# DISTRIBUTIONAL DATASET DISTILLATION WITH SUB-TASK DECOMPOSITION

**Tian Qin** *
Harvard University
Cambridge, MA

**Zhiwei Deng**
Google DeepMind
Mountain View, CA

**David Alvarez-Melis**
Harvard University & MSR
Cambridge, MA

## ABSTRACT

What does a neural network learn when training from a task-specific dataset? Synthesizing this knowledge is the central idea behind Dataset Distillation, which recent work has shown can be used to compress a large dataset into a small set of input-label pairs (*prototypes*) that capture essential aspects of the original dataset. In this paper, we make the key observation that existing methods that distill into explicit prototypes are often suboptimal, incurring in unexpected storage costs from distilled labels. In response, we propose *Distributional Dataset Distillation* (D3), which encodes the data using minimal sufficient per-class statistics paired with a decoder, allowing for distillation into a compact distributional representation that is more memory-efficient than prototype-based methods. To scale up the process of learning these representations, we propose *Federated distillation*, which decomposes the dataset into subsets, distills them in parallel using sub-task experts, and then re-aggregates them. We thoroughly evaluate our algorithm using a multi-faceted metric, showing that our method achieves state-of-the-art results on TinyImageNet and ImageNet-1K. Specifically, we outperform the prior art by $6.9\%$ on ImageNet-1K under the equivalence of 2 images per class budget.

## 1 INTRODUCTION

Large datasets such as ImageNet (Deng et al., 2009) can be used for a variety of purposes, ranging from image classification, single-object localization to generative tasks. If one only needs to accomplish one of those tasks, say image classification, can we synthesize only relevant information in the data and thus achieve compression? The goal of data distillation, first introduced by Wang et al. (2018), is to answer this question: how to 'condense' a dataset into a smaller (synthetic) counterpart, such that training on this distilled dataset achieves performance comparable to training on the the original dataset. Since its inception, this problem has garnered significant attention due to its obvious implications for data storage efficiency, faster model training, and democratization of large-scale model training. It also holds the promise of speeding up downstream applications such as neural architecture search, approximate nearest neighbor retrieval, and knowledge distillation, all of which often require data-hungry methods (Sachdeva & McAuley, 2023). Moreover, data distillation has emerged as a promising approach for continual learning (Rosasco et al., 2021) and differential privacy (Dong et al., 2022), often outperforming bespoke differentially-private data generators both in terms of performance and privacy, and allowing for private medical data sharing (Li et al., 2022).

Most current state-of-the-art data distillation methods produce *synthetic protoypes*: a small subset of learnt (input, label) pairs that capture the most 'salient' (in terms of their impact on classifier performance) aspects of the original dataset. These prototypes are often defined in the original input (e.g., image) space (Wang et al. (2018; 2022)). Recently, some work (Deng & Russakovsky, 2022; Zhao & Bilen, 2022; Lee et al., 2022) propose to distill images into a latent space, and use a decoder to map latent codes back to the input (image) space. Overall, dataset distillation methods have achieved remarkable success in producing much smaller datasets —typically measured in terms of Images (or Prototypes) Per Class (IPC)— with limited loss of downstream model performance. While early methods suffered from limited scalability, recent ones have managed to scale to large datasets like ImageNet-1K or even ImageNet-21K (Yin & Shen, 2023; Yin et al., 2023; Liu et al.,

*Correspondence to: Tian Qin ⟨tqin@g.harvard.edu⟩.

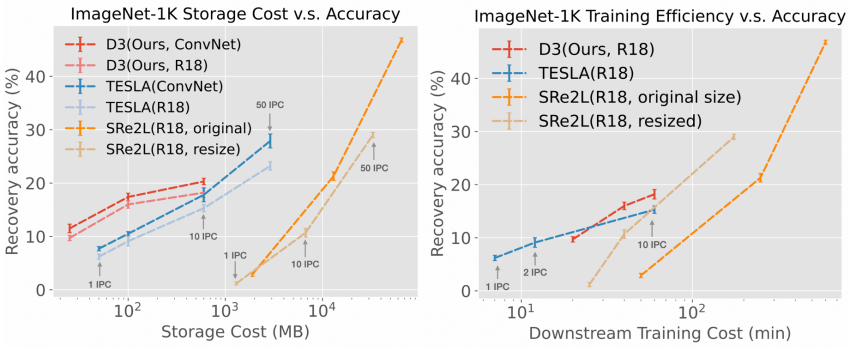


Figure 1: **Three-dimensional evaluation on methods that scale to ImageNet-1K.** *Left:* Recovery accuracy vs. storage trade-off comparison for our (D3) and other methods on resized ($64 \times 64 \times 3$) ImageNet-1K. Our method achieves SOTA performance at small memory cost regime. *Right:* Accuracy vs. downstream task training cost on resized ImagNet-1K.

2023). For example, SRe$^2$L (Liu et al., 2022) achieved a $\sim$100$\times$ IPC reduction on ImageNet-1K and recovered $\sim$77% of the classification accuracy[1].

Although encouraging, we will show that these results tell an incomplete story. When considering the *total storage* (e.g., disk space used to store all necessary distillation outputs) and the runtime needed to train new models on the distilled data, the efficacy of these methods is much more subdued. Beyond the prototypes, some of these methods output other artifacts that are necessary for downstream use but whose memory footprint is rarely reported. These include soft labels (often multiple per prototype) and augmentation parameters used (Zhou et al., 2022b; Yin & Shen, 2023; Yin et al., 2023). The use of distilled labels are crucial (Figure 7) but incur a storage cost that is not captured by IPC (Table 7, Appendix A.1). Once we take into account the storage cost of these artifacts, the true compression rate of such methods is much lower than implied by the IPC metric (Figure 1, left). On the other hand, decoding/generation/augmentation procedures often translate into additional post-distillation training time (Figure 1, right). In light of these observations, we argue that IPC as a metric of distillation is incomplete, and that the methods that have been developed to optimize it should be revisited with a more comprehensive set of evaluation metrics.

In response to the above observation, we propose a new dataset distillation method with efficient compression properties, not in terms of dataset cardinality (i.e., IPC), but directly in terms of storage size and downstream model training time. Distilling into the latent space not only allows a more compact representation of the data by sharing inter-class mutual information in decoder parameters, but also offers finer-grained control on compression than working directly with prototypes (e.g., by varying number of latent codes per class, latent dimensions and the decoder size). We challenge the conventional approach of distilling into a finite set of (latent) prototypes and propose to cast the problem into a *distributional* one: finding a synthetic probability distribution which can be sampled to produce training data for downstream tasks. This Distributional Dataset Distillation (D3) approach yields an efficient representation of distilled data without incurring much additional computation costs on downstream tasks.

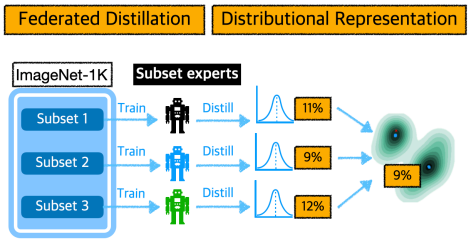


Figure 2: **Illustration of Federated Distillation and Distributional Representation** We decompose large datasets into subtasks and distill each subset into distributions using locally trained experts. Distributions distilled on subtasks generalize well to the full task.

To scale our method efficiently to large datasets such as ImageNet-1K, we propose a simple-yet-effective *federated distillation* scheme that parallelizes the distillation process (Figure 2). Instead

[1]SRe$^2$L used ResNet18 as the teacher model, which achieved 69.8% classification accuracy from full ImageNet-1K training. SRe2L's distilled data with 10 IPC achieved 46.2% classification accuracy

of directly distilling the entire dataset, we divide the full classification tasks into subtasks, where each task only classifies a subset of all classes. Data distillation is performed on subtasks, using local experts trained on subtasks. We then aggregate the locally distilled datasets to form the distilled data for the full task. We show that data distilled on subtasks generalize well to the full task, which ensures the good performance of our federated distillation process. Using the distributional representation and federated distillation, we achieve SOTA performance on ImageNet-1K as measured by storage cost.

Our contributions can be summarized as follows:

- We show that state-of-the-art prototype-based data distillation methods yield unexpectedly high storage costs and post-distillation training times, an under-reported phenomenon that is not captured by commonly-used compression metrics (e.g., IPC, the *number* of distilled items per class). The large storage cost and training time could hinder the usability of these methods in practice.
- We propose a novel distillation framework with smaller memory footprint that distills datasets into *distributions*, extending recent methods that distill into a latent space to now operate on (latent) distributions. We show this method matches or outperforms state-of-the-art distillation methods in terms of prediction accuracy on various datasets (e.g., TinyImageNet, ImageNet-1K), with smaller storage costs.
- We propose a simple-yet-effective federated distillation strategy that allows distillation training process to be parallelized, and which has general applicability beyond our specific method.
- In response to our observations above, we propose new evaluation protocols and metrics for dataset distillation methods that more accurately characterize the extent of 'distillation', and compare existing work and our work along these axes. [2]

## 2 Related Work

We focus our discussion of prior work on the lines that are most closely related to ours, but note that methods with similar goals have been developed in the context of statistical sample compression (Winter, 2002; Dwivedi & Mackey, 2021) and core-set selection (Mirzasoleiman et al., 2020; Zhou et al., 2022a).

