyon17 with $(9.92, 3 \cdot 10^{-6})$-DP. The first five rows correspond to one class, and the rest correspond to the other class. FID=10.66.

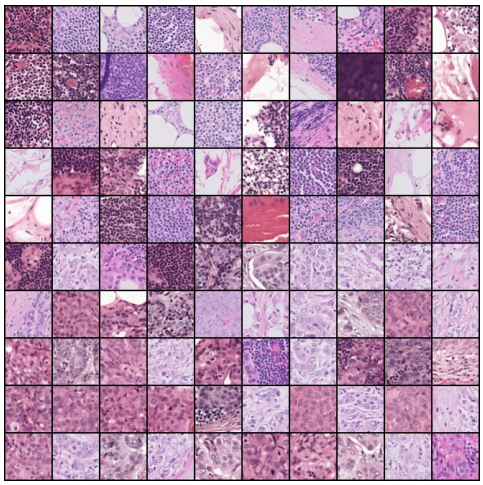
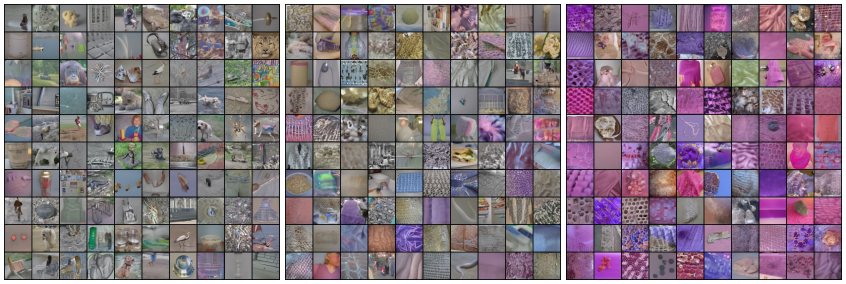


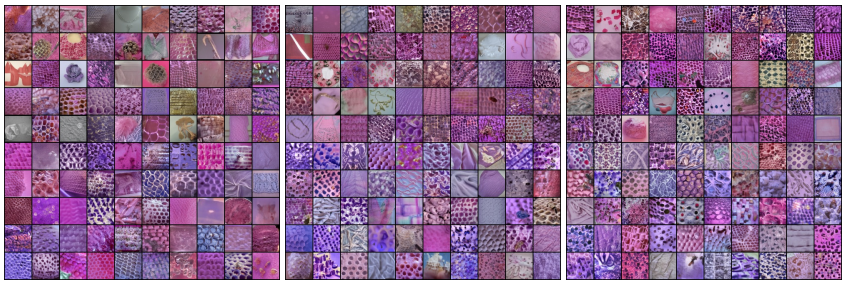
Figure 22: Real samples from Camelyon17. The first five rows correspond to one class, and the rest correspond to the other class.

Nearest samples in the private dataset Figs. 24 and 25 show generated images and their nearest neighbors in the private dataset evaluated using two distance metrics: ℓ_2 distance in the inception embedding space and the pixel space. Similar to the results in CIFAR10, we can see that the generated images are different from private images. This is expected due to the DP guarantee.

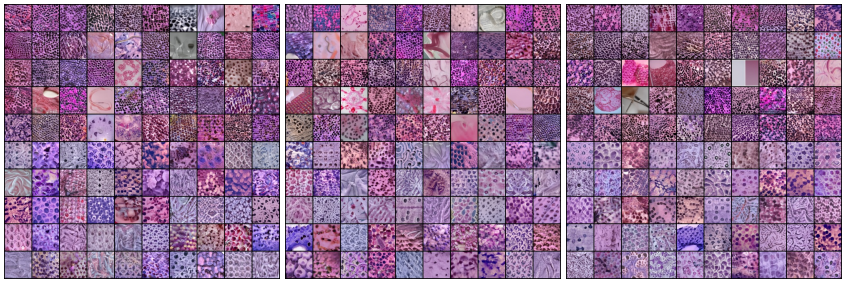
[17] https://github.com/openai/improved-diffusion



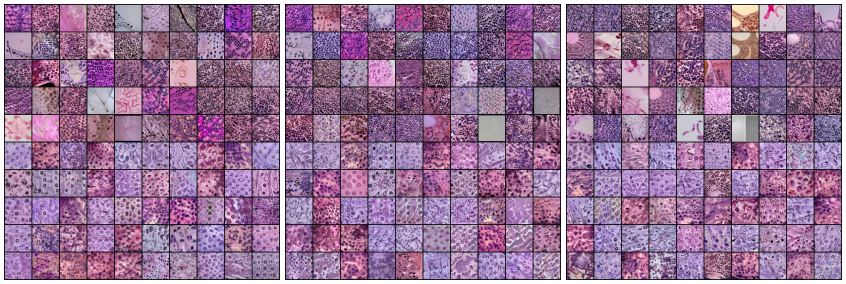
(a) Iteration=0. (b) Iteration=1. (c) Iteration=3.



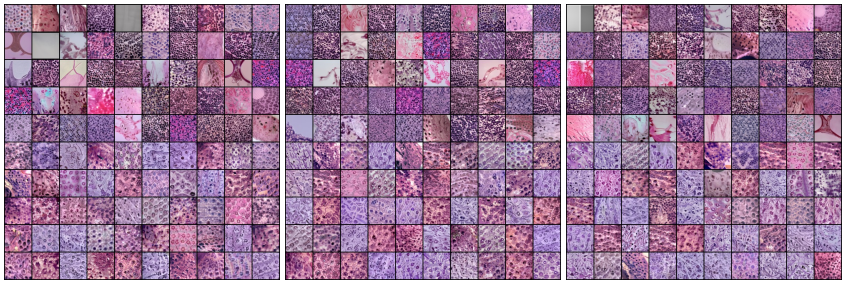
(d) Iteration=5. (e) Iteration=7. (f) Iteration=9.



(g) Iteration=11. (h) Iteration=13. (i) Iteration=15.



(j) Iteration=17. (k) Iteration=19. (l) Iteration=21.



(m) Iteration=23. (n) Iteration=25. (o) Iteration=27.

Figure 23: Generated samples on Camelyon17 at the first few iterations. We can see that the generated images are gradually guided from ImageNet, the pre-training dataset, to Camelyon17, the (very different) private dataset. "Iteration=0" means the initial random samples from RANDOM_API.

Samples with the highest and the lowest votes. Fig. 26 shows the samples with the highest and the lowest votes in DP Nearest Neighbors Histogram across different PE iterations. We can see that DP Nearest Neighbors Histogram gradually picks samples with the right patterns as the private data, and drops samples with more different patterns. As Camelyon17 is further away from the pre-training dataset ImageNet than CIFAR10, we can see that it converges slower than the case in CIFAR10 (Fig. 19).