**Optimization Methods** Wang et al. (2018) originally approached the distillation problem as a bi-level optimization task, which is computationally intensive. To tackle the computation challenge, many work has proposed proxy training objectives to simplify the distillation process. Nguyen et al. (2020) leveraged NTK-based algorithms to solve the inner optimization in closed form. Zhao et al. (2020) proposed gradient matching to avoid the unrolling of the inner-loop and make the distillation process more efficient. Further improvements on single-iteration gradient matching also include (Lee et al., 2022; Zhao & Bilen, 2021). Matching training trajectories (MTT) was proposed by (Cazenavette et al., 2023), claiming that matching long-range training dynamics provides further improvements on single-iteration gradient matching. Cui et al. (2022) proposed TESLA as a scalable alternative to the original MTT method. Du et al. (2022) proposed a variant that uses "flat" trajectory matching to further improve trajectory-matching based methods. Distribution Matching (DM), proposed by (Zhao & Bilen, 2023), seek to minimize the Maximum Mean Discrepancy (MMD) between original and distilled dataset samples. Further refinement on the method includes Wang et al. (2022); Zhou et al. (2022b). Specifically, Neural Feature Regression with Pooling (FRePo) (Zhou et al., 2022b) addressed the memory-concern with a pooling strategy for distribution matching based method.

Recently, SRe$^2$L (Yin et al., 2023) proposed to decouple the expensive bi-level optimization and used a three step procedure - first, produce feature mapping; second, generate distilled images; and third generate soft labels. Follow-ups such as Liu et al. (2023) and Yin & Shen (2023) brought further improvements on the method. This line of work has achieved impressive performance on large datasets such ImageNet-1K and even ImageNet-21K.

[2]Code for all experiments is available here: `https://github.com/sunnytqin/D3`

**Representing Distilled Dataset** In contrast to all methods listed so far, a new line of work proposed to distill data into the latent space (Deng & Russakovsky, 2022; Liu et al., 2022; Lee et al., 2022; Cazenavette et al., 2023; Zhao & Bilen, 2022). This line of work proposes to learn the latent code and use decoder(s) to map the latent code back into training images. Zhao & Bilen (2022); Cazenavette et al. (2023) leveraged pre-trained GANs as the decoder such that only latent code needed to be learned during distillation. (Deng & Russakovsky, 2022; Lee et al., 2022; Liu et al., 2022) trained both latent codes and decoders during the distillation process. Our work is mostly similar to IT-GAN (Zhao & Bilen, 2022) in using a generative model to represent distilled data. However, we model the *prototypes themselves* as distributions, allowing for e.g., unlimited sampling from them, and leading to more diverse generation. IT-GAN (Zhao & Bilen, 2022) only showed the feasibility on CIFAR-10 while we scale the idea to TinyImageNet, and ImageNet-1K. Furthermore, we show that by using a distributional framework and a generator trained from scratch, one can achieve a more compact representation of data.

# 3 METHODOLOGY

## 3.1 THREE-DIMENSIONAL EVALUATION

The most important aspect of evaluating data distillation methods is the trade off between the memory footprint (i.e., how large is the distilled dataset) and the recovery accuracy (i.e., can models trained the compressed data achieve comparable performance compared to the original dataset). When it was first proposed by Wang et al. (2018), the distillation task was restricted to finding a set of images $\{\boldsymbol{s}_i\}_{i=1}^{n}$. Same number of prototypes were used for each class along with hard labels. Since storing the corresponding label incurred a trivial cost, IPC was sufficient to capture the distilled dataset size in early works. However, two recent trends brought innovations to different ways to store information in the distilled dataset. As a result, the IPC metric no longer reflects the trade-off between storage and recovery accuracy.

**Storage Cost** Instead of distilling into pixels, many recent works (Deng & Russakovsky, 2022; Lee et al., 2022; Liu et al., 2022) distill data into a latent space $\mathcal{Z}$ and represent each prototype as a latent code $z \in \mathcal{Z}$. One or multiple decoders are used to map the latent code into original space during downstream training, by trading memory with compute. On the other hand, images are not the only way one can store information in the distilled dataset. Many methods that scale to ImageNet-1K leverage distilled labels as an additional way to store information. TESLA (Yin et al., 2023), and FRePo (Zhou et al., 2022b) distill prototypes into pixel space and assign one unique soft label to each prototype. Compared to hard labels, storing softmax values incurs a small but non-trivial storage cost. On the other hand, SRe$^2$L and its follow-up work (Yin et al., 2023; Liu et al., 2023; Yin & Shen, 2023) take a slightly different approach by assigning *multiple* distilled labels to each prototype. For each prototype, different distilled labels correspond to variants of the prototype by applying augmentations. As a result, these work (Yin et al., 2023; Liu et al., 2023; Yin & Shen, 2023) require the augmentation parameters stored along with the corresponding to distilled labels.

**Downstream Training Cost** When training models on the distilled data, using soft labels instead of hard labels, decoding latent codes on-the-fly, and applying augmentations to prototypes all bring additional computation cost during downstream training. Therefore, in addition to storage cost, we also propose to look at the the wall clock time to train models on the distilled data, which we abbreviate as downstream training cost. This training cost can help us gain insights into the memory versus compute trade-off between different distillation methods. However, the primary objective of dataset distillation is to achieve information compression by saving only relevant features needed for a certain task, storage cost should be the primary metric for evaluation and downstream training cost should be a secondary metric. Despite being a secondary metric, downstream task training cost is still relevant because if training models on the distilled data takes too long, the distilled dataset may have limited usability on applications such as continual learning or neural architecture search.

We propose a more comprehensive evaluation process based on the following three metrics:

(i) Total storage cost: being distilled images, prototypes, latent codes, soft labels, augmentations, and/or decoders

(ii) Downstream training cost: wall clock time it takes to train models on the distilled data

(iii) Recovery accuracy: accuracy achieved by model trained on the distilled data

Table 1: **Tiny ImageNet distilled and evaluated on ConvNet** Storage cost is measured in MB and in parenthesis, we annotate the equivalence if storing only images (measured using IPC). In our most compact setting, the storage cost to store the distilled distribution averages to storing less than 1 IPC. N/A: indicates the distillation size is smaller than the minimum size the method can distill.

| Storage Cost (MB) | Random | DM | MTT | LinBa | KFS | FrePo | D3(ours) |
|---|---|---|---|---|---|---|---|
| 4 (∼0.5IPC) | N/A | N/A | N/A | N/A | N/A | N/A | **24.6 (0.2)** |
| 10 (∼1IPC) | 1.6 (0.1) | 3.9 (0.2) | 8.8 (0.3) | 16.0 (0.7) | 22.7 (0.2) | 15.4 (0.3) | **26.0 (0.4)** |
| 100 (∼10IPC) | 6.2 (0.2) | 12.9 (0.4) | 23.2 (0.2) | | 27.8 (0.2) | 25.4 (0.2) | **30.5 (0.3)** |

Table 2: **Tiny ImageNet distilled and evaluated on different architectures** We use the distribution distilled under the 100MB ($\sim$ 10 IPC) storage cost budget. Our method generalizes well to different architectures.

| ConvNet (self) | AlexNet | ResNet18 | VGG11 | ViT |
|---|---|---|---|---|
| 30.5 (0.3) | 22.6 (0.5) | 25.7 (0.5) | 27.0 (0.2) | 15.1 (0.7) |

We perform the comprehensive evaluation on TESLA (Cui et al., 2022), SRe$^2$L (Yin et al., 2023) and D3 (ours) on ImageNet-1K (resized). TESLA (Cui et al., 2022) represents SOTA results among existing methods that distill directly into image spaces along with soft labels. SRe$^2$L (Yin et al., 2023) represents results among existing methods that distill into images, augmentations and soft labels. Finally, our work (D3) represents results that distill into latent (distributions). Using the new metric, we observe a different landscape that is not captured by IPC, as shown in Figure 1. Bi-level optimization-based methods (TESLA and ours) excel at small-scale, extremely efficient dataset distillation while decoupled methods (SRe$^2$L) achieves superior performance at the cost of a larger storage footprint and longer downstream training time. In Appendix A.1, we list further details on the exact storage cost breakdown and a discussion on the storage cost is measured.

### 3.2 Distilling into distributions

The motivation behind distilling into the latent space is to achieve further compression in data storage (Deng & Russakovsky, 2022; Lee et al., 2022; Liu et al., 2022). Instead of images, the final distilled dataset consists of one or multiple latent codes for each class, along with one or multiple decoder(s). Decoders map those latent codes back into the pixel space. Formally, we denote the $i$-th latent code for image class $c$ as $\boldsymbol{z}_c^i$, and denote decoder(s) as $g_\theta$, where $\theta$ are parameters for the decoder. The distilled dataset $\mathcal{S}$ can be expressed as:

$$\mathcal{S} = \{(\boldsymbol{x}_c^i, y_c^i) : (g_\theta(\boldsymbol{z}_c^i), c)\}_{c\in[1,C]}^{i\in[1,\text{IPC}]}.$$

The above formulation indicates a *one-to-one* correspondence between latent code and output image - namely, a deterministic data generation process. In this work, we propose to achieve an even more efficient way to represent dataset by generalizing the idea of latent codes into latent distributions. Instead of a deterministic data generation process, we now have a probabilistic one where one can repeatedly sample from the latent distribution and pass into the decoder to generate images.

To represent the latent distribution, we borrow ideas from Deep Latent Variable models (Kingma & Welling, 2019; 2013) and assume the latent distribution to be Gaussian: $p(\boldsymbol{z}|c) \sim \mathcal{N}(\mu_c, \Sigma_c)$. During the data distillation process, we learn the parameters $\mu_c$ and $\Sigma_c$ for those Gaussian priors, as well as decoder parameters $\theta$. During downstream training, sample data is generated in an "online" fashion by sampling from the latent distribution $\mathcal{N}(\mu_c, \Sigma_c)$ at each epoch. See Appendix A.2 for a detailed description on distributional representation.

### 3.3 Federated Distillation

The challenge to scale data distillation methods to ImageNet-1K comes from the significant memory and computation costs (Zhou et al., 2022b; Yin et al., 2023; Cui et al., 2022). Our method D3 also suffers from the same challenge. To resolve the scaling issue, we propose to use a federated distillation strategy. First, we divide the datasets into $k$ subsets in the class space. Each subset only contains $C/k$ classes, where $C$ denotes the total number of classes in the full set. Then, we perform data distillation independently on each subset. Note that in this step, we train local experts for each

sub-task and optimize the distill data on those subtasks, which is simpler than the full classification task. Finally, the distill subsets are aggregated to form the distilled dataset for the full task. For an illustration of our federated distillation strategy, see Figure 2.

Since each subset has only been trained by local experts for each subtask (i.e, classify only $C/k$ classes as opposed to all $C$ classes), one certainly would expect the federated strategy to yield sub-optimal results compared to directly distilling on the full dataset. In section 4.3 we confirm such intuition. However, we observe that dataset distilled on those simpler subtasks transfers relatively well to the full task. This nice generalization property allows us to distill ImageNet-1K in a highly parallelized fashion while achieving SOTA results.

## 4 Experiments

In pursuit of impartial comparisons with existing data distillation methodologies, we align all our design choices with existing work. We use ConvNet for data distillation on all datasets. We evaluate the recovery accuracy on five randomly initialized neural networks and report mean and standard deviation. We provide a detailed description of datasets, experiment setup and hyper-parameters in Appendix C. We compare our methods on competitive baselines that distill into pixel space, including MTT (Cazenavette et al., 2022), TESLA (Cui et al., 2022) , concurrent work DataDAM (Sajedi et al., 2023), FRePo (Zhou et al., 2022b), FTD (Du et al., 2022), and DM (Zhao & Bilen, 2023). We also compare our method on competitive baselines that distill into latent space, including LinBa (Deng & Russakovsky, 2022) and KFS (Lee et al., 2022).

Figure 3: **Visualization of distilled mean (col 1) and variations (col 2 onwards) for four ImageNet-1K classes**

### 4.1 Quantitative Results

We apply the federated distillation strategy on **ImageNet-1K** by breaking down the dataset into 2 and 5 sub-tasks. To scale up the distilled distributions, we use 1, 2 and 10 latent priors per class and scale up decoder sizes accordingly. We report results on both ConvNetD4 and ResNet18 (cross-architecture generalization) with our three-dimensional evaluation metric in Figure 1 and a tabular version can be found in Appendix A.1. Our method outperforms TESLA (Cui et al., 2022), DataDAM (Sajedi et al., 2023) and FRePo (Zhou et al., 2022b) on ConvNet under 100MB and 500MB storage budget. Furthermore, our method can also distill a distribution under 25MB (0.5IPC) storage budget, which outperforms existing work under 50MB (1PC) storage budget.

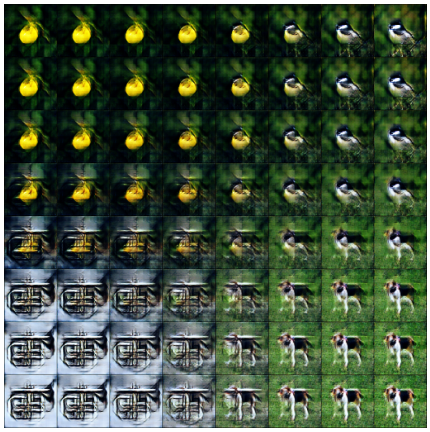

Figure 4: **Visualization of the latent Gaussian space by interpolating priors for four classes from ImageNet-1K**

Our method distills a more compact dataset through distributional representation, however, when we evaluate on the downstream task training cost, we can see that that our compact representation comes at a (small) compute cost. On average, models trained on our distilled distribution takes more training iterations to converge compared to fixed-output methods and generating images on-the-fly costs additional small but non-negligible compute time. We report **TinyImageNet** results in Table 1, and corresponding cross-architecture results in Table 2. We use 2, 5, and 10 latent priors for three storage costs and a larger decoder for the last one. Similar to ImageNet-1K, our method can distill distribution that achieves SOTA performance with small storage costs and generalize well to different architectures. Additionally, we report CIFAR-100, CIFAR-10 and the two ImageNet subsets in Appendix D. In Appendix C, we include a full list of decoder hyper-parameters and exact storage cost for all the experiments listed above.

### 4.2 Latent Space Analysis

We impose a Gaussian structure to the latent distribution space, and the motivation behind distillation is to save only relevant information from a dataset for a specific task (image classification, in our case). Through the distilled distribution, we can visualize the Gaussian space to reach some qualitative understanding on the 'salient' features that are essential for image classification task. In Figure 3, we visualize the prototype distribution of distilled ImageNet-1K under 500MB ($\sim$ 10 IPC) storage budget. Specifically, we visualize the average (first column) of four randomly chosen classes and their corresponding variations (second column onwards). Qualitatively, we observe that the typical (i.e., mean) sample is more interpretable and higher quality compared to its variations. We also visualize four randomly chosen classes (four corners) and the inter-class "distributions" by linearly interpolate the Gaussian space between (the rest), shown in Figure 4.

### 4.3 Federated Distillation

**Ability to task-generalize** When we perform federated distillation, we are essentially distilling datasets on simpler subtasks (i.e., classification on fewer classes). To understand the extent to which breaking down distillation tasks can impact the overall performance on the distillation process, we conduct the following experiments on TinyImageNet. First, we perform federated data distillation by dividing the dataset into two subsets, first one containing the first hundred classes and the second one containing the rest (second hundred classes). The full distilled dataset distribution is obtained from aggregating the two distributions. To form a comparison, we perform dataset distillation directly on the full dataset using the same-sized decoder, and the same settings for the latent prior distributions (i.e., same dimension for the latent Gaussian distribution, and same number of latent priors per class).

Table 3: **Federated distillation compared to full task distillation on TinyImageNet** row indicates the dataset being distilled on and column indicates the dataset being evaluated on.

| | (Subset 1, Subset 2) | Full |
|---|---|---|
| (Subset 1, Subset 2) | (27.6 (0.6), 27.2 (0.5)) | 21.9 (0.6) |
| Full | (32.2 (0.3), 33.4 (0.4)) | 24.6 (0.3) |

Experiment results are reported in Table 4. Rows indicates the dataset that distillations are performed on, and the column indicates the dataset that distilled distributions are evaluates on. First row summarizes the federated distillation outcome: each subtask achieves $\sim 27\%$ recovery accuracy. We then evaluate the aggregated distributions on the full task, which achieves $21.9\%$ recovery accuracy (row 1, col 2). In comparison, the second row summarizes results for the non-federated counterpart: directly distilling on the full set achieved higher accuracy $24.6\%$ than the federated outcome. In this experiment, we observe that the federated distillation only slightly underperforms the non-federated version. In addition, we notice that the data distribution distilled on the full dataset outperforms the federated counterpart (row 2, col 1) when evaluated on the subtask.

**Impact factors** To understand whether the ability of distilled data to task generalize is sensitive to the distillation training objective and/or the use of distributional outcome, we further experiment on TinyImageNet using the same set-up as above. In this set of experiments, we perform data distillation using different training objectives, using distributional or fixed outputs to distill each of the TinyImageNet subsets, shown in Table 4.

In Section 5.2, we perform a more detailed ablation study on the impact of training objectives and distributional representation on distillation outcomes. Here, we are only interested in examining whether those factors impact the ability for the distilled data to task generalize. For fixed outputs, we repurpose latent distributional priors as latent codes and simply distilling only the mean. Table 4 shows that the relative transfer performance is not sensitive to different

Table 4: **Federated distillation on TinyImageNet using different training objectives, and using distributional or fixed representation** *Subset 1, Subset 2* is recovery accuracy on the subtask and *full* is recovery accuracy by aggregating the two distill datasets and evaluating on the full set. Chg$\%$ computed by `full/avg(subset 1, subset 2)`.

| Loss Term | Distributional | Subset 1($\%$) | Subset 2($\%$) | Full($\%$) | Chg$\%$ |
|---|---|---|---|---|---|
| MTT | Yes | 12.7 (0.2) | 18.4 (0.4) | 13.4 (0.4) | 86$\%$ |
| MMD | Yes | 25.4 (0.4) | 26.1 (0.5) | 20.4 (0.1) | 80$\%$ |
| Both | Yes | 27.6 (0.3) | 27.2 (0.5) | 21.9 (0.6) | 80$\%$ |
| Both | No (5 IPC) | 10.28 (0.5) | 12.38 (0.2) | 7.5 (0.5) | 66$\%$ |
| Both | No (10 IPC) | 19.58 (0.4) | 17.68 (0.5) | 13.9 (0.2) | 74$\%$ |
| Both | No (20 IPC) | 27.5 (0.3) | 25.32 (0.4) | 20.5 (0.6) | 78$\%$ |

training objectives. However, when we restrict ourselves to fixed outputs (i.e., without distributional representation), the relative task transfer ability suffered by a non-trivial amount. The performance drop is most evident when we allow fewer fixed latent codes per class. However, as we increased the number of fixed images, the task transfer ability converge to the distributional version. This observation indicates that our federated distillation scheme could potentially be generalized to other data distillation methods.