Distributions of the distances to nearest samples. Continuing the above experiments, we further show the distribution of the distances between (1) generated samples and their nearest real samples and (2) real samples and their nearest generated samples in Figs. 27 and 28 respectively. Similar to the CIFAR10 experiments (Figs. 17 and 18), we see that: (1) During the early PE iterations, the *inception* distances tend to decrease. This means that PE is effective in pushing the generated distribution to be closer to the private distribution. (2) However, as PE continues, the *inception* distances stop decreasing. It is expected, as DP upper bounds the probability of reconstructing any sample in the private dataset. However, one difference to CIFAR10 results is that the distance between each generated sample and its nearest private samples measured in the *original pixel space* (Fig. 27b) tends to increase. We hypothesize that it is be due to nature of this dataset that any shifts of the histological images are still in-distribution but can result in a high distance in the *original pixel space*.

Distribution of counts in DP Nearest Neighbors Histogram. Fig. 29 shows that the standard deviation of counts in DP Nearest Neighbors Histogram tends to decrease with more PE iterations on Camelyon17. This accords with the observations and takeaways messages in CIFAR10 (Fig. 20).

Inspescting the ImageNet class labels of the generated samples. As discussed in App. H, each generated samples have an associated ImageNet label. Here, we inspect those labels to understand how PE works in this dataset. The list of the labels and their associated number of generated images are: "honeycomb" (164647), "velvet" (83999), "nematode, nematode worm, roundworm" (35045), "handkerchief, hankie, hanky, hankey" (14495), "bib" (3102), "head cabbage" (934), "bubble" (142), "stole" (72). We see that 54.4% images are with the label "honeycomb". Many honeycomb images in ImageNet have a "net-like" structure, which shares some similarities with Camelyon17. However, the details, colors, and structures are still different. We compute the FID between honeycomb images and CIFAR10. The FID score is 162.86, which is much higher than the FID score of the final generated images we get (10.66 in Fig. 21). These results show that the existence of the honeycomb class in ImageNet helps with PE, but PE does more than just picking this class.

Why PE works under large distribution shifts. The fact that PE is able to make ImageNet-pre-trained-models to generate Camelyon17 images may appear surprising to readers. Here, we provide intuitive explanations. Even though the diffusion model is trained on natural images, the support of the generated distribution could be very large due to the formulation. More concretely, let's look at two examples: score-based models (Song & Ermon, 2019), which is closely related to diffusion models, and the diffusion model we used (Ho et al., 2020). For score-based models, its modeling target is a distribution perturbed by Gaussian noise, and therefore, the generated distribution spans the entire sample space. For diffusion models (Ho et al., 2020), the latent space has the same size as the image, and the last denoising step is modeled as a distribution derived from a Gaussian distribution that spans the entire pixel space (see Section 3.3 of Ho et al. (2020)). Therefore, the generated distribution of diffusion models also spans the entire sample space.

In other words, for any (trained) score-based models or diffusion models, it is theoretically possible to generate images similar to the private dataset (or in fact, generate any images). The problem is that the probability of generating such images is small if there is a large distribution shift from the pre-training data to the private data. PE is effective in guiding the diffusion model to generate samples from the region that is low-density in the original pre-trained distribution but high-density in the private distribution.

To make it more concrete, we provide how the generated images evolve from natural images to Camelyon17 dataset in Fig. 23 and the selected/filtered samples by PE in Fig. 26. At every iteration, PE selects the set of images that are most similar to Camelyon17 dataset (Fig. 26). Those images might still appear different from Camelyon17 in early iterations. However, as long as we get images that are more similar to Camelyon17 through VARIATION_API at every iteration, we will make progress, and finally, we can get images similar to Camelyon17 (Fig. 21).

We further experiment PE under different levels of distribution shifts. To do that, we take the Camelyon17 dataset and modify the *saturation* of the images to create a sequence of datasets, each with a different saturation change. This way, we create a sequence of datasets with different levels of distribution shifts from ImageNet. We can see that no matter what the level of distribution shifts is, PE can consistently improve the generated distribution towards the private data.

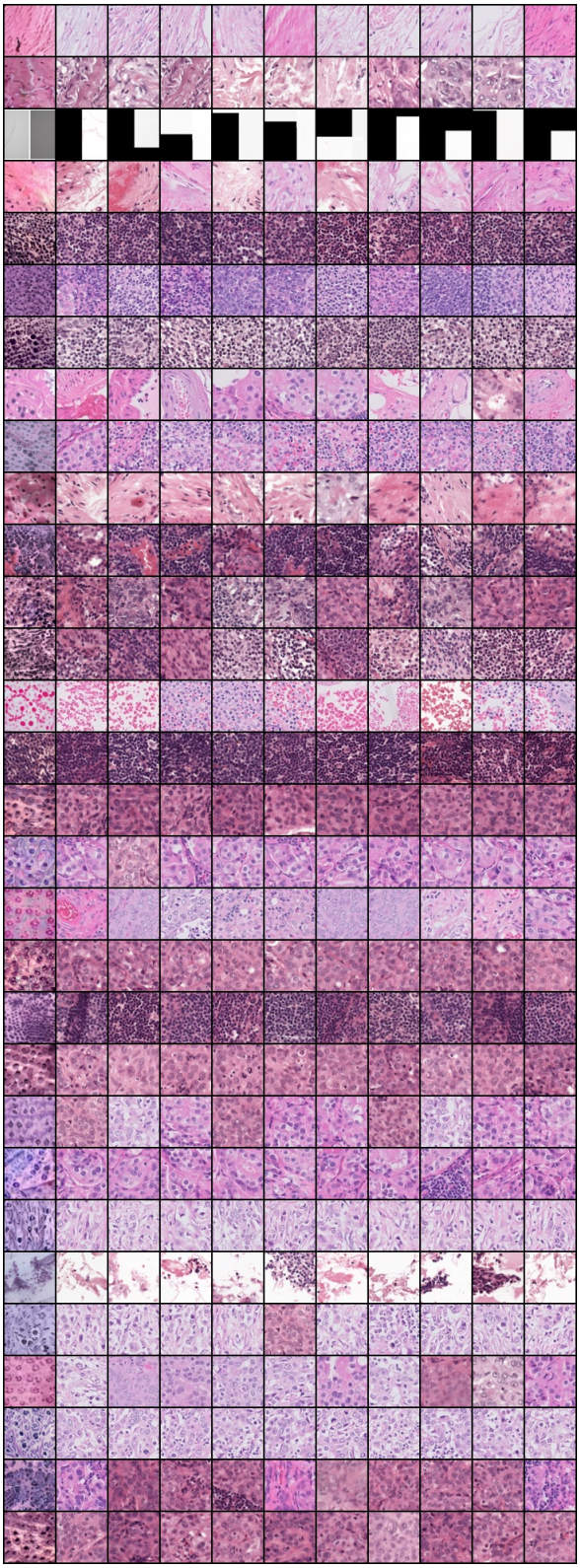


Figure 24: Nearest samples in the private dataset on Camelyon17. In each row, the first column is a generated image (from Fig. 21), and the other columns are its nearest neighbors in the private dataset, sorted by the distance in ascending order. Every fifteen rows correspond to generated image from one class. The distance metric is ℓ_2 in the **inception embedding space**.