**Number of Subtasks** To understand to what extent the number of subtasks negatively impacts the federated distillation strategy, we experiment three division sizes on ImageNet-1K: 10 sub-tasks (100 classes for each task), 5 sub-tasks (200 classes for each task) and 2 sub-tasks (500 classes for each task). Table 5 shows that the distillation quality increase monotonically as we decrease the number of subtasks. This observation provides a straightforward guideline for subtask selection in practice: one should divide into as few tasks as memory and computation allows to achieve the best distillation outcome. For fair comparisons, we use two different decoder sizes for the first two and a large one for the later.[3] For the same decoder size, we keep the hyper-parameters same for the latent distribution (same latent dimension, and same number of priors per class).

Table 5: **Performance of federated distillation with different subtask sizes on ImageNet-1K.**

| Subset size | # Tasks | Decoder Size | Accuracy |
|---|---|---|---|
| 100 | 10 | S | 9.7 (0.2) |
| 100 | 10 | M | 9.6 (0.3) |
| 200 | 5 | S | 10.6 (0.3) |
| 200 | 5 | M | 13.8 (0.5) |
| 500 | 2 | L | 14.7 (0.5) |

# 5 Ablation Study

## 5.1 Distributional Outcome

**On training accuracy** Using the same training objective and using the same decoder setting, we experiment disabling the distributional outcome. By allowing a distributional representation, a more diverse set of samples are generated on-the-fly. As a result, the distilled distribution reaches a higher distillation quality compared to its non-distributional counterpart. We experiment on CIFAR-10 and TinyImageNet, shown in Figure 6. **On prototype quality** We also visualize samples from distilled TinyImageNet outcomes above (with and without distributional representation) in Figure 5. We observe that the distributional objective makes the distilled data more interpretable.

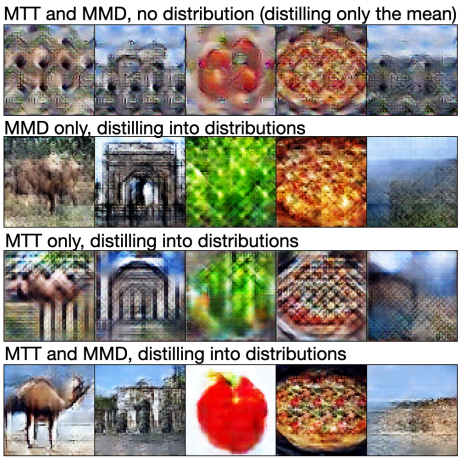


Figure 5: **Visualization of distilled samples from five classes using different training objectives and fixed or distributional representation on TinyImageNet.** For distributional outcomes we visualize the mean.

## 5.2 Loss Term Contribution

To study the effect of combing two objectives, we perform distillation on CIFAR-10 and TinyImageNet with either loss terms while keeping the decoder hyper-parameters constant, results shown in Figure 6.

[3] smaller and medium sized decoder failed to converge on 500-class subtask

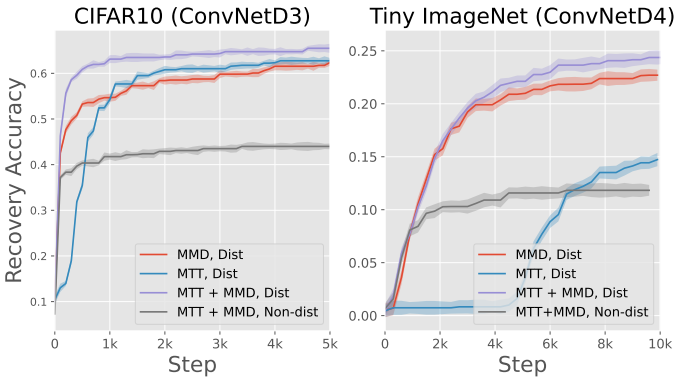


Figure 6: **Ablation study on dual training objective and distributional representation** Both distributional representation and dual training objective are essential for the performance of our method.

The dual training objective yields superior performance than using either one stand alone. However, using the MMD or MTT objective alone could already achieve good results, depending on the dataset. While performing distillation, we observe that the dual objective consistently outperform using one alone. From visualizations in Figure 5, the combined objective yields more interpretable results than using either alone. While it might be possible that one can further simplify the training objective by only using one of them, we keep the dual objective based on the above observations.

### 5.3 Distilled Labels

Similar to prior works, we also find that the use of distilled labels brings additional benefit to the dataset distillation. We use softmax values generated by pretrained experts as distilled labels, an intuitive strategy already used by Cui et al. (2022); Zhou et al. (2022b). However, since we distill into distributions, we use the mean of every latent prior to generate soft labels. In practice, we find that the distilled labels for each mean work well even for randomly generated samples. Different from existing work, our distilled distribution is more robust against having only hard labels - our method significantly outperforms TESLA Cui et al. (2022) and FRePo Zhou et al. (2022b) when only hard labels are used, shown in Figure 7.

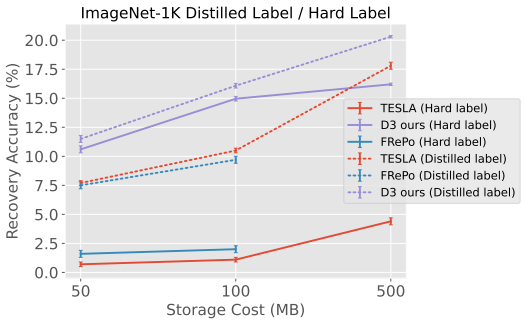


Figure 7: **ImageNet-1K recovery accuracy using distilled labels or hard labels** Our method still performs well even without using distilled labels.

## 6 Conclusion

In this paper, we first made a key observation that existing work distilling into explicit prototypes *and* distilled labels incurred unexpected storage cost and post-distillation training time, both of which could not be captured the conventional metric used dataset distillation. We proposed to evaluate distillation methods on a three-dimensional metric that captures the total storage cost and the test-time runtime efficiency.

We also proposed *Distributional Dataset Distillation*, which encodes the data using minimal sufficient per-class statistics and a decoder, resulting in a distileld data distribution that is a more memory-efficient representation of the training data. To scale up the process of data distillation, we proposed a federated distillation strategy, which can have broader applications on other data distillation methods. For future work, we aim to scale our results to larger datasets and to higher distillation quality.

## REFERENCES


Mikołaj Bińkowski, Danica J Sutherland, Michael Arbel, and Arthur Gretton. Demystifying MMD GANs. January 2018. arXiv:1801.01401.

George Cazenavette, Tongzhou Wang, Antonio Torralba, Alexei A Efros, and Jun-Yan Zhu. Dataset distillation by matching training trajectories. March 2022. arXiv:2203.11932.

George Cazenavette, Tongzhou Wang, Antonio Torralba, Alexei A Efros, and Jun-Yan Zhu. Generalizing dataset distillation via deep generative prior. pp. 3739–3748, May 2023. arXiv:2305.01649.

Justin Cui, Ruochen Wang, Si Si, and Cho-Jui Hsieh. Scaling up dataset distillation to ImageNet-1K with constant memory. November 2022. arXiv:2211.10586.

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, 2009. doi: 10.1109/CVPR.2009.5206848.

Zhiwei Deng and Olga Russakovsky. Remember the past: Distilling datasets into addressable memories for neural networks. pp. 34391–34404, June 2022. arXiv:2206.02916.

Tian Dong, Bo Zhao, and Lingjuan Lyu. Privacy for free: How does dataset condensation help privacy? June 2022. arXiv:2206.00240.

Jiawei Du, Yidi Jiang, Vincent Y F Tan, Joey Tianyi Zhou, and Haizhou Li. Minimizing the accumulated trajectory error to improve dataset distillation. November 2022. arXiv:2211.11004.

Raaz Dwivedi and Lester Mackey. Kernel thinning. May 2021. arXiv:2105.05842v10.

Diederik P Kingma and Max Welling. Auto-Encoding variational bayes. December 2013. arXiv:1312.6114v11.

Diederik P Kingma and Max Welling. An introduction to variational autoencoders. June 2019. arXiv:1906.02691.

Hae Beom Lee, Dong Bok Lee, and Sung Ju Hwang. Dataset condensation with latent space knowledge factorization and sharing. August 2022. arXiv:2208.10494.

Chun-Liang Li, Wei-Cheng Chang, Yu Cheng, Yiming Yang, and Barnabás Póczos. Mmd gan: Towards deeper understanding of moment matching network. *Adv. Neural Inf. Process. Syst.*, 30, 2017.

Guang Li, Ren Togo, Takahiro Ogawa, and Miki Haseyama. Compressed gastric image generation based on soft-label dataset distillation for medical data sharing. *Comput. Methods Programs Biomed.*, 227:107189, December 2022.

Haoyang Liu, Tiancheng Xing, Luwei Li, Vibhu Dalal, Jingrui He, and Haohan Wang. Dataset distillation via the wasserstein metric. November 2023. arXiv:2311.18531.

Songhua Liu, Kai Wang, Xingyi Yang, Jingwen Ye, and Xinchao Wang. Dataset distillation via factorization. pp. 1100–1113, October 2022.