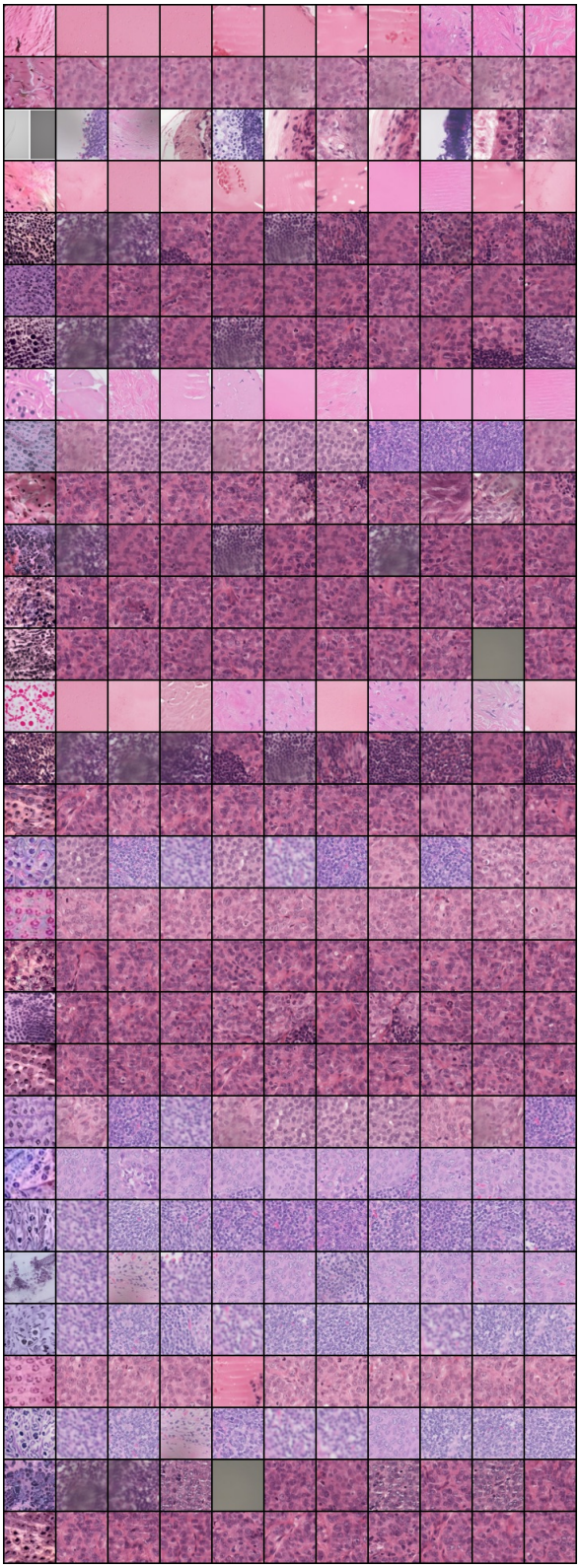
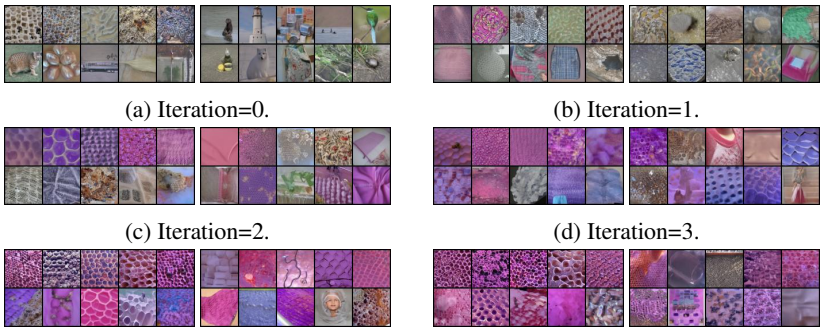


Figure 25: Nearest samples in the private dataset on Camelyon17. In each row, the first column is a generated image (from Fig. 13), and the other columns are its nearest neighbors in the private dataset, sorted by the distance in ascending order. Every fifteen rows correspond to generated image from one class. The distance metric is ℓ_2 in the **pixel space**.

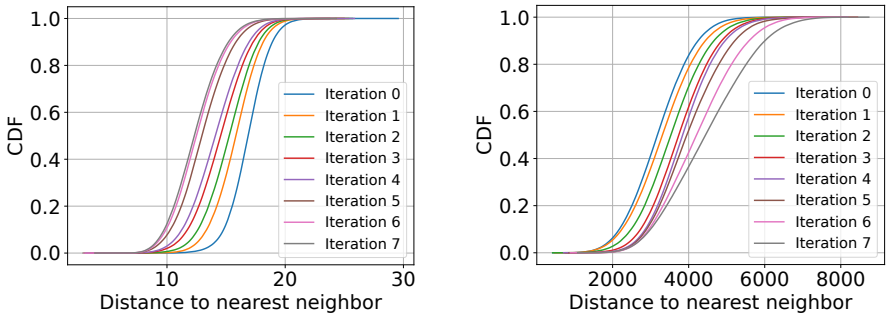


(a) Iteration=0. (b) Iteration=1.

(c) Iteration=2. (d) Iteration=3.

(e) Iteration=4. (f) Iteration=5.

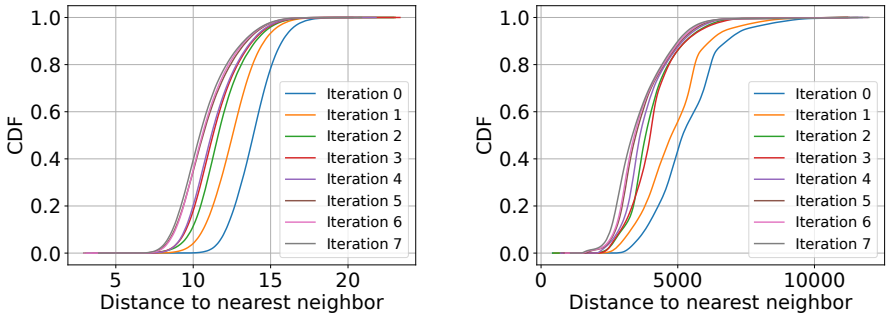
Figure 26: Samples with the highest and the lowest counts in the DP Nearest Neighbors Histogram on Camelyon17. In each subfigure, each row corresponds to one class; the left subfigure shows the samples with the highest counts and the right subfigure shows the samples with the lowest counts.



(a) The distance metric is ℓ_2 in the **inception embedding space.**

(b) The distance metric is ℓ_2 in the **pixel space.**

Figure 27: CDF of the distributions **between each generated sample and its nearest private samples** on CIFAR10 across different PE iterations. "Iteration 0" refers to the initial random samples from Line 1 in Alg. 1.



(a) The distance metric is ℓ_2 in the **inception embedding space.**

(b) The distance metric is ℓ_2 in the **pixel space.**

Figure 28: CDF of the distributions **between each private sample and its nearest generated samples** on CIFAR10 across different PE iterations. "Iteration 0" refers to the initial random samples from Line 1 in Alg. 1.

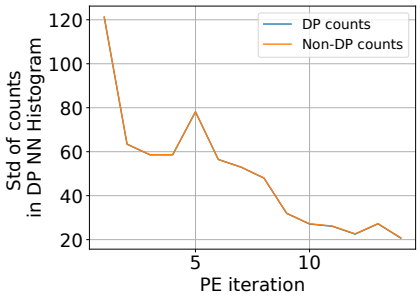


Figure 29: The standard deviation of counts in DP Nearest Neighbors Histogram across different PE iterations on Camelyon17. "DP counts" refers to the counts after adding Gaussian noise and thresholding. "Non-DP counts" refer to the counts before adding Gaussian noise and thresholding.

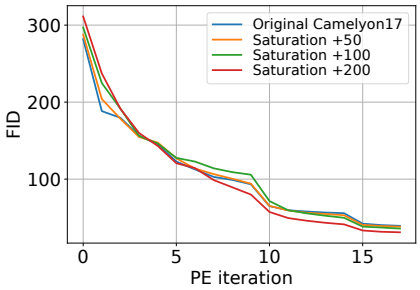


Figure 30: FID vs. PE iterations under Camelyon17 variants with different levels of distribution shifts.

K More Details and Results on Stable Diffusion Experiments

Dataset construction. We start with cat photos taken by the authors, crop the region around cat faces with a resolution larger than 512x512 manually, and resize the images to 512x512. We construct two datasets, each for one cat with 100 images. See Figs. 31 and 32 for all images. Note that having multiple samples from the same identity is not meaningful from the practical perspective of DP. Instead of regarding these datasets as real-world use cases, they should be treated as "toy datasets" for experimenting DP generative models with a small number of high-resolution images.

Figure 31: All images from Cat Cookie dataset. The original resolution is 512x512; we resize them to 128x128 here for reducing the file size of the paper.

API implementation. We use off-the-shelf open-sourced APIs of Stable Diffusion. For RANDOM_API, we use the text-to-image generation API[18], which is implemented by the standard diffusion models' guided sampling process. For VARIATION_API, we use the image-to-image generation API[19], which allows us to control the degree of variation. Its internal implementation is SDEdit (Meng et al., 2021), which adds noise to the input images and runs diffusion models' denoising process.

Hyperparameters We set lookahead degree $k = 8$, and number of generated samples $N_{\text{syn}} = 100$. For RANDOM_API and VARIATION_API, we use the default DDIM sampler with 50 steps. For RANDOM_API, we use the prompt "A photo of ragdoll cat". This gives reaonable cat images but still far away from the private data (Fig. 35). For VARIATION_API, we use variation degrees $[0.98, 0.96, 0.94, 0.92, 0.90, 0.88, 0.84, 0.8, 0.76, 0.72, 0.68, 0.64, 0.6]$ for each iteration re-

[18] https://huggingface.co/docs/diffusers/api/pipelines/stable_diffusion/text2img

[19] https://huggingface.co/docs/diffusers/api/pipelines/stable_diffusion/img2img

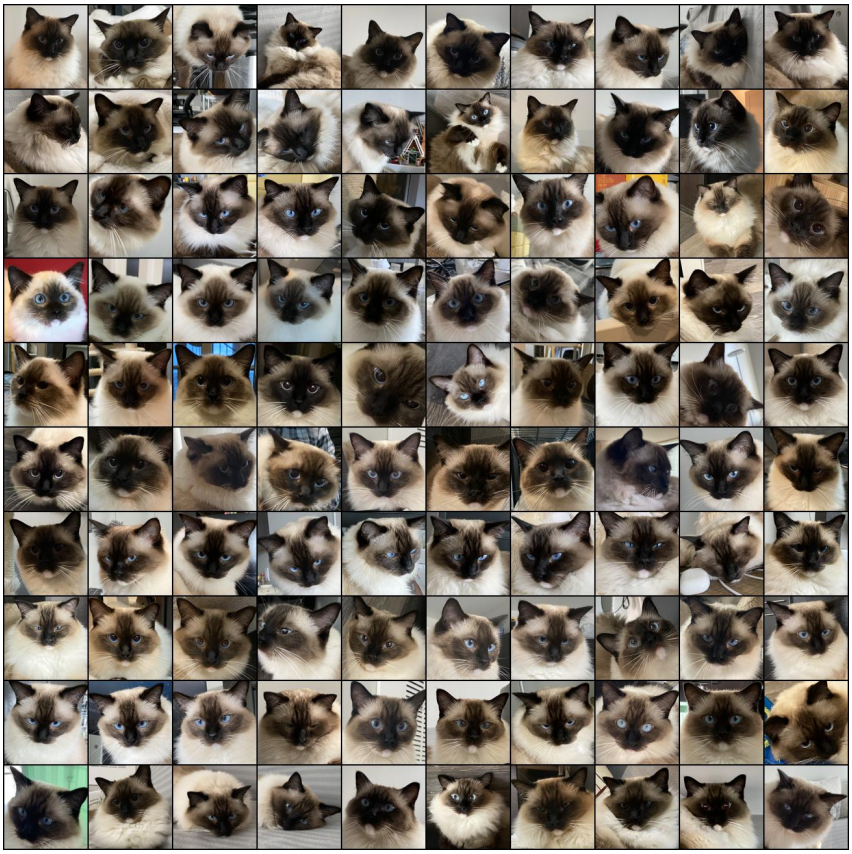


Figure 32: All images from Cat Doudou dataset. The original resolution is 512x512; we resize them to 128x128 here for reducing the file size of the paper.

spectively with the same prompt. We use noise multiplier $\sigma = 2$ and threshold $H = 2$. We use inception embedding for Eq. (1).