Baharan Mirzasoleiman, Jeff Bilmes, and Jure Leskovec. Coresets for data-efficient training of machine learning models. In Hal Daumé Iii and Aarti Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 6950–6960. PMLR, 2020.

Timothy Nguyen, Zhourong Chen, and Jaehoon Lee. Dataset Meta-Learning from kernel Ridge-Regression. October 2020. arXiv:2011.0005.

Edgar Riba, Dmytro Mishkin, Daniel Ponsa, Ethan Rublee, and Gary Bradski. Kornia: an open source differentiable computer vision library for PyTorch. In *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 3674–3683. IEEE, March 2020.

Andrea Rosasco, Antonio Carta, Andrea Cossu, Vincenzo Lomonaco, and Davide Bacciu. Distilled replay: Overcoming forgetting through synthetic samples. March 2021. arXiv:2103.15851.

Noveen Sachdeva and Julian McAuley. Data distillation: A survey. January 2023. arXiv:2301.04272.

A Sajedi, Samir Khaki, Ehsan Amjadian, Lucy Z Liu, Y Lawryshyn, and K Plataniotis. DataDAM: Efficient dataset distillation with attention matching. *ICCV*, pp. 17051–17061, September 2023.

Kai Wang, Bo Zhao, Xiangyu Peng, Zheng Zhu, Shuo Yang, Shuo Wang, Guan Huang, Hakan Bilen, Xinchao Wang, and Yang You. CAFE: Learning to condense dataset by aligning FEatures. pp. 12196–12205, March 2022. arXiv:2203.01531.

Tongzhou Wang, Jun-Yan Zhu, Antonio Torralba, and Alexei A Efros. Dataset distillation. November 2018. arXiv:1811.10959.

Andreas Winter. Compression of sources of probability distributions and density operators. August 2002. arXiv:quant-ph/0208131.

Zeyuan Yin and Zhiqiang Shen. Dataset distillation in large data era. November 2023. arXiv:2311.18838.

Zeyuan Yin, Eric Xing, and Zhiqiang Shen. Squeeze, recover and relabel: Dataset condensation at ImageNet scale from a new perspective. June 2023. arXiv:2306.13092.

Bo Zhao and Hakan Bilen. Dataset condensation with differentiable siamese augmentation. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 12674–12685. PMLR, 2021.

Bo Zhao and Hakan Bilen. Synthesizing informative training samples with GAN. April 2022. arXiv:2204.07513.

Bo Zhao and Hakan Bilen. Dataset condensation with distribution matching. In *2023 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pp. 6514–6523. IEEE, January 2023.

Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. Dataset condensation with gradient matching. June 2020. arXiv:2006.05929.

Xiao Zhou, Renjie Pi, Weizhong Zhang, Yong Lin, Zonghao Chen, and Tong Zhang. Probabilistic bilevel coreset selection. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 27287–27302. PMLR, 2022a.

Yongchao Zhou, Ehsan Nezhadarya, and Jimmy Ba. Dataset distillation using neural feature regression. June 2022b. arXiv:2206.00719.

# A METHODOLOGY (EXTENDED)

## A.1 FURTHER DETAILS ON FIGURE 1

In this section, we provide details for Figure 1 in Tables 6 and 7. First, in Table 6 we compare our method with SOTA distillation methods at various storage costs. All methods perform distillation on a ConvNet architecture. Additionally, we evaluate our method and TESLA on ResNet18 to examine cross-architecture generalization. We also annotate storage cost with the equivalence of storing images as distilled dataset. Table 7 lists all the methods we used to generate Figure 1. We report IPC whenever applicable.

Table 6: **ImageNet-1K Performance comparison for SOTA methods aligned on storage budget** Storage cost is rounded (See Table 7 for exact storage breakdown). N/A: indicates the smallest size the method can distill into is larger than the corresponding size.

| | ConvNetD4 | | | | | ResNet18 | | |
|---|---|---|---|---|---|---|---|---|
| Storage Cost (MB) | Random | FrePo | DataDAM | TESLA | Ours | Random | TESLA | Ours |
| 25 (∼0.5IPC) | N/A | N/A | N/A | N/A | **11.5 (0.5)** | N/A | N/A | **9.7 (0.8)** |
| 50 (∼1IPC) | 0.6 (0.1) | 7.5 (0.3) | 2.0 (0.1) | 7.7 (0.2) | | 0.5 (0.1) | 6.2 (0.5) | |
| 100 (∼2IPC) | 0.8 (0.1) | 9.7 (0.2) | 2.2 (0.1) | 10.5 (0.2) | **17.4 (0.7)** | 0.6 (0.1) | 9.1 (1.5) | **16.0 (0.7)** |
| 500 (∼10IPC) | 3.6 (0.5) | | 6.3 (0.1) | 17.8 (1.3) | **20.3 (0.9)** | 3.6 (0.1) | 15.3 (1.3) | **18.2 (0.6)** |
| 3000 (∼50IPC) | 12.5 (1.5) | | 15.5 (0.2) | 27.9 (1.2) | | 15.3 (2.3) | 23.2 (0.9) | |

Table 7: **Details on storage cost breakdown, downstream task training cost and recovery accuracy distilled on ImageNet-1K and evaluated on ResNet18** Input Storage Cost refers to distilled synthetic images for prototype-based methods, and refers to latent prior and decoder for D3 (ours). DTC stands for Downstream task training cost defined in Section 3.1

| Method | Distill Arch | IPC | Input Storage (MB) | Label Storage (MB) | Accuracy (%) | DTC (min) |
|---|---|---|---|---|---|---|
| TESLA | ConvNet | 1 | 58 | 4 | 6.2 (0.5) | 8 |
| TESLA | ConvNet | 2 | 116 | 8 | 9.1 (1.5) | 17 |
| TESLA | ConvNet | 10 | 579 | 38 | 15.3 (0.8) | 60 |
| TESLA | ConvNet | 50 | 2897 | 238 | 23.2 (0.9) | - |
| SRe2L (orig) | ResNet18 | 1 | 583 | 1229 | 2.9 (0.2) | 50 |
| SRe2L (orig) | ResNet18 | 10 | 5848 | 6145 | 21.3 (0.6) | 250 |
| SRe2L (orig) | ResNet18 | 50 | 29764 | 30725 | 46.8 (0.2) | 600 |
| SRe2L (resize) | ResNet18 | 2 | 116 | 1229 | 1.2 (0.1) | 25 |
| SRe2L (resize) | ResNet18 | 10 | 579 | 6145 | 10.7 (0.5) | 40 |
| SRe2L (resize) | ResNet18 | 50 | 2897 | 30725 | 29.0 (0.5) | 175 |
| D3(Ours) | ConvNet | N/A | 17 | 4 | 9.7 (0.8) | 20 |
| D3(Ours) | ConvNet | N/A | 76 | 8 | 16.0 (0.7) | 40 |
| D3(Ours) | ConvNet | N/A | 440 | 38 | 18.2 (0.6) | 60 |

**TESLA** Cui et al. (2022): We replicated results for 1/2/10 IPC settings to produce the results above. For the 10 IPC setting, Cui et al. (2022) reported much lower performance on ResNet18 ($7.7\%$), and we used our reproduced results with higher accuracy($15.3\%$). We failed to repliate results for 50 IPC and obtained results directly from authors and therefore do not have downstream training cost estimate.

**SRe$^2$L** Yin et al. (2023): SRe$^2$L was originally implemented on higher resolution ImageNet-1K ($224 \times 224$). We replicated SRe$^2$L using hyperparameters provided by authors, which set of results are denoted as "original" above. We also re-implemented the pipeline on resized ImageNet-1K ($64 \times 64$), which set of results are denoted as "resized" above. In the original implementation, the image was saved in `.jpeg` format and the soft label and the augmentation parameter was saved in `fp16` format. We recomputed the storage cost assuming both images, augmentation parameters and soft labels were saved as floating point tensor format. In the default hyperparameter setting, SRe$^2$L generated 300 distilled labels for each prototype by applying augmentations to each prototype. As a result, both the labels and augmentation parameters needed to be saved, causing a rather large storage cost on label storage[4].

[4] 50 IPC distilled dataset Yin et al. (2023): `https://huggingface.co/datasets/zeyuanyin/SRe2L`

**Computing storage cost** To facilitate a direct and meaningful comparison between methods that use different distillation approaches, we quantify the total storage cost of each method, including all generated artifacts that are needed to reproduce the distilled dataset (decoder weights, images prototypes, soft labels, augmentations parameters etc). For prototypes and images, we measure their memory footprint when saved in single-precision floating-point tensor format (`fp32`). For our method, which distills into latent priors and decoder(s), we save all decoder and latent prior parameters again in `fp32` format (saving the entire `state_dict` for the decoder). Likewise, we measure memory footprint for distilled soft labels and augmentation parameters in the same `fp32` format. We report all computed storage cost in Megabytes (MB) rounded to closest integer value. While one could potentially achieve better image compression rates saving images into alternative formats (`.jpeg` for example), this would prevent an apples-to-apples comparison. Moreover, similar improvements could be achieved for distilled model parameters via quantization or compressed (e.g., `.zip`) storage. However, all these additional steps (if used) bring further complications to the discussion without bringing useful insight into using storage cost as a metric for dataset distillation. As a result, we decide the most natural and fair comparison is to assume floating point tensor format as the unified way of storing distilled data.