Generated images. We use the same hyper-parameters to run PE on two datasets separately. This can also be regarded as running the conditional version of PE (Alg. 3) on the whole dataset (with labels Cat Cookie or Cat Doudou) together. All generated images are in Figs. 33 and 34. While the two experiments use completely the same hyper-parameters, and the initial random images are very different from the cats (Fig. 35), our PE can guide the generated distribution in the right direction and the final generated images do capture the key color and characteristics of each of the cats. This demonstrates the effectiveness of PE with large foundation models such as Stable Diffusion.

We also observe that the diversity of generated images (e.g., poses, face directions) is limited compared to the real data. However, given the small number of samples and the tight privacy budget, this is an expected behavior: capturing more fine-grained features of each image would likely violate DP.

Generated images with more diversity. To make the generated images more diverse, we can utilize the approach in § 4.2, which passes the generated images through VARIATION_API. We have demonstrated in § 5.1.3 that this approach can generate more samples that are useful for downstream classification tasks. Here, we use it for a different purpose: enriching the diversity of generated samples.

Figure Figs. 36 and 37 show the results. We can see that this simple approach is able to generate cats with a more diverse appearance. This is possible because the foundation model (Stable Diffusion) has good prior knowledge about cats learned from massive pre-training, and PE is able to utilize that effectively.

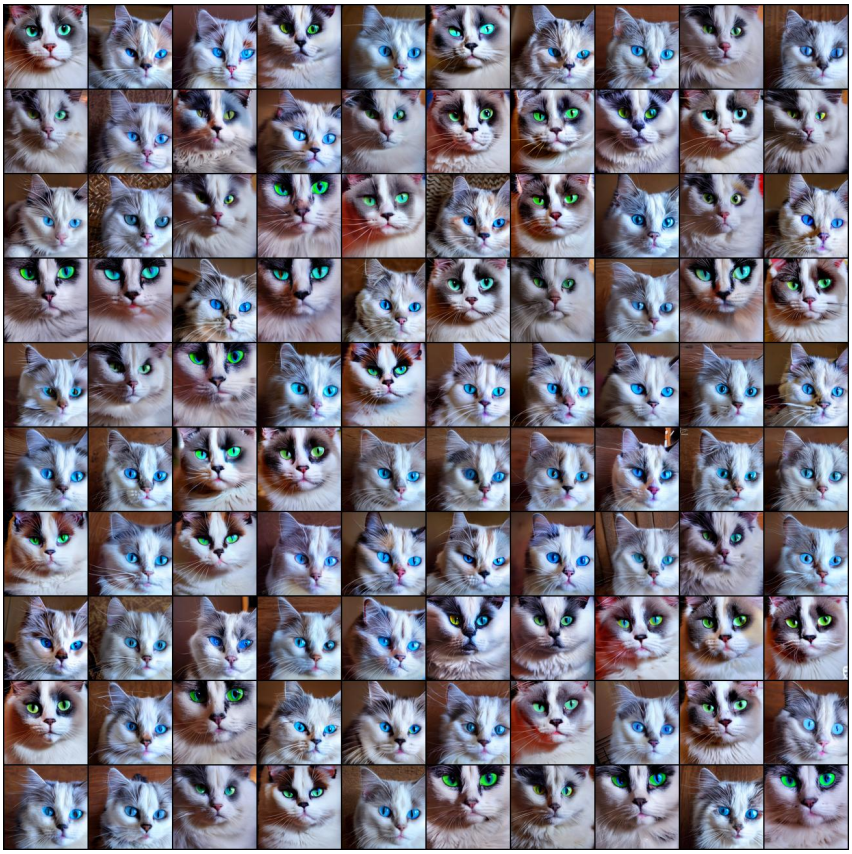


Figure 33: All generated images from Cat Cookie dataset with $(6.62, 10^{-3})$-DP. The original resolution is 512x512; we resize them to 128x128 here for reducing the file size of the paper.

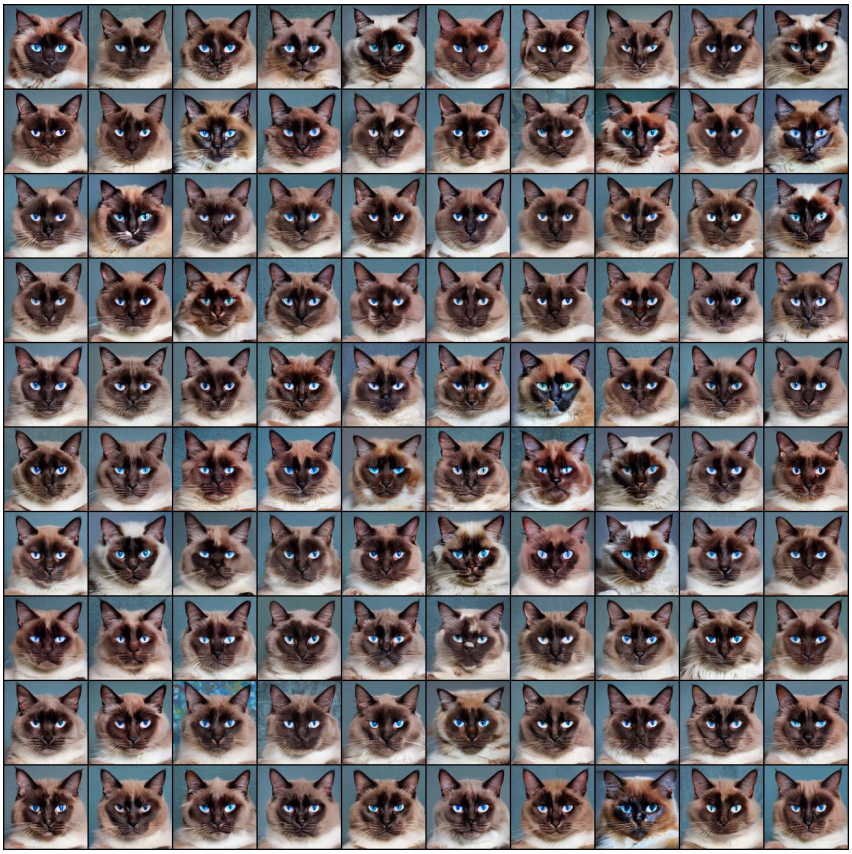


Figure 34: All generated images from Cat Doudou dataset with $(6.62, 10^{-3})$-DP. The original resolution is 512x512; we resize them to 128x128 here for reducing the file size of the paper.