**Downstream training cost** All test-time runtime experiments are run using two NVIDIA A100-SXM4-40GB GPUs with data parallelism to ensure fair comparison. We use default parameters or hyperparameters provided whenever available. For TESLA, we used learned learning rate for Convnet, default learning rate for ResNet18, default learning rate scheduler, default training epochs (1000 epoch) and default batch size (i.e., batch size = IPC). For SRe$^2$L we used default learning rate, default training epochs (300 epoch) and default batch size. For D3 (our method), we used learned learning rate for Convnet, default learning rate for ResNet18, default batch size (i.e., batch size = latent prior per class) and default training epochs (2000 training epochs). We allowed early termination if all above methods converged earlier than the default epoch setting. However, for all methods, using default parameters only provides a rough estimate for the downstream training cost, and it may be possible to further optimize for downstream training cost with hyperparameter tuning. While it should only be used as a secondary metric to evaluate data distillation methods, we hope future work could provide more details on this metric when reporting results.

### A.2 Further details on distributional representation

**Formulation** Formally, for a class of models $f(\cdot, w) : \mathcal{X} \to \mathcal{Y}$ parameterized by their weights $w \in \mathbb{R}^D$, and a loss function $\mathcal{L}$, the objective of distillation can be phrased as finding $\mathcal{S}$ such that

$$\mathbb{E}_{\boldsymbol{x}\sim P_{\text{eval}}}\left[\mathcal{L}\left(f(\boldsymbol{x};\boldsymbol{w}^{\mathcal{S}}), y\right)\right] \simeq \mathbb{E}_{\boldsymbol{x}\sim P_{\text{eval}}}\left[\mathcal{L}\left(f(\boldsymbol{x};\boldsymbol{w}^{\mathcal{D}}), y\right)\right] \tag{1}$$

where $\boldsymbol{w}$ are the weights obtained by training (e.g., by empirical risk minimization) on either on the full training dataset $\mathcal{D}$ or on the distilled dataset $\mathcal{S}$:

$$\boldsymbol{w}^{\mathcal{D}} = \arg\min_{\boldsymbol{w}} \sum_{(\boldsymbol{x}_i, y_i)\in\mathcal{D}} \mathcal{L}(f(\boldsymbol{x}_i;\boldsymbol{w}), y_i), \qquad \boldsymbol{w}^{\mathcal{S}} = \arg\min_{\boldsymbol{w}} \sum_{(\boldsymbol{x}_i, y_i)\in\mathcal{S}} \mathcal{L}(f(\boldsymbol{x}_i;\boldsymbol{w}), y_i) \tag{2}$$

and $P_{\text{eval}}$ is the target data distribution that models are evaluated on (typically validation or test set). In the distributional representation, we consider the population counterparts of equation 2, i.e.,

$$\boldsymbol{w}^{\mathcal{D}} = \arg\min_{\boldsymbol{w}} \mathop{\mathbb{E}}_{(\boldsymbol{x}_i, y_i)\sim P} \mathcal{L}(f(\boldsymbol{x}_i;\boldsymbol{w}), y_i), \qquad \boldsymbol{w}^{\mathcal{S}} = \arg\min_{\boldsymbol{w}} \mathop{\mathbb{E}}_{(\boldsymbol{x}_i, y_i)\sim Q} \mathcal{L}(f(\boldsymbol{x}_i;\boldsymbol{w}), y_i). \tag{3}$$

Instead of finding the optimal distilled data set $\mathcal{S}$, we need to find a synthetic distribution $Q$ which, as before, leads to comparable predictive performance for $f$ on the target distribution $P_{\text{eval}}$. To make the problem tractable, we use a family of distributions $Q_\xi$ with parameter set $\xi$. The use of parameterizable distributions in turn allows us to formulate the problem as an optimization over the finite-dimensional parameter space rather than the infinite-dimensional space of distributions.

Concretely, we assume a Gaussian prior distribution in latent space $\mathcal{Z}$, and a posterior distribution $Q_\xi(\boldsymbol{x}|\boldsymbol{z})$ that can be parameterized by a decoder. In this formulation, parameters $\xi$ include Gaussian priors ($\mu$'s, and $\Sigma$'s) and decoder parameters $\theta$. The distilled distribution can be represented in a variational form:

$$Q_\xi(\boldsymbol{x}) = \int Q_\xi(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})\,\mathrm{d}\boldsymbol{z}, \text{ where } p(\boldsymbol{z}) \sim \mathcal{N}(\mu, \Sigma),\ Q_\xi(\boldsymbol{x}|\boldsymbol{z}) = g_\theta(\boldsymbol{z}).$$

In Section 3.1, we propose to use total storage cost instead of IPC as an evaluation metric for data distillation. Storage cost as a metric can be applied to our distributional representation as well. Specifically, we argue that when we distill into distributions, "distillation" is satisfied if the storage footprint as discussed is sufficiently small. Furthermore, we also argue that to achieve information compression, the effective number of samples from $Q$ on which a model needs to be trained is comparable, or lower, than that of training on the original dataset $\mathcal{D}$.

In particular, we seek to avoid the two trivial corner-cases $Q_\xi = \frac{1}{N}\sum_{i=1}^{N}\delta_{\boldsymbol{x}_i}$, i.e., the uniform empirical measure associated with training set $\mathcal{D}$, and $Q_\theta \approx P_\mathcal{D}$, i.e., learning the full distribution of the original data — a much harder problem to solve. Using a Gaussian prior has the advantage of encoding information into a latent (typically lower-dimensional) space and provide a framework to ensure only relevant information is preserved in the distributional representation.

**Scaling Up** There are multiple ways to scale up the amount of information encoded in the distilled distribution with a larger storage budget and more compute:

(i) We can use multiple Gaussian priors for each class. We refer to the number of Gaussian priors as latent priors per class. By using multiple distributional priors, we are essentially assuming that each class follows a multi-modal distribution. For simplicity, we assume that of these prior distributions $\mathcal{N}\left(\mu_c^i, \Sigma_c^i\right)$ is equally likely (i.e., uniform distribution among all Gaussian priors).

(ii) We can increase the dimension of the latent Gaussian distribution to allow more information to be encoded for each prior.

(iii) We can increase the size of the decoder $g_\theta$ to allow more shared information to be stored for the entire dataset.

Empirically, we find that the most effective way to scale up the size of distilled dataset and achieve higher recovery accuracy is to first increase the number of latent priors per class. Once we exceed a certain number of priors per class, we also need to use a larger decoder and higher latent dimension to achieve higher distillation quality.

### A.3 Training Objective

Building on the foundations of existing data distillation techniques, we introduce a learning objective compromised of two distinct terms. The first term is derived from Matching Training Trajectories (MTT) proposed by Cazenavette et al. (2022). The second term in our objective aims to minimize the Maximum Mean Discrepancy (MMD) between the true dataset and our learned dataset distribution. Different from the formulation used in DM (Zhao & Bilen, 2023), we use a set of Reproducing Hilbert Kernels (RHKS) for the MMD computation to fully leverage the power of MMD. We first map the pixel space to latent feature space using trained experts. For model simplicity and training efficiency purposes, we recycle the experts used in MTT to generate feature mappings. We then use a mixture of Radial Basis Function (RBF) kernels $k(x, x') = \sum_{q=1}^{K} k_{\sigma_q}(x, x')$, where $k_{\sigma_q}$ represents a Gaussian kernel with bandwidth $\sigma_q$. We choose a mixture of $K = 5$ kernels with bandwidths $\{1, 2, 4, 8, 16\}$. The hyperparameter choice is inspired by MMD GANs (Bińkowski et al., 2018; Li et al., 2017). See below for a full description of the training objective.

**MTT Loss** Expert trajectories are training trajectories generated from training neural networks on the full training set. At each distillation step, we initialize a student network that has the same architecture as the experts. The student network's initialization weight $\boldsymbol{w}^Q$ is sampled from the experts training trajectory by randomly selecting an expert and a random iteration $t$, such that $\boldsymbol{w}_t^Q = \boldsymbol{w}_t^\mathcal{P}$. We perform $N$ gradient updates on the student network using data drawn from the distilled distribution:

$$\text{for}\, n = 0....N-1 : \boldsymbol{w}_{t+n+1}^Q = \boldsymbol{w}_{t+n}^Q - \alpha\nabla\mathcal{L}(Q; \boldsymbol{w}_{t+n}^Q), Q \sim Q_\mathcal{S}^\theta$$

We then collect expert parameters from $M$ training updates after iteration $t$, which denote as $\boldsymbol{w}_{t+M}^\mathcal{P}$. The distance between the updated student parameters and the updated expert parameters is quantified

using normalized squared error:

$$D_{\texttt{MTT}} = \frac{\|\boldsymbol{w}_{t+N}^{Q} - \boldsymbol{w}_{t+M}^{\mathcal{D}}\|_2^2}{\|\boldsymbol{w}_{t}^{\mathcal{P}} - \boldsymbol{w}_{t+M}^{\mathcal{D}}\|_2^2}$$

**MMD Loss** We use a set of Reproducing Hilbert Kernels (RHKS) for the MMD computation to fully leverage the power of MMD. Since we only have access to the distilled distribution $Q_{\mathcal{S}}^{\theta}$ but not the training data distribution $P$, we use the empirical MMD measure: In general, given random variable $X = \{x_1, ..., x_n\} \sim \mathbb{P}$ and $Y = \{y_1, ..., y_m\} \sim \mathbb{Q}$, the unbiased estimator of the MMD measure is Li et al. (2017):

$$\widehat{\mathrm{MMD}}^2(X, Y) = \frac{1}{\binom{n}{2}} \sum_{i \neq j}^{n} k(x_i, x_j) - \frac{1}{mn} \sum_{i=1}^{n} \sum_{j=1}^{m} (x_i, y_j) + \frac{1}{\binom{m}{2}} \sum_{i \neq j}^{m} k(y_i, y_j) \tag{4}$$

We also map the pixel space to latent feature space. For model simplicity and training efficiency purposes, we recycle the experts used in MTT to generate feature mappings, and denote them as $\psi(\cdot)$. Inspired by MMD GANs (see Li et al. (2017); Bińkowski et al. (2018)), we use a mixture of Radial Basis Function (RBF) kernels $k(x, x') = \sum_{q=1}^{K} k_{\sigma_q}(x, x')$, where $k_{\sigma_q}$ represents a Gaussian kernel with bandwidth $\sigma_q$. We choose a mixture of $K = 5$ kernels with bandwidths $\{1, 2, 4, 8, 16\}$.

To encourage distribution matching with the original dataset, we penalize large MMD:

$$\mathcal{L}_{\texttt{MMD}} = \sum_{c=1}^{C} \widehat{\mathrm{MMD}}^2(\psi(\mathcal{D}_c), \psi(\mathcal{S}_c)), \tag{5}$$

where the $\widehat{\mathrm{MMD}}^2$ computation is defined in Eqn. 4. $\mathcal{D}_c$ and $\mathcal{S}_c$ simply refers to the subset of the training data or distilled data with class label $c$.

## B Decoder Architecture

Our decoder is adopted from the decoder part of the VAE designed by Kingma & Welling (2019), with small modifications. First, we project the latent $z$ in to a $k$ dimension feature vector, which is then fed into a sequence of 2D `ConvTranspose` blocks. Each of the decoder block contains a `ConvTranspose` layer followed by a `BatchNorm` layer and a `LeakyReLU` activation. For larger decoder, we increase the latent dimension, and consequently the size of `ConvTranspose` blocks. After the those blocks, there is a 2D convolutional layer followed by a `tanh` activation. The exact dimension of the convolution layer differs by image output size. The original VAE was designed only for images with size $32 \times 32$, and used only 3 blocks. We also increase the number of deconv blocks for larger datasets.

Table 8: **Architecture and hyperparameter details for the decoders we used** *Total parameters* are counted in millions. *#Blocks* indicates the number of convolutional blocks.

| Size | # Blocks | Latent Dimension | Total Params | Output Image Size |
|---|---|---|---|---|
| S | 5 | 64 | 0.75M | $32 \times 32 \times 3$ |
| S | 6 | 64 | 0.75M | $64 \times 64 \times 3$ |
| M | 5 | 1028 | 5.7M | $64 \times 64 \times 3$ |
| L | 5 | 2048 | 6.3M | $64 \times 64 \times 3$ |

## C Details on experiment setup

In this section, we provide a detailed description on experiment setups for all experiment resuls presented in the paper.

**Dataset** **CIFAR-10** contains 50,000 training images from 10 classes, each with dimensions of $32 \times 32 \times 3$. **CIFAR-100** contains same number of images but more classes: 100 classes with 500 images each in the training data with dimension $32 \times 32 \times 3$. **TinyImageNet** consists of 100,000 images distributed across 200 classes. The images within TinyImageNet are characterized by larger dimensions, measuring $64 \times 64 \times 3$. **ImageNet-1K** contains 1000 classes with around 1300 classes

each, totalling 1.2 million images. We resized the images to $64 \times 64 \times 3$, aligning with prior works (Cui et al., 2022; Sachdeva & McAuley, 2023; Zhou et al., 2022b). Finally, we also include two known ImageNet subsets: **ImageNette** and **ImageWoof**. In line with established practices from prior work, we resize the images within both subsets of ImageNet to dimensions of $128 \times 128 \times 3$. Each subset comprises 10 classes in their respective training sets, at total size of around 10k images.

**Dataset preprocessing** For all three datasets, only a simple channel-based mean-variance scaling is performed as the preprocessing step. For CIFAR-10 we perform ZCA whitening as done in all data distillation work (Nguyen et al., 2020) using Kornia implementation with default parameters ((Riba et al., 2020). To generate experts used in MTT, we also perform random simple augmentations to the images, including rotations, flip, crop, and color changes. The preprocessing step is chosen to mirror the baselines we make direct comparisons to.

**Student network architecture** The student network is a neural network consists of multiple ConvNet blocks, and we call them ConvNet. The ConvNet configuration consists of multiple convolutional blocks, each housing a convolutional layer, a normalization layer, `ReLU` activation, and an average pooling layer. For larger datasets, we increase the number of convolutional blocks used in the ConvNet. For CIFAR10 we use ConvNet with 3 convolutional blocks, and for TinyImageNet and ImageNet-1K we use 4 convolution blocks. For ImageNet subsets, we use 5 convolutional blocks. In our MMD objective, we use the features generated by those convolutional blocks to compute MMD. Finally, a linear layer with Softmax activation is used to map the features generated by convolutional blocks into class prediction.

**Training** The distillation time is not the primary concern for data distillation tasks since it only needs to be done once for all downstream tasks. However, methods that are overly expensive to train might become infeasible when distilling large datasets. Because we compute and back propagate on both MTT and MMD losses, our compute time is comparable to both method combined. For CIFAR-10, CIFAR-100, ImageNette and ImageWoof, our method converges in fewer than 10,000 steps, usually taking less than 10 GPU hours on NVIDIA A100-SXM4-40GB. For TinyImageNet, our method converges around 10,000 steps, totalling around 20 GPU hours. Finally, for federated distilation on ImageNet, since we decompose into distillation sizes comparable to TinyImageNet, the training time is similar.

For evaluation, we use SGD optimizer with momentum 0.9 and weight decay $5 \times 10^{-4}$. We only allow hyper-parameter tuning on the learning rate, number of epochs and we train student networks until convergence.

**Decoder Hyperparameters** In table 9, we list the hyper-parameters we used for each setting and the exact storage costs to store the latent priors and decoders.

Table 9: **Details on decoder hyper-parameters for all experiments** *Decoder*: refer to Table 8, # *Decoders*: more than one decoders for federated distillation when we aggregate from subtasks. *LPC*: Latent Priors per Class refers to the number of Gaussian distributions we used to represent each class.

| Dataset | Decoder | # Decoders | LPC | Total Storage (MB) |
|---|---|---|---|---|
| CIFAR10 | S | 1 | 10 | 3.4 |
| CIFAR10 | S | 1 | 50 | 7.8 |
| CIFAR100 | S | 1 | 2 | 4.2 |
| CIFAR100 | M | 1 | 5 | 9.7 |
| ImageNette | S | 1 | 5 | 3.9 |
| ImageWoof | S | 1 | 5 | 3.9 |
| TinyImageNet | S | 1 | 2 | 3.4 |
| TinyImageNet | M | 1 | 2 | 10 |
| TinyImageNet | L | 1 | 10 | 56 |
| ImageNet-1K | S | 5 | 1 | 17 |
| ImageNet-1K | M | 2 | 2 | 76 |
| ImageNet-1K | L | 5 | 10 | 440 |

## D Additional Results

Here we show additional results on CIFAR-100, CIFAR-10, ImageNette and ImageWoof, and compare our methods to methods that distill into image space: MTT (Cazenavette et al., 2022) and FTD (Du et al., 2022) as well as methods that distill into latent space HaBa (Liu et al., 2022),

LinBa (Deng & Russakovsky, 2022) and GLaD (Cazenavette et al., 2023). We report CIFAR-100 results in Table 10, CIFAR-10 results in Table 11, and ImageNette and ImageWoof results in Table 12. Additionally, we also report cross-architecture results for CIFAR-10 (see Table 13) and for ImageNette and ImageWoof (See Table 14, 15).

Table 10: **CIFAR-100 distilled and evaluated on ConvNet**

| Storage Cost (MB) | MTT | FTD | LinBa | D3(Ours) |
|---|---|---|---|---|
| 5 (∼ 1IPC) | 24.3 (0.3) | 25.2 (0.2) | 34.0 (0.4) | **37.3 (0.7)** |
| 10 (∼ 10IPC) | 40.1 (0.4) | 43.4 (0.3) | | **46.8 (0.4)** |
| 50 (∼ 50IPC) | 47.7 (0.3) | 50.7 (0.2) | | |

Table 11: **CIFAR-10 distilled and evaluated on ConvNet**

| Storage Cost (MB) | MTT | HaBa | LinBa | Ours |
|---|---|---|---|---|
| 0.5 (∼1IPC) | 46.3 (0.8) | 48.3 (0.8) | 66.4 (0.4) | |
| 5 (∼10IPC) | 65.3 (0.7) | 69.9 (0.4) | 72.2 (0.4) | **71.8 (0.2)** |
| 25 (∼50IPC) | 71.6 (0.2) | 74.0 (0.2) | 73.6 (0.5) | **74.4 (0.3)** |

Table 12: **ImageNette and ImageWoof distilled and evaluated on ConvNet** N/A: indicates the distillation size is smaller than the minimum size the method can distill.

| **Dataset** | Storage (MB) | **Method** | | | |
|---|---|---|---|---|---|
| | | MTT | HaBa | FTD | Ours |
| **ImageNette** | 5(∼0.5IPC) | N/A | | N/A | **71.04 (0.71)** |
| | 10(∼1IPC) | 47.7 (0.9) | 51.92 (1.65) | 52.2 (1.0) | |
| | 100(∼10IPC) | 63.0 (1.3) | 64.72 (1.60) | 67.7 (0.7) | |
| **ImageWoof** | 5(0.5IPC) | N/A | | N/A | **41.60 (1.15)** |
| | 10(∼1IPC) | 28.6 (0.8) | 32.40 (0.67) | 35.8 (1.8) | |
| | 100(∼10IPC) | 35.8 (1.8) | 38.60 (1.26) | 38.8 (1.4) | |

Table 13: **CIFAR-10 cross-architecture generalization results** we scaled down our distilled dataset by reducing number of latent priors per class such that our performance on ConvNet aligns with baseline (MTT)

| | | **Evaluation Model** | | | |
|---|---|---|---|---|---|
| **Method** | Storage Cost (MB) | ConvNet | ResNet18 | VGG11 | AlexNet |
| MTT | 5 (∼10IPC) | 65.3 (0.7) | 46.4 (0.6) | 50.3 (0.8) | 34.2 (2.6) |
| D3(ours) | 3 (∼10IPC) | **66.64 (0.26)** | **61.57 (0.48)** | **59.70 (0.48)** | **54.56 (0.74)** |

Table 14: **ImageNette and ImageWoof cross-architecture generalization results** Unseen architecture results from averaging ResNet18, VGG11, AlexNet, Vision Transformer.

| | | **ImageNette** | | **ImageWoof** | |
|---|---|---|---|---|---|
| **Method** | Storage Cost (MB) | ConvNet | Unseen | ConvNet | Unseen |
| MTT | 10(∼1IPC) | 47.9 (0.9) | 24.1 (1.8) | 28.6 (0.8) | 16.0 (1.2) |
| GLaD MTT | 10(∼1IPC) | 38.7 (1.6) | 30.4 (1.5) | 23.4 (1.1) | 17.1 (1.1) |
| GLaD DC | 10(∼1IPC) | 35.4 (1.2) | 31.0 (1.6) | 22.3 (1.1) | 17.8 (1.1) |
| GLaD DM | 10 (∼1IPC) | 32.3 (1.7) | 21.9 (1.1) | 21.1 (1.5) | 15.2 (0.9) |
| D3(ours) | 5 (∼0.5IPC) | **71.04 (0.7)** | **48.95 (1.3)** | **41.60 (1.2)** | **28.82 (0.93)** |

Table 15: **ImageNet Subset cross-architecture performances breakdown** Per-architecture breakdown for the unseen average listed in Table 14

| | | **Evaluation Model** | | | | |
|---|---|---|---|---|---|---|
| Dataset | Storage Cost (MB) | ConvNet | ResNet | VGG11 | AlexNet | ViT |
| ImageNette | 5(∼0.5IPC) | 71.04 (0.71) | 47.28 (0.94) | 64.80 (1.2) | 49.20 (1.45) | 34.5 (1.5) |
| ImageWoof | 5(∼0.5IPC) | 41.60 (1.15) | 28.04 (0.51) | 35.48 (1.07) | 29.28 (1.4) | 22.48 (0.73) |

## E SAMPLES FROM DISTILLED DISTRIBUTION

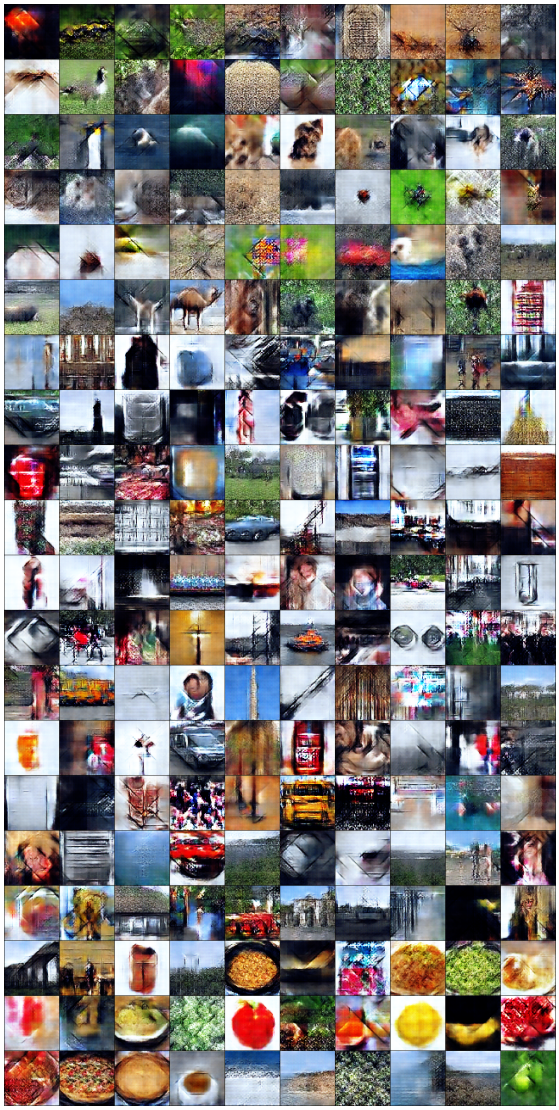

Figure 8: Samples from Distilled Distribution on TinyImageNet under 100MB storage cost. 1 latent priors per class visualized

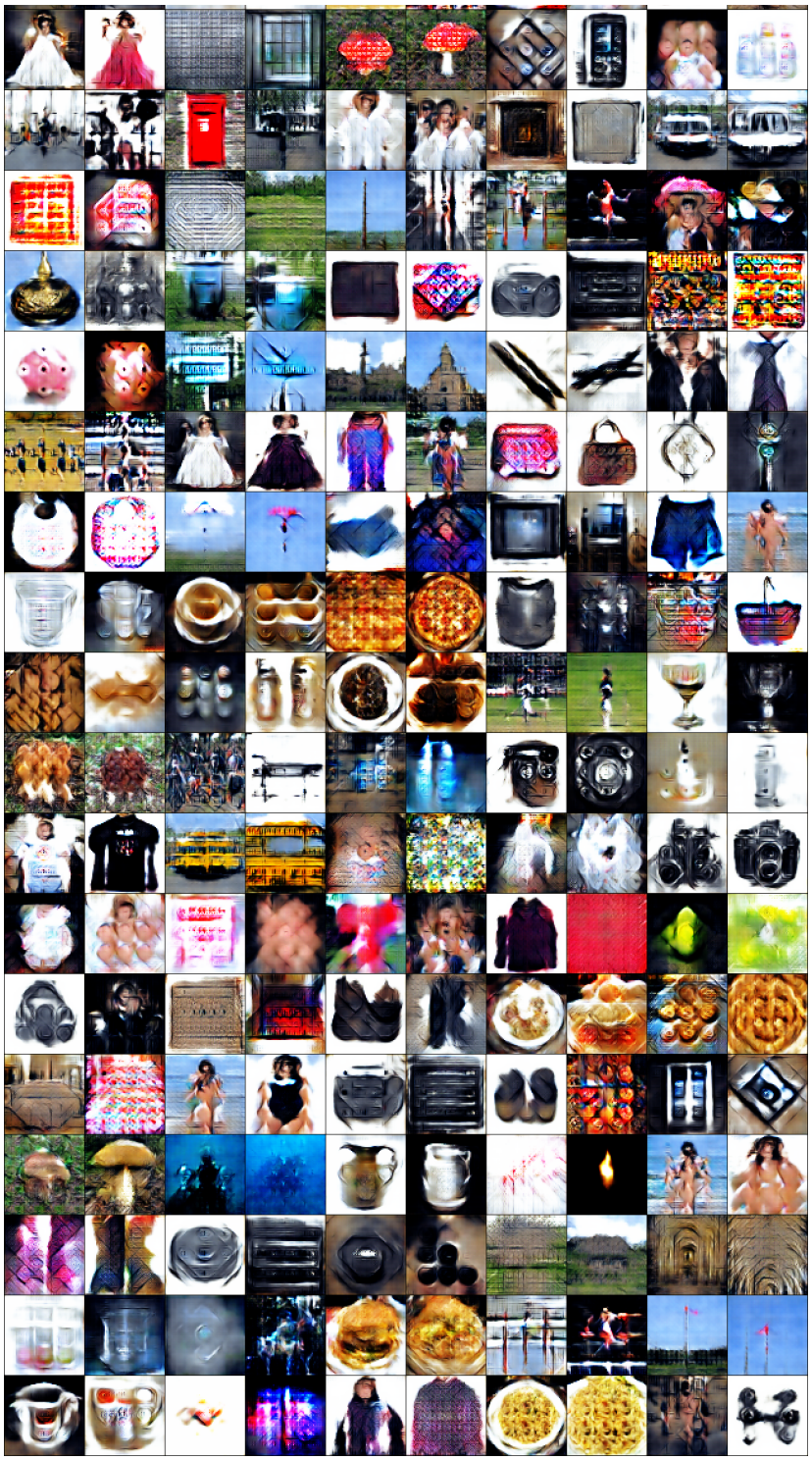

Figure 9: Samples from Distilled Distribution on ImageNet-1K under 100MB storage cost. 2 latent priors per class visualized

Figure 10: Samples from Distilled Distribution on ImageNet-1K under 100MB storage cost (continued). 2 latent priors per class visualized