Figure 35: The initial random images from Stable Diffusion. The original resolution is 512x512; we resize them to 128x128 here for reducing the file size of the paper.

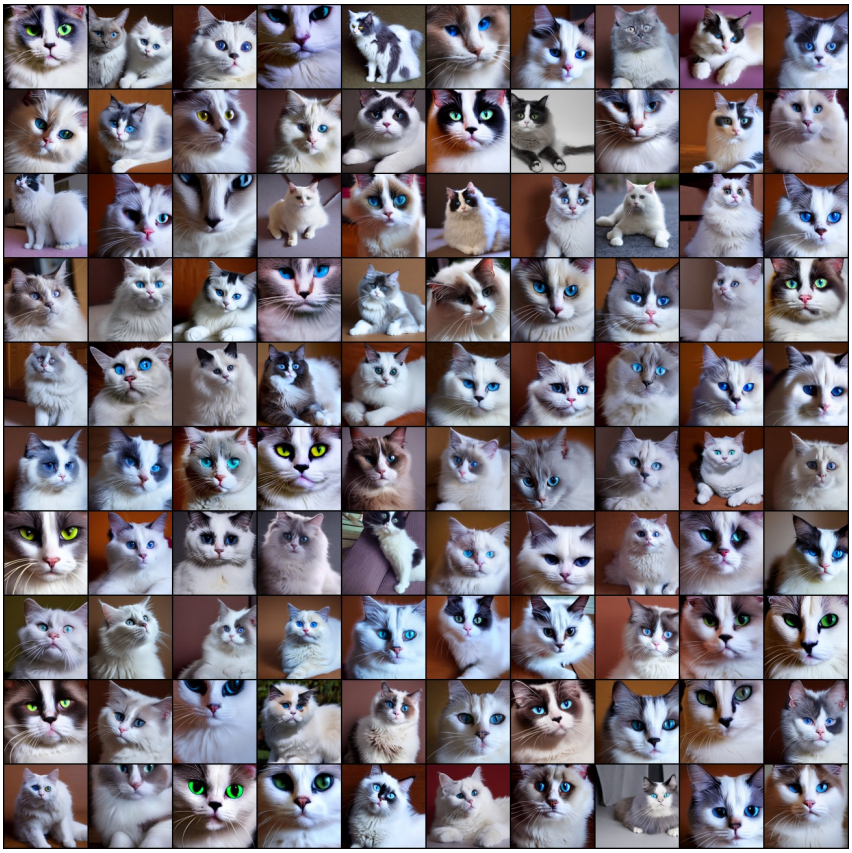


Figure 36: Generated images with enhanced diversity using the approach in § 4.2 on Cat Cookie. The original resolution is 512x512; we resize them to 128x128 here for reducing the file size of the paper.

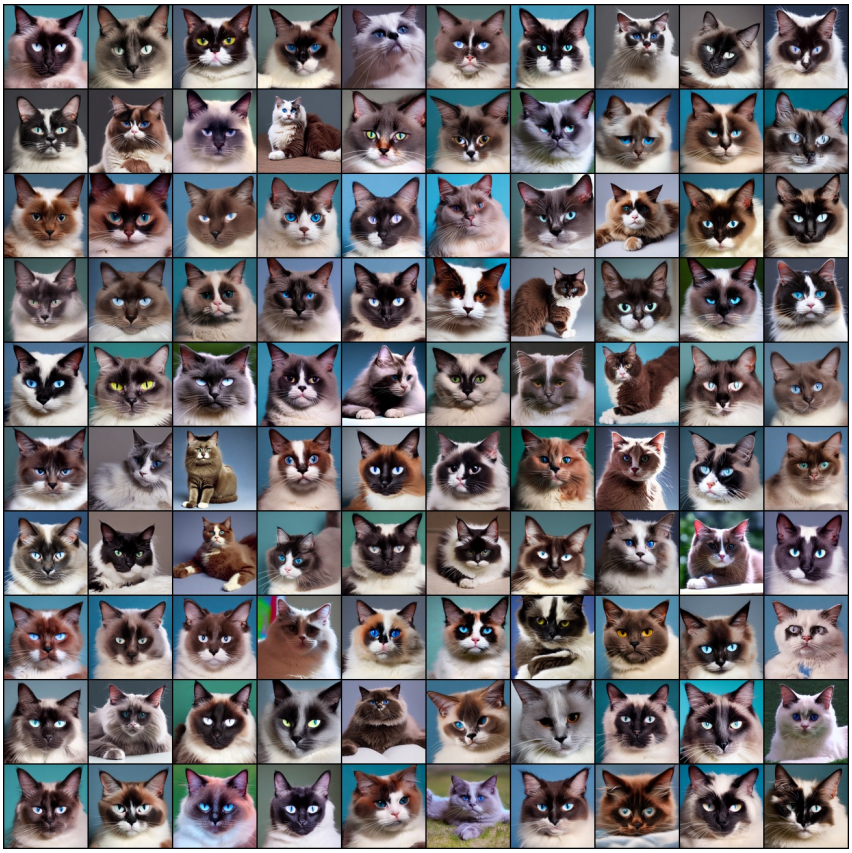


Figure 37: Generated images with enhanced diversity using the approach in § 4.2 on Cat Doudou. The original resolution is 512x512; we resize them to 128x128 here for reducing the file size of the paper.

L CINIC EXPERIMENTS

Given that PE is based on sampling from pre-trained foundation models, a natural baseline is to conditionally sample from the classes closest to the private data. For example, in the experiments of § 5.1.1 (CIFAR10 as private data, ImageNet as the public data), it would be interesting to consider baselines that conditionally sample from the same classes in CIFAR10 (with privacy) from a foundation model trained on ImageNet. However, there could be many ways to implement such an algorithm, and the model and hyper-parameter choices could impact the result. To eliminate the influence of these arbitrariness choices, we conduct the following experiment which *gives an estimate on the best FID such an approach could achieve*.

Experiment setup. We use CINIC10 dataset (Darlow et al., 2018), which contains CIFAR10 images and the filtered images from ImageNet that belong to CIFAR10 categories. We compute the FID between (a) the subset of CINIC10 images that come from ImageNet (i.e., excluding CIFAR10 images from CINIC10 dataset), and (b) CIFAR10 dataset. This is to simulate the best case when the foundation model (1) learns the ImageNet dataset perfectly and (2) is able to draw only the samples from CIFAR10 classes. In practice, the above two assumptions will certainly not hold; especially, achieving (2) would necessarily incur privacy costs as the knowledge of which samples belong to CIFAR10 is assumed to be private. Therefore, this process gives us a lower bound of the FID (i.e., *the best possible FID*) we can get by such a baseline that samples the same classes as CIFAR10 from an ImageNet model using an arbitrary privacy cost.

Results. The FID score from the above is 12.21. We can see from Fig. 4 that PE is able to achieve a smaller FID with ϵ as small as 0.38. This result suggests that PE does a non-trivial job: it can achieve a lower FID score than simply sampling the same classes as CIFAR10 from an ImageNet model.

M MORE ABLATION STUDIES

All ablation studies are conducted by taking the default parameters in unconditional CIFAR10 experiments and modifying one hyperparameter at a time. The default noise multiplier $\sigma = 5 \cdot \sqrt{2}$ and the default threshold $H = 10$.

Lookahead degree. Fig. 38 shows how the lookahead degree k (§ 4) impacts the results. We can see that higher lookahead degrees monotonically improve the FID score. However, the marginal benefits diminish as the lookahead degree goes beyond 8, and a higher lookahead degree increases the required of API calls. Throughout all experiments, we used $k = 8$. This experiment suggests that better results can be obtained with a higher k. In practice, we suggest users set the lookahead degree as the highest value within their computation or API budget.

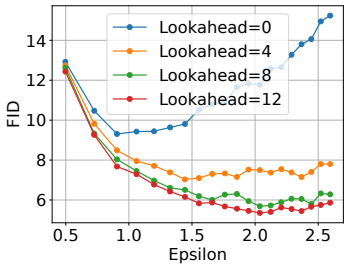


Figure 38: Ablation studies on the lookahead degree k for DP Nearest Neighbors Histogram. Lookahead=0 means lookahead is not used (Eq. (1)).

Population size. Fig. 39 shows how the number of generated samples N_{syn} impacts the results in *non-DP* setting. We can see that, as the number of samples increases, FID score monotonically gets better. This is expected because with more generated samples, there are higher chances to get samples similar to the private data. However, we want to point out that in the DP case, it may not be true, as a large number of samples would flatten the DP Nearest Neighbors Histogram and thus decrease the signal noise ratio.

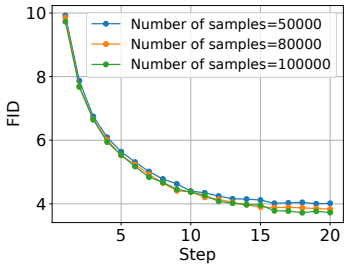


Figure 39: Ablation studies on the number of generated samples N_{syn} in non-DP setting.

Histogram threshold. Fig. 40 shows how the threshold H in DP Nearest Neighbors Histogram impacts the results. We can see that a large threshold results in a faster convergence speed at the beginning. This is because, in the early iterations, many samples are far away from the private data. A larger threshold can effectively remove those bad samples that have a non-zero histogram count due to the added DP noise. However, at a later iteration, the distribution of generated samples is already close to the private data. A large threshold may potentially remove useful samples (e.g., the samples at low-density regions such as classifier boundaries). This may hurt the generated data, as shown in the increasing FID scores at threshold=15. In this paper, we used a fixed threshold across all iterations. These results suggest that an adaptive threshold that gradually decreases might work better.

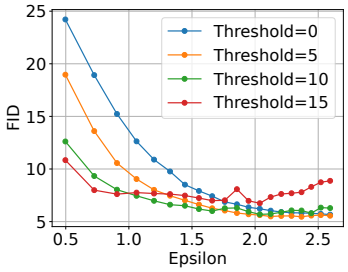


Figure 40: Ablation studies on the threshold H for DP Nearest Neighbors Histogram.

Embedding. Fig. 41 compares the results with inception embedding or CLIP embedding in Eq. (1). The results show that both embedding networks work well, suggesting that PE is not too sensitive to the embedding network. Inception embedding works slightly better. One reason is that the inception network is trained on ImageNet, which is similar to a private dataset (CIFAR10). Therefore, it might be better at capturing the properties of images. Another possible reason is that FID score is calculated using inception embeddings, which might lead to some bias that favors inception embedding.

The number of private samples. In this experiment, we show how the number of private samples N_{priv} impacts the performance of PE. Specifically, on CIFAR10, we vary the number of samples N_{priv} by sub-sampling. For each $N_{\text{priv}} \in \{50000, 20000, 10000, 5000\}$:

- We fix $N_{\text{syn}}/N_{\text{priv}} = 1$ when running PE.

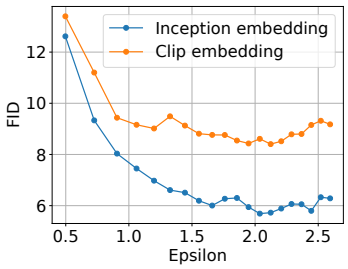


Figure 41: Ablation studies on the embedding network in Eq. (1) to use.

- After PE is done, we use the approach in § 4.2 to augment the number of generated samples back to 50000, and compute the FID between the generated samples and CIFAR10.
- All other hyper-parameters are set the same as the CIFAR10 experiments in Fig. 3.

The results are shown in Table 2, which suggests that larger N_{priv} helps the performance of PE, similar to the observation in DP-SGD (Anil et al., 2021). We hypothesize that the reason is as follows. When we set $N_{\text{syn}}/N_{\text{priv}} = 1$, although the signal-noise ratio in the DP Nearest Neighbors Histogram remains constant, larger N_{syn} does allow the generated samples to explore larger space, and therefore it is more likely to get a sample closer to the private data, which helps the convergence of PE. Our theorem in App. D also shows that increasing the number of variations (which is controlled by N_{syn}) speeds up the convergence.

Number of samples	FID
50000 (original CIFAR10)	7.87
20000 (2.5x smaller)	10.47
10000 (5x smaller)	12.51
5000 (10x smaller)	18.60

Table 2: FID vs. the size of the private dataset in CIFAR10.

N Generating More Samples

We discussed two ways of generating more samples after PE is done: (1) taking the final generated samples from PE and passing them through VARIATION_API to get more samples (§ 4.2), and (2) using a larger N_{syn} when running PE (Fig. 39). While we see that both approaches are promising, they have a key difference: the first approach can be applied post hoc without the need to rerun PE, whereas the second approach requires rerunning PE. This difference would result in a very different number of API calls. More concretely, let's say we want to generate N more samples after PE is done. In the former approach, we simply take the final generated samples from PE and call VARIATION_API N times. In contrast, the latter approach requires rerunning the entire PE process and would need to increase the number of API calls by $N \cdot (k+1)$, as we will need to generate more samples at every iteration.

O Computational Cost Evaluation

We compare the GPU hours of the SOTA DP fine-tuning method (Ghalebikesabi et al., 2023) and PE for generating 50k samples in § 5.1.1. Note that PE is designed to use existing pre-trained models and we do so in all experiments. In contrast, DP fine-tuning methods usually require a careful selection of pre-training datasets and architectures (e.g., (Ghalebikesabi et al., 2023) pre-trained their own diffusion models), which could be costly. Even if we ignore this and only consider the computational cost after the pre-training, *the total computational cost of PE is only 37% of (Ghalebikesabi et al., 2023)* while having better sample quality and downstream classification accuracy (§ 5.1.1). See Fig. 42 for a detailed breakdown of the computational cost. Note that PE's computational cost is mostly on the APIs.

The key takeaway is that even if practitioners want to run the APIs locally (i.e., downloading the foundation models and running the APIs locally without using public API providers), there are still benefits of using PE: (1) The computational cost of PE can be smaller than training-based methods. (2) Implementing and deploying PE are easier because PE only requires blackbox APIs of the models and does not require code modifications inside the models.

Experimental details. In this experiment, we follow the setting in § 5.1.1, where the user has a private dataset of size $N_{\text{priv}} = 50000$ and a pre-trained model (hosted locally, running on GPUs), and wants to generate a total number of $N_{\text{syn}} = 50000$ synthetic data samples. For the DP fine-tuning baseline, the procedure includes (1) DP fine-tuning the pre-trained model, and then (2) using the fine-tuned model to generate 50000 synthetic samples. Therefore, we break the time into these two parts. For PE, the procedure includes running PE with $N_{\text{syn}} = 50000$. PE includes the steps

of RANDOM_API, VARIATION_API, nearest neighbor search, and feature extraction. Therefore, we break down the time of PE into these four parts.

In the above evaluation, we did not consider the time of generating more samples beyond $N_{\text{syn}} = 50000$, and we discuss its effect here. To generate more samples beyond $N_{\text{syn}} = 50000$ samples, the DP fine-tuning method can directly use the fine-tuned model to generate more samples without more fine-tuning. PE can use the approach in § 4.2, where we pass the generated N_{syn} samples through VARIATION_API to generate more samples (what we did in § 5.1.3; see results thereof). Note that, for the same size of the model, the same number of generated samples, the same diffusion sampler, and the same total number of denoising steps, *the running time of PE to generate more samples is smaller than the DP fine-tuning baseline.* The reason is as follows. Let's say we use a diffusion sampler with 100 steps in total. In the DP fine-tuning baseline, generated samples must take all 100 steps (i.e., starting from Gaussian noise, passing the image through the diffusion model 100 times iteratively to get the final generated samples). For PE, VARIATION_API is implemented by SDEdit (Meng et al., 2021) (see App. I), where we add noise to input images and let the diffusion model denoise them *starting from the middle of the diffusion process* (e.g., adding Gaussian noise to the image, then passing the image through the diffusion model starting from 20th denoising step for all the rest 80 steps iteratively to get the final generated samples). In other words, each generated does not need to go through all 100 steps, but only a fraction of the steps (80 steps in the above example).

To ensure a fair comparison, we estimate the runtime of both algorithms using 1 NVIDIA V100 32GB GPU.

To evaluate the computational cost of (Ghalebikesabi et al., 2023), we take the open-source diffusion model implementation from (Dhariwal & Nichol, 2021)[20] and modify the hyper-parameters according to (Ghalebikesabi et al., 2023). We obtain a model with 79.9M parameters, slightly smaller than the one reported in (Ghalebikesabi et al., 2023) (80.4M). This difference might be due to other implementation details that are not mentioned in (Ghalebikesabi et al., 2023). To implement DP training, we utilize Opacus library (Yousefpour et al., 2021). To evaluate the fine-tuning cost, we use `torch.cuda.Event` instrumented before and after the core logic of forward and backward pass, ignoring other factors such as data loading time. We estimate the total runtime based on the mean runtime of 10 batches after 10 batches of warmup. We do not implement augmentation multiplicity with data and timestep (Ghalebikesabi et al., 2023); instead, we use multiplicity=1 (i.e., a vanilla diffusion model), and multiply the estimated runtime by 128, the multiplicity used in (Ghalebikesabi et al., 2023). To evaluate the generation cost, we use `torch.cuda.Event` instrumented before and after the core logic of sampling. We estimate the total runtime based on the mean runtime of 10 batches after 1 batch of warmup.

To evaluate the computational cost of our PE, we use a similar method: we use `torch.cuda.Event` instrumented before and after the core logic of each component of our algorithm that involves GPU computation. RANDOM_API and VARIATION_API are estimated based on the mean runtime of 10 batches after 1 batch of warmup. Feature extraction is estimated based on the mean runtime of 90 batches after 10 batch of warmup. The nearest neighbor search is estimated based on 1 run of the full search. We use faiss library[21] for nearest neighbor search. Its implementation is very efficient so its computation time is negligible compared with the total time.

[20] https://github.com/openai/guided-diffusion
[21] https://github.com/facebookresearch/faiss

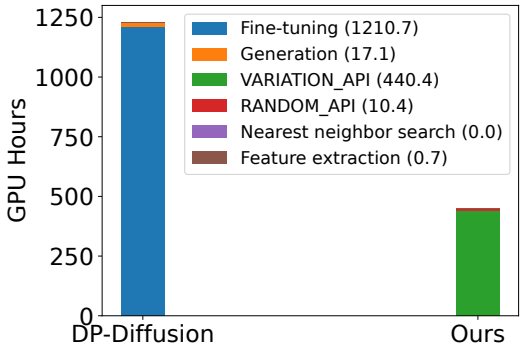


Figure 42: GPU hours (on 1 NVIDIA V100 32GB) required to obtain the samples for § 5.1.1 with DP-Diffusion (Ghalebikesabi et al., 2023) and ours. The legend denotes the steps and the GPU hours; (Ghalebikesabi et al., 2023) contains the fine-tuning and generation steps, whereas ours contains the other steps